2010 International Conference of Information Science and Management Engineering

Mining Web Log Sequential Patterns with Layer Coded Breadth-First Linked WAP-Tree

Lizhi Liu[1][2]
Hubei Province Key Laboratory of Intelligent Robot
Wuhan Institute of Technology
School of Computer Science and Engineering
Wuhan Institute of Technology
Wuhan Hubei, China
llz73@163.com

Jun Liu[1][2]
Hubei Province Key Laboratory of Intelligent Robot
Wuhan Institute of Technology
School of Computer Science and Engineering
Wuhan Institute of Technology
Wuhan Hubei, China
lorime@163.com

***Abstract*—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. An important application of sequential mining techniques is web usage mining, for mining web log accesses, which the sequences of web page accesses made by different web users over a period of time, through a server, are recorded. Web access pattern tree (WAP-tree) mining is a sequential pattern mining technique for web log access sequences. This paper proposes a more efficient approach for using the BFWAP-tree to mine frequent sequences, which reflects ancestor-descendant relationship of nodes in BFWAP tree directly and efficiently. The proposed algorithm 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. Experiments show huge performance gain over the WAP-tree technique.**

Keywords:* *sequential patterns; Web usage mining;BFWAP-tree mining; layer codes

I. INTRODUCTION

Mining on WWW is classified into three areas, (1)Web usage mining, (2)Web structure mining and (3) Web content mining. Web usage mining tries to find user's behavior in Web accesses and is mostly approached from access log analysis. Web structure mining finds linkage structure of the complex Web structure, which often means the hypertext structure. The linkage analysis may find some community on Web. Web content mining aims automatic extraction of desirable information from Web. It studies to use the Web as a big dictionary or databases. It is mainly approached from the text analysis of the HTML documents through WWW[1-4].

PLWAP-tree[5], a recent technique to mining sequential patterns from web logs, avoids generating long lists of candidate sets to scan support for[6][7]. However, PLWAP-tree algorithm can't denote ancestor-descendent relationship of nodes in PLWAP-tree clearly. In this paper, a layer code is assigned to nodes in WAP tree and the linkage of frequent event is chained by a Breadth-First algorithm. These two changes can improve the process of mining WAP tree.

II. MINING PROCESS OF PLWAP TREE

Each of this WAP tree's nodes has a binary position code assigned for directly mining the sequential patterns without re-constructing the intermediate WAP trees. The technique presents a much better performance than that achieved by the WAP-tree technique. The web log access sequence database (WASD) in Table 1 is used to show how to construct the PLWAP-tree and do PLWAP-tree mining.

Table 1. Sample WASD of PLFWAP-tree

TID	Web access sequence	Frequent subsequence
100	abdac	abac
200	eaebcac	abcac
300	babfaec	babac
400	afbacfc	abacc

Construct PLWAP Tree

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 structure, a tree is usually transformed into its equivalent binary tree, which has a fixed number of child nodes.

Rule 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 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 of β, where x is the* ((*number of bits in the position code of α*)

+ 1).

