A Web Log Frequent Sequential Pattern Mining Algorithm Linked WAP-Tree

Zhengyu Zhu[1], Meiyu Zheng[1*], Yihan Wu[2]

[1] College of Computer Science, Chongqing University, Chongqing 400030, China.
[2] Beijing Normal University, Beijing 100000, China.

* Corresponding author. Tel.: 18680926090; email: qq616079491@gmail.com
Manuscript submitted June 1, 2015; accepted August 28, 2015.
doi: 10.17706/jsw.10.10.1228-1234

Abstract: Web log frequent sequence pattern mining is an important field of Web log mining and of discovering interactive frequent sequence pattern between users and websites. It is easy to analyse users' access sequence patterns by utilizing these sequence patterns and it is meaningful to build an intelligent website by mining Web log frequent sequential patterns. The PREWAP algorithm proposed in the paper makes full use of creating descendant of new node while constructing its WAP-tree and results in the serial number of pre-visiting while constructing its header table to judge the relationship of all nodes. The experiments show that mining web log using the PREWAP algorithm is much more efficient than using the current PLWAP, NGCWAP and WAP algorithms, especially on memory consumption.

Key words: Data mining, web log, frequent pattern, sequential pattern; WAP-tree.

1. Introduction

With the rapid development of Internet, Web generates a huge size of log data and collects a lot of users' data [1]. It's significant to mine some meaning sequences or item sets based on these datum [2], [3]. Mining frequent sequence patterns is actually mining rules related among events [4]-[6]. A Website designer can change the linked order of Web-pages to improve performance of servers, and at the same time, provide personalized services to users by discovering different users' frequent sequence patterns [6]-[8].

Web log frequent sequence pattern mining can use the tranditional Apriori algorithm that needs to generate many candidate datasets, the WAP [8] algorithm that needs to generate recursively a lot of conditional trees(WAP-tree) [9], [10], or the PLWAP [11] algorithm that can judge the relationship among two nodes on WAP-tree by using a position code and is much more efficient than WAP. Different from PLWAP, NGCWAP [12] algorithm can realize this by visiting WAP-tree two times instead.

This paper will put forward a novel Web log frequent sequence pattern mining algorithm, called PREWAP, which has a better execution time and memory consumption than the above algorithms.

2. Related Concepts and Definitions

Definition 1 (length of dataset): Web access sequence database (WASD) is a dataset of sequences $S1$, $S2$, …, Sn. where every $Si(1 \le i \le n)$ is a Web access sequence (or called Web sequential pattern). $|WASD| = n$ is called the length of WASD.

For example: Table 1 (the second column) gives a Web access sequence database WASD0 of 4-length, which contains 4 access sequences : (abdac), (eaebcac), (babfaec), (afbacfc).

Definition 2 (length of sequence): If the length of a sequence S is ε, then S is called a ε-sequence.

For example, S=(abcde) is 5-sequence.

Definition 3 (support and frequent sequence): The support of a sequence S on WASD is defined by λ = (NS / | WASD |)×100% and noted by support(S)WASD, where NS represents occurrence number of all the Web access sequence S' in database WASD and each S' satisfies that it contains S as its sub-sequence (that is each element of S is also an element of S'). Furthermore, for a given positive integer δ as support threshold, if support(S)WASD ≥ δ, then S is called a frequent sequential pattern that meets support δ.

For example: for sequence (ac), its support on WASD0 is (4/4)×100%=100%. If we define support threshold δ=3, then (ac) is a frequent sequential pattern that meets support 3.

Definition 4 (web sequence prefixes and suffixes): For sequence S=<$e1e2...en$>, if $S1$=<$e1e2...ei$> and $S2$ = <ei+1ei+2... en> for some i satisfing 1≤i≤n ($S1$ is an empty set if i=1, and $S2$ is an empty set if i=n), then $S1$ is called the prefix of $S2$ and $S2$ is called the suffix of $S1$.

For example: for sequence (eaebcac), (eae) is the prefix of (bcac), and (bcac) is the suffix of (eae).

Definition 5 (suffix tree set): A suffix tree set is defined by the sub-tree forest comprised by all the children nodes of a specified node as the root node.

Definition 6 (first node set): If a node in PLWAP-tree does not have any ancestor node that shares the same name of the node in a suffix tree or in a tree, then it is called a first node of the tree. If node e is a first node in a tree, then the set composed with all the nodes that share the same name with e is called a first node set.

