Extracting Sequential Access Pattern from Pre-processed Web Logs.

S.Vijayalakshmi
Assistant Professor, Department of Computer Applications, TCE
Dr.V.Mohan
Professor and Head ,Department of mathematics, TCE
M.S.Sassirekha,O.R.Deepika
PG Students, Department of Computer Applications, TCE

***Abstract*-Finding Frequent Sequential Pattern (FSP) is an important problem in web usage mining. In this paper, we systematically explore a pattern-growth approach for efficient mining of sequential patterns in large sequence database. The approaches adopts a (divide and conquer) pattern-growth principle as follows: Sequence databases are recursively projected into a set of smaller projected databases based on the current sequential pattern(s), and sequential patterns are grown in each projected databases by exploring only locally frequent fragments. Our proposed method combines tree projection and prefix growth features from pattern-growth category with position coded feature from early-pruning category, all of these features are key characteristics of their respective categories, so we consider our proposed method as a pattern growth / early-pruning hybrid algorithm that considerably reduces execution time. These approaches were implemented in hybrid concrete method using algorithms of sequential pattern mining.**

I.INTRODUCTION

One of the data mining methods is sequential pattern discovery introduced in [**Agrawal, 95**.]. Informally, sequential patterns are the most frequently occurring subsequence's in sequences of sets of items. Among many proposed sequential pattern-mining algorithms, most of them are designed to discover all sequential patterns exceeding a user specified minimum support threshold.

Sequential pattern mining is an important data mining problem with broad applications, including the analyses of customer purchase behavior, web access patterns, scientific experiments, disease treatments, natural disaster, and protein formations. A sequential pattern mining algorithm mines the sequence database looking for repeating patterns (known as frequent sequences) that can be used later by end users or management to find associations between the different items or events in their data for purposes such as marketing campaigns, business reorganization, prediction and planning.In this paper, we expand the horizon of frequent sequence pattern mining by analysing an efficient algorithm for mining frequent sequence patterns with web usage mining based on flexible support constrains.

1. Web Usage Mining

It is the task of discovering the activities of the users while they are browsing and navigating through the Web. The aim of understanding the navigation preferences of the visitors is to enhance the quality of electronic commerce services (e-commerce), to personalize the Web portals or to improve the Web structure and Web server performance. In this case, the mined data are the log files which can be seen as the secondary data on the web where the documents accessible through the Web are understood as primary data.

2. Web Usage Mining As A Sequential Pattern Mining Application

Web usage mining (also called *web log mining*) is a special case of sequential pattern mining concerned with finding user navigational patterns by extracting knowledge from web logs, where ordered sequences of events in the sequence database are composed of single items and not sets of items, with the assumption that a web user can physically access only one web page at any given point in time. Currently, most web usage mining solutions consider web access by a user as one page at a time, giving rise to special sequence database with only one item in each sequence's ordered event list. Thus, given a set of events $E = \{a, b, c, d, e, f\}$,which may represent product web pages accessed by users in an E-Commerce application, a web access sequence database for four users may have the four records:

Table – 1 Web Access Sequence

USER ID	Web Access Sequence
T1	<*abdac*>
T2	<*eaebcac*>
T3	<*babfaec*>
T4	<*abfac*>

Mining on this web sequence database can find a frequent sequence *abac* indicating that over 90% of users who visit product a's web page of

978-1-61284-764-1/11/$26.00 ©2011 IEEE

http://www.company.com/producta.htm also immediately visit product b's web page of http://www.compay.com/productb.htm and then revisit product a's page, before visiting product c's page. The web log could be on the server-side, client-side or on a proxy server, each having its own benefits and drawbacks on finding the users' relevant patterns and navigational sessions **[Iváncsy, 06**]. While web log data recorded on the server side reflect the access of a web site by multiple users and is good for mining multiple users' behavior and for web recommender systems, server logs may not be entirely reliable due to caching as cached page views are not recorded in a server log. Web user navigational patterns can be mined using sequential pattern mining of the preprocessed web log. An example of a line of data in a web log is:

137.207.76.120 - [30/Aug/2007:12:03:24 -0500] "GET /jdk1.3/docs/relnotes /deprecatedlist.html HTTP/1.0" 200 2781.