The first step of the PLWAP algorithm is to scan the WASD once to obtain the supports of the events in the event set $E = \{a, b, c, d, e, f\}$ and store the frequent 1-events in the header List, L as $\{a:4, b:4, c:4\}$. The events d:1, e:2, f:2 have supports less than the minimum support of 3 and thus, are not frequent events (supposing the minimum support is 75%). The second step in the PLWAP algorithm causes the WASD database to be scanned the second time, using only the frequent events in each transaction, and constructing pre-order linkages of the header nodes for frequent events *a, b,* and *c*. The algorithm PLWAP Construct is called by the main algorithm to construct the PLWAP tree. The first step creates a root node for the PLWAP tree, and for each access sequence in the second column of the WASD (Table 1), it removes all non-frequent events *d*, *e*, *f* to create the frequent events in the transactions in their sequential order shown as the third column of Table 1. Thus, for the first sequence *abdac*, the frequent sequence *abac* is obtained and for the second sequence, *eaebcac*, the frequent sequence *abcac* is obtained. These frequent sequences of the WASD transactions are used for constructing the PLWAP tree, one frequent sequence after the other. Thus, the PLWAP Construct 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. Since there is no such node yet, an *a* node is inserted as the leftmost child of the root as shown in figure 1. 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 since no *b* node following an *a* node exists yet. Since it is the first of its kind, its count is 1 and its position code is 11 because Rule 2.1 says to append 1 to the position code of its parent node to obtain the position code of the leftmost child. 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 (node label: count: position code). Next, the second frequent sequence, *abcac* is inserted starting from root. Since the root has a child labeled *a*, thus *a*'s count is increased by 1. Now the node (*a*:1:1) is changed to (*a*:2:1). Similarly, we now have (*b*:2:11). The next event c does not match the existing node *a*. Thus, a new node is created. The new node is not the first child of its parent (*b*:2:11), thus, its position code is the position code of its closest left sibling (*a*:1:111) with 0 appended. Thus, the position code of the new node is 1110. Finally, the new node can be represented as (*c*:1:1110). The next two events are newly created as (*a*:1:11101) and (*c*:1:111011) respectively. This process keeps running until there is no more access sequence in WASD. The sequence *babac* and *abacc* also insert into the tree follow the same way described above. Now, that the PLWAP tree is constructed, the third step in the algorithm PLWAP Construct 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. The algorithm keeps running until the whole PLWAP-tree has been traversed. Figure 1 shows the completely constructed PLWAP tree with the pre-order linkages.

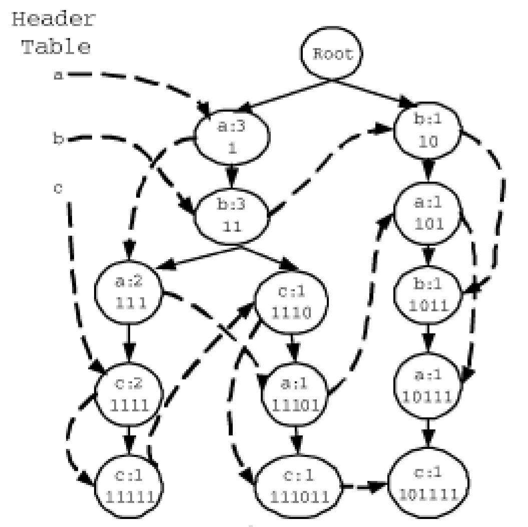

Figure 1. PLWAP tree

Mine PLAP Tree

The algorithm starts by mining the tree in figure 2 for the first element in the header linkage list, *a* following the *a* link to find the first concurrencies 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), and more than the minimum support of 3. 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}. 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 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 position codes of the nodes are used to identify which nodes of each suffix tree are descendants of the first event node e_i in the same subtree so that their counts are ignored. The position codes are also used to quickly identify the e_i nodes on a current suffix trees set that belong to a different subtree so that their counts can contribute to the total support count of e_i. 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 2, 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. 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 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 the same as above. Finally, we have the frequent sequence set *{ a, aa, aac, ab, aba, abac, abc, ac, b, ba, bac, bc, c}*

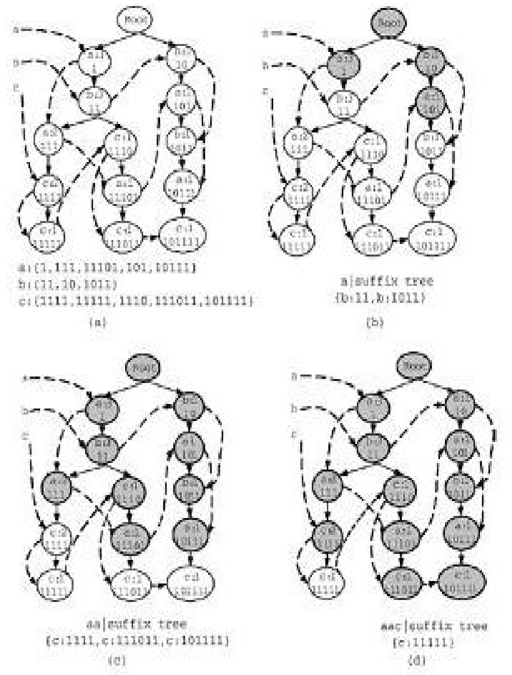

Figure 2. Mining PLWAP-tree to find frequent sequence

III. THE BREAD-FIRST WAP TREE

The main problem of the PLWAP is how the first occurrence of node a:3:1 and a:1:101 can be found efficiently. From the head table the PLWAP tree shown in Figure 2, event a chain is {a:3:1, a:2:111, a:1:11101, a:1:101, a:1:10111}. Although the ancestor- descendant relationship of nodes can be discovered by property 2.1, but how can we find the first two descendants of root from the event a chain. Why the first two descendants is a:3:1 and a:1:101, but not a:3:1 and a:2:111. In this paper, a layer code is assigned to nodes in WAP tree and the linkage of frequent event is chained by a Breadth-First algorithm. These two changes can improve the process of mining WAP tree.