Table 1. Sample Web Access Sequence Database (WASD0)

TID	Web access sequences	sub-sequence
100	abdac	abac
200	eaebcac	abcac
300	babfaec	babac
400	afbacfc	abacc

3. Improved PREWAP Algorithm Based on PLWAP Algorithm

3.1. PLWAP Algorithm

PLWAP algorithm [12] is based on WAP algorithm [13]. It can avoid recursiving to construct conditional trees in the process of mining by creating a unique position code for every node in the process of creating its PLWAP-tree. But when the PLWAP-tree is very high, its position code will be very long. In that case, it needs to move its pointers of root set many times to judge the relationship between two nodes. Therefore, the algorithm needs more memory and much time to process its mining.

3.2. Improved PREWAP Algorithm

Here, the PREWAP-Tree in our improved PREWAP algorithm is built by the following algorithm 1. Each node in PREWAP-Tree with header table can be denoted as a quadruples: (vent name, occur, PREID, desPREID), which is denoted as the form of 'vent name : occur , (PREID, desPREID)' in Figure 2. In Property 2, we omit the other two components and denote a node α briefly as α[PREID,decPREID].

Table 2. Structure of the Nodes in PREWAP-Tree

Property of a node	Meaning
Event name	name of the event
occur	occurrences of the event
desLink	pointer of the descendant node
PREID	serial number of Pre-visiting in the tree
desPREID	serial number of Pre-visiting of descendant node in the tree

Property 1 If *e* is the first node of frequent sequence L in suffix tree in PREWAP-tree, and the count of *e* first node set is greater than or equal to the minimum support, then Le is also a frequent sequence.

For example, if (ac) is a frequent access pattern (frequent sequence), and b is a frequent event within the suffix sets of c, then, (acb) is also a frequent access pattern.

Property 2 For two nodes α[PREID, decPREID] and β[PREID, decPREID] in PREWAP-tree： it is obvious that:

1) if α. PREID < β. PREID and α. decPREID ≥ β. decPREID, then α is β's ancestor;
2) if α. PREID > β. PREID and α. decPREID ≤ β. decPREID, then α is β's descendant;
3) if α. PREID < β. PREID and α. decPREID < β. decPREID, then α and β are in different branches, besides α is β's left tree;
4) if α. PREID > β. PREID and α. decPREID > β. decPREID, then α and β are in different branches, besides α is β's right tree.

Algorithm 1 (Construction of PREWAP-Tree)

Input: a database WASD, minimum support λ (0 <λ≤|WASD|)

Output: a PREWAP-tree

Begin

(1) /* produce the root */

Add a 'root' node as the root node of PREWAP-tree T;

(2) /* produce 'vent name' and 'occur' values of node*/

Foreach Web access sequence S in WASD, do /* doing (a) and (b)*/

(a) delete all the events in S which don't meet the support λ, and gain frequent subsequence *S'*(*e*1*e*2...*en*). Set currentNode to the leftmost child of root in *T*;

/* All the frequent subsequences of WASD0 in is shown in Table 1 (the third column) */

(b) For *i*=1 to *n* (the length of *S'*) do /* doing (A) and (B)*/

(A) If currentNode is NULL

Create a new child node(*ei*:1);

Else if currentNode is labeled *ei*

Set NodeExist to true;

Else

Set currentNode to currentNode's sibling until *ei* will be found or the sibling is NULL;

(B) if NodeExist

Increase count of *ei* by 1 and set currentNode; /*namely, for the node, occur := occur +1*/

Else

Create a new child node(*ei*:1) ,set currentNode ;

Update descendant links of *ei's* ancestors;

/* The PREWAP-tree of WASD0 is shown in Figure 1 */

(3) /* produce 'PREID' and 'desPREID' values of node*/

Construct a header table used to store frequent events;

Then pre-visit T from root: 'root left subtree right subtree', and at the same time add all nodes with the same event to a linkage queue and record the serial number when one node is visited;

/* The PREWAP-tree (with header table) of WASD0 is shown in Fig. 2 */

End.

Algorithm 2 (Judying the relationship among two nodes in PREWAP-tree)

Input: two nodes α and β of PREWAP-tree

Output : case 1: α is β' *s* ancestor , return 0;

case 2: α is β' *s* descendant, return 3;

case 3: α is on β's left tree, return 1;

case 4: α is on β's right tree, return 2.

Begin

if (α. PREID <β. PREID and α. decPREID≥β. decPREID)

Return 0;

Else if (α. PREID >β. PREID and α. decPREID≤β. decPREID)

Return 3;

Else if (α. PREID < β. PREID and α. decPREID < β. decPREID)

Return 1;

Else

Return 2;

End.