Layer Code Assignment Mechanism

Layer code reflects ancestor- descendant relationship of nodes in WAP tree directly and clearly. Suppose any ancestor node in WAP tree has no more than n (n<100) children node, layer code can be assigned to nodes in WAP tree easily and exclusively. Root node has a layer code which is 0, the first child of node root (from left to right) is coded as 001, the second as 002 and so on. The first child of node coded as 001 is coded as 00101, the second as 00102 and so on. So the main characters of layer code are: 1) the length of ancestor nodes is longer than descendant node. 2) The code of ancestor node is fully contained by descendant node code.

Rule 2. *Suppose any ancestor node in WAP tree has no more than n (n<100) children node Given a WAP-tree with some nodes, the layer code of each node can simply be assigned following the rule that the root has 0 layer code, 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 node plus its position (from left to right) 01-99.*

Property 2. *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.*

Breadth-First Algorithm

The event linkage created by Pre-Order traversal mechanism can't reflect ancestor-descendant relationship directly and clearly. In figure 1, follow the linkage started in head table, the chain of event a, {a:3:1, a:2:111, a:1:11101, a:1:101, a:1:10111} is gotten. In this chain, a:2:111 appears before a:1:101, but a:1:101 is the second level of root, 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 all chain. Bread-First traversal mechanism can reflect the level relationship of nodes in WAP tree straightforwardly. Starting from the root using Breadth-First traversal mechanism to create queue, we first add first child of root (a:3:001) into the a-queue. Then, visiting the second child of root (b:1:002), (b:1:002) will be added to the b-queue. Since (a:3:001) is visited before (b:1:002), so the child node (b:3:00101) of (a:3:001) is visited prior to the child node (a:1:00201) of (b:1:002). To implement Breadth-First mechanism, data structure QUEUE is referred. All child nodes of root enter the queue from left to right, then according to the FIFO (first in first out) rule, the first element of the queue is accessed, after adding all its children nodes in the queue, the element is deleted from the queue, then the second element of the queue is dealt the same as the first element and so on.

BFWAP tree after adding the Breadth-First linkage is show as figure 3. From the linkage, the chain of event a is gotten as (a:3:001, a:1:00201, a:2:0010101, a:1:001010201, a:1:002010101). In the chain of event a, we know the level of a:3:001 is 1, a:1:00201 is 2, a:2:0010101 is 3, a:1:001010201 and a:1:002010101 is 4. To find the first occurrence event a nodes of root, we scan the chain of event a from left to right manner, a:3:001 is definitely the first occurrence of root because its level is 1, in the chain of event a a:2:0010101 and a:1:001010201 are removed because they are descendant node of a:3:001. a:1:00201 is also the first occurrence of root, although its level is 2, but it is not the child node of a:3:001 based on property 2.2, a:1:002010101 is also removed the same reason as above. So a:3:001 and a:1:00201 is the first occurrence of root. To discover 2- sequences aa from the suffix tree of 1-sequence a shown as figure (b), we get a nodes from the suffix tree in a event's chain are { a:2:0010101, a:1:001010201, a:1:002010101}, their counts are all contributed to the 1-sequenec a, because a:1:001010201 is not the child node of a:2:0010101, and a:1:001010201 and a:1:002010101 are at the same level. To discover 3-sequences aac from the suffix tree of 2-sequence aa shown as figure (c), we get c nodes from the suffix tree in c event's chain are {c:2:001010101, c:1:00101010101,c:1:00101020101, c:1:00201010101}, c:1:00101010101 is the child node of c:2:001010101, c:1:00101020101 is not the child node of c:2:001010101 and the same level of c:1:00201010101, so only c:1:00101010101's count is not contributed to the 2-sequences aa.