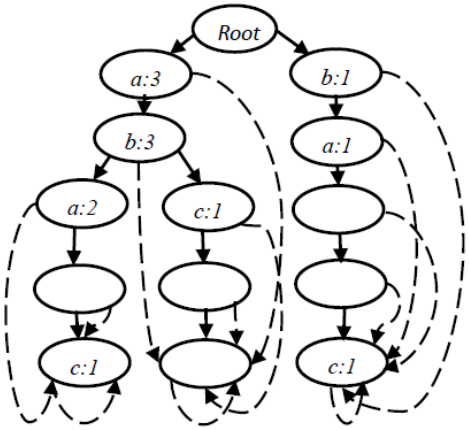

Fig. 1. Construction of PREWAP-tree.

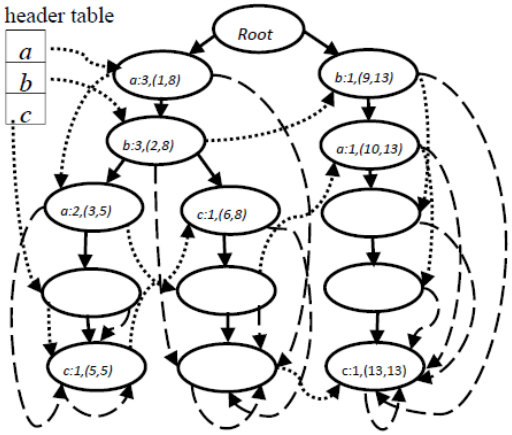

Fig. 2. Construction of PREWAP-tree (with header table).

Algorithm 3 (Mining on PREWAP-tree)

Input: PREWAP-Tree T, the header table H(ei reprents its node), the minimum supporot λ(0 <λ ≤ |WASD|), ε-frequent pattern F(firstly F is empty), the a set of root of suffix tree R (R is equal to the root initially and vj reprents its node)Output: ε+1-frequent sequence F

The main variable: C is used to store the count of the first node set of ei.

Begin

(1)If R is empty

return;

(2) Foreach ei in H, find the suffix tree of vj in T, do

(a)While ei is not empty and vj is not end

Judge the relationship of (vj->lSon, ei) by calling Algorithm 3:

Case 0:If ei is not descendant of e visited in H

$C = C + ei$.occur;

If ei-> lSon is not empty

Push ei to R as R';

Next ei in H;

Case 1: Next vj in R;

Case2,3: Next ei in H;

(b) if C is greater than or equal to λ

Add ei after F as Patter F';

Calling Algorithm 3 recursively but updating R and F with R' and F'

End.

For example, Mine on the PREWAP-Tree in Fig. 2, and set the minimum support λ as 3. Firstly, start from

the first frequent event (a) in the header table, and initialize *R* ={root} and *F*={}. Then search a's First node set in the *R* suffix tree. The process by calling Algorithm 3 recursively can be described as below:

The first call of Algorithm 3: Find the link event queue of event a, and compare the first node (a:3, [1], [8]) in the queue with nodes in *R*. Then it will be found that (a:3, [1,8]) is the descendant of root, and SavePoint is empty, so it is a first node. Then the node a will be inserted into the suffix tree root set *R* as *R'* and (a:3, [1], 8]) is assigned to the SavePoint, and then continue to visit the next node (a:2, [3,5]) in the queue. Although it is the descendant of root, yet is descendant of SavePoint, so it is not a first node, and then skip to the next node (a:1, [7], [8]) in the queue. Since it does not meet the condition too, skip to the next node (a:1, [10,13]) in the queue. This node is not only root's descendant but also in the SavePoint's right tree, and it is also a's first node, so insert it into the suffix tree root set R' as new R' and update SavePoint to (a:1, [10], [13]). Now the next node in the queue is its last node (a:1, [12], [13]), and it is the descendant of root, yet is descendant of SavePoint, so it doesn't meet the condition. In this case, the first node set is {(a:3, [1], [8]) and (a:1,[10], [13])}. Since the a's total count is 4 and 4> 3, so add it after F as F' = {a}.

The scond recursive call of Algorithm 3 will use parameters *R'*={(a:3, [1], [8]), (a:1, [10], [13])} and *F'*={a}, and will find *F'*={a, aa}.

In the end, by calling Algorithm 3 recursively, it can be found that all the frequent patterns areF'={ a, aa , aac, ab, aba, abac, ac, b,ba, bac, bc, c}.

4. Performance Analysis and Experimental Evaluation

For PLWAP mining, at the beginning of mining process, an extra space of memory is needed to store the position code. It may need some space to store the pointers of the root set during mining. NGCWAP mining needs to visit its NGCWAP-tree twice by pre-visiting firstly and post-visiting secondly. While our PREWAP mining needs only one visiting on the tree and needs no more space of memory to save the position code. It can record the descendant relationship of every node when constructing its PREWAP-tree and signs serial number when constructing its header table. Therefore, it can save much time and space. The comparative experimental results among PREWAP, NGCWAP, PLWAP and WAP algorithms will be given in this section.