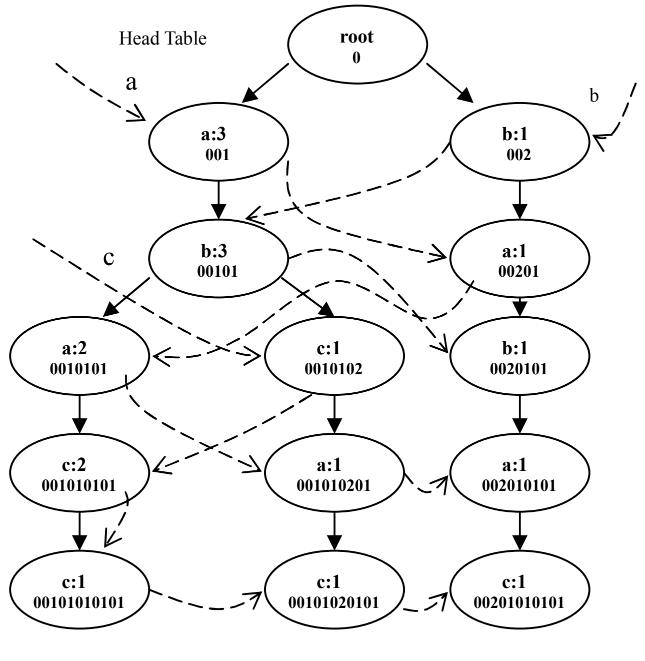

Figure 3. Breadth-First WAP tree

IV. Conclusions

This paper presents an improved algorithm (BFWAP) for efficiently mining sequential patterns from web log. The BFWAP algorithm adapts the WAP-tree structure for storing frequent sequential patterns to be mined. However, to improve on mining efficiency, the paper proposes to find layer patterns instead of prefix patterns as done by PLWAP-tree mining. Moreover, in order to find ancestor-descendant relationship of node in WAP tree efficiently, Breadth-First frequent header node linkages and layer codes are proposed. While the Breadth-First linkage provides a way to traverse the event queue without going backwards, 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 and more direct and efficient than PLWAP algorithm. Thus, it avoids re-constructing WAP-tree recursively. The experiments show that mining web log using BFWAP algorithm is much more efficient than with WAP-tree and GSP algorithms, especially when the average frequent sequence becomes longer and the original database becomes larger.

Acknowledgement

It is a project supported by National Natural Science Foundation of China (60603074) and 973 National Basic Research Program of China (2004CB318203).

References

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. 1994 Int'l Conf. Very Large Data Bases (VLDB '94), pp. 487-499, Sept. 1994.
[2] Kosala and Blockeel, "Web mining research: A survey," SIGKDD: SIGKDD Explorations: Newsletter of the Special Interest Group (SIG) on Knowledge Discovery and Data Mining, ACM, Vol. 2, 2000.
[3] S. K. Madria, S. S. Bhowmick, W. K. Ng, and E.-P. Lim, "Research issues in web data mining," in Data Warehousing and Knowledge Discovery, 1999, pp. 303-312.
[4] M. N. Garofalakis, R. Rastogi, S. Seshadri, and K. Shim, "Data mining and the web: Past, present and future," in ACM CIKM'99 2nd Workshop on Web Information and Data Management (WIDM'99), Kansas City, Missouri, USA, November 5-6, 1999, C. Shahabi, Ed. ACM, 1999, pp. 43-47.
[5] C.I. EZEIFE, and YI LU, "Mining Web Log Sequential Patterns with Position Coded Pre-Order Linked WAP-Tree." Proc. 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands. Data Mining and Knowledge Discovery, pp. 5-38, Oct. 2005
[6] Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., and Hsu, M.-C. 2000. FreeSpan: Frequent pattern-projected sequential pattern mining. In Proceedings of the 2000 Int. Conference on Knowledge Discovery and Data Mining (KDD'00). Boston, MA, U.S.A., pp. 355–359.
[7] Han, J., Pei, J., Yin, Y., and Mao, R. 2004. Mining frequent patterns without candidate generation: A frequentpattern tree approach. International Journal of Data Mining and Knowledge Discovery. Kluwer Academic Publishers, 8(1): 53–87.