The four algorithms are implemented with C++ language running under Inprise C++ Builder environment. All experiments are performed on 2.5 GHz AMD Sempron X2 190 PC machine with 6 GigaBytes memory. The operating system is Windows 7 Professional. Datasets are generated using the IBM Quest data mining project. These datasets consist of sequences of events, where each event represents an accessed web page. Comparative experiment 1: Execution time and memory consumption for different support thresholds.

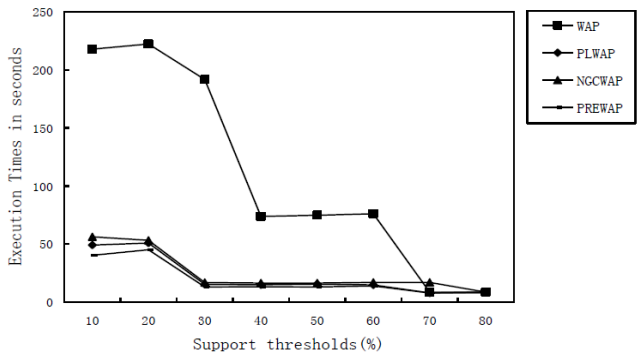

Fig. 3. Execution time for different support under 0.1 million rows dataset in seconds

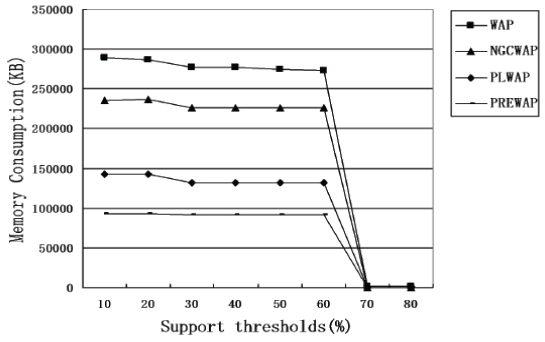

Fig. 4. Memory Consumption for different support under 0.1 million rows dataset in kilobytes(KB).

This experiment uses fixed size database of 0.1 million rows and different minimum support to compare the performance of PREWAP, NGCWAP, PLWAP and WAP algorithms. These algorithms are tested with

minimum supports between 10% and 80% under the test dataset with 0.1 million rows generated by the IBM generator.

Fig. 3 and Fig. 4, it can be seen that the execution time and memory consumption of every algorithm decreases as the minimum support threshold increases. Among these four algorithms, the WAP algorithm is the most time-consuming and memory-consuming algorithm. It is hard to generate the conditional trees .The PLWAP algorithm always uses less runtime and memory than the WAP algorithm. In terms of memory-consumption, PREWAP algorithm gains significantly improvement than PLWAP,NGCWAP and WAP algorithms, and in terms of time-consumption, it improves a little more than PLWAP, NGCWAP and WAP algorithms.

Comparative experiment 2: Execution time and memory consumption for databases with different sizes.

This experiment uses fixed support threshold of 10% and different size datasets to compare the performance of PREWAP, NGCWAP, PLWAP and WAP algorithms. These algorithms are tested on the test datasets with sizes between 0.1 and 0.8 million rows generated by the IBM generator under 10% support threshold.

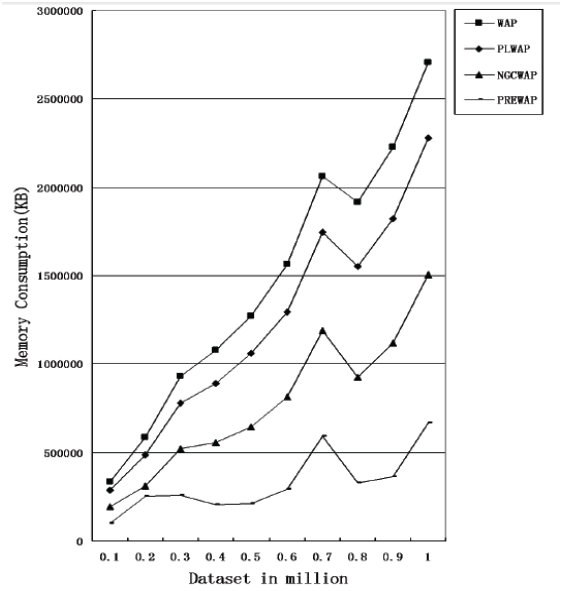

Fig. 5. Memory Consumption for different sizes datasets(KB) under 0.1 support threshold

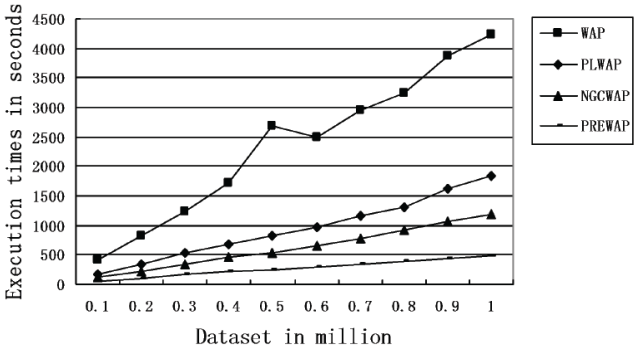

Fig. 6. Execution time for different sizes datasets in seconds under 0.1 support threshold.

From Fig. 5 and Fig. 6, it can be seen that the execution time and memory consumption of every algorithm increases as the size of dataset increases. In terms of memory-consumption and time-consumption, PREWAP algorithm gains much more improvement than PLWAP, NGCWAP and WAP algorithms.

5. Conclusion

The deficiency of PLWAP algorithm is that the length of position code will become very long and need more storage space when the size of its PLWAP-tree is too large. PREWAP algorithm proposed in this paper replaces the position code in PLWAP by signing the descendant relationship and recording the serial number of pre-visiting. Our experiments show that PREWAP algorithm has better performance than NGCWAP, PLWAP and WAP algorithms in terms of runtime and memory consumption, especially in terms of memory consumption.

Reference

[1] Dekun, T. (2011). Association rules mining method of web log based on improved genetic algorithm. *Energy Procedia*, *13*.

[2] Ezeife, C. I., & Liu, Y. (2009). Fast incremental mining of web sequential patterns with PLWAP tree. *Data Min Knowl Disc*, 376-416.

[3] Wang, H., & Zhang, J. W. (2012). Research on the web sequence pattern based on the improved wap algorithm. *Computer Science*, *2*, 206-210 .

[4] Michal, M., Jozef, K., & Peter, Š. (2010). Data preprocessing evaluation for web log mining: reconstruction of activities of a web visitor. *Procedia Computer Science*, *1(1)*, 2273-2280.

[5] Verma, V. (2011). Comprehensive analysis of web log files for mining. *International Journal of Computer Science Issues*, *8(6)*, 199-202.

[6] Pei, J., Han, J., Mortazavi, A. B., & Pinto, H. (2001). PrefixSpan: Mining sequential patterns efficiently by prefix projected pattern growth. *Proceedings of the 2001 International Conference on Data Engineering* (pp. 215–224).

[7] Barouni, E. M. (2008). An interactive search assistant architecture based on intrinsic query stream characteristics. *Computational Intelligence*, *24(2)*, 158-190.

[8] Pan, D. (2002). Discovery of frequency access patterns from web data. *Hsi-An Chiao Tung Ta Hsueh/Journal of Xi'an Jiaotong University*, *36(6)*, 631-634.

[9] Pei, J., & Han, J. W. (2000). Mining access patterns efficiently from web logs. *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining* (pp. 396-407).

[10] Li, C. R. J. (2007). Mining frequent ordered patterns without candidate generation. *Proceedings - Fourth International Conference on Fuzzy Systems and Knowledge Discovery* (pp .402-406).

[11] Spiliopoulou, M. The laborious way from data mining to web mining. *Journal of Computer Systems Science and Engineering*, *14*, 113-126.

[12] Fast incremental mining of web sequential patterns with PLWAP tree. D*ata Mining and Knowledge Discovery*, *19(3)*, 376-416.

[13] Ezeife, C. I., & Lu Y. (2005). Mining web log sequential patterns with position coded pre-order linked WAP-tree. *Data Mining and Knowledge Discovery*, *10(1)*, 5-38.

Meiyu Zheng was born in Dunhua of Jilin province, China, in 1990. She is a master at College of Computer Science, Chongqing University, Chongqing, China. Her research direction is data mining .

Zhengyu Zhu is from Maanshan of Anhui province. He is a Ph.D., a professor and a doctoral supervisor, a senior member of CCF(E200009348S). He research direction is web intelligent search, e-commerce applications, data mining technology.

Yihan Wu was born in Chengdu of Sichuang province, China, in 1991. She is a master at College of Educational Technology, Beijing Normal University, Beijing, China. Her research direction is intelligence science and knowledge systems .