II. RESEARCH PROBLEM

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 Web Usage Mining techniques provide knowledge about the users' behavior on a Web site, knowledge expressed through the relations (patterns) hidden in the log files. Among the techniques available **[Pei 00; Ezeife,03]** the sequential pattern mining is particularly well adapted to the log study due to the sequential nature of the Web site users activity.

A. Problem Statement

The problem of web user access pattern mining is: given web access sequence database WASD and a support threshold ξ, mine the complete set of ξ-patterns of WASD.

Example: Let {s, t, u, v, w, x} be a set of events and 100, 200, 300, and 400 are identifiers of users. A fragment of web log records the information as follows.

(100, s) (100, t) (200, s) (300, t) (200, t) (400, s) (100, s) (400, t) (300, s) (100, u) (200, u) (400,s) (200,s) (300,t) (400,u) (400,u) (300,s)

A pre-processing which divides the log files into access sequences of individual users is applied to the log file, while the resulting access sequence database, denoted as WAS, is shown in the first two columns in Table2. There are totally 4 access sequences in the database. They are not with same length. The first access sequence, *stvsu,* is a 5-sequence, while *st* is a subsequence of it. In access sequence of user 200, both w and *wswtu* prefix with respect to *su. xu* is a 50% pattern because it gets supports from access sequence of user 300 and 400. Please note that even *xu* appears twise in the access sequence of user 400, *sxtsuxu,* but the sequence contributes only one to the count of *xu.*

Table-2 A web access sequence Database.

User ID	Web access Sequence	Frequent subsequence
100	*stvsu*	*stsu*
200	*wswtusu*	*stus*
300	*tstxswu*	*tsts*
400	*sxtsuxu*	*stsuu*

III. RELATED WORK

A. Sequential Pattern Mining:

In this paper, we systematically explore a pattern-growth approach for efficient mining of sequential patterns in large sequence database. Based on this philosophy, we first examine a straightforward pattern growth method, FreeSpan (for Frequent pattern-projected Sequential pattern mining), which reduces the efforts of candidate subsequence generation. we examine another and more efficient method, called PrefixSpan (for Prefix-projected Sequential pattern mining), which offers ordered growth and reduced projected databases.The PrefixSpan consumes a much smaller memory space in comparison with GSP. We examine whether one can fix the order of item projection in the generation of a projected database. Intuitively, if one follows the order of the prefix of a sequence and projects only the suffix of a sequence, one can examine in an orderly manner all the possible subsequence's and their associated projected database.

B.WAP-MINE

[Pei, 00] Pattern-Growth miner with Tree Projection At the same time of FreeSpan and PrefixSpan in 2000/2001, another major contribution was made as a pattern growth and tree structure mining technique, which is the WAP-mine algorithm **[Pei, 00]** with its WAP-tree structure. Here, the sequence database is scanned only twice to build the WAP-tree from frequent sequences along with their support, a "header table" is maintained to point at the first occurrence for each item in a frequent itemset, which is later tracked in a threaded way to mine

the tree for frequent sequences, building on the suffix.

Table-3. Frequent subsequence of web access Sequence DB

User ID	Web access Sequence	Frequent subsequence
T1	<*bcbae*>	<*bbae*>
T2	<*bacbae*>	<*babae*>
T3	<*abe*>	<*abe*>
T4	<*abebd*>	<*abeb*>
T5	<*abad*>	<*aba*>

The first scan of the database finds frequent 1-sequences and the second scan builds the WAP-tree with only frequent subsequences. As an example of WAP-tree, the database of Problem 1 (given in Tables-2) is mined with the min_sup = 3 transactions. The frequent subsequences (third column of Table 3) of each original database sequence are created by removing non-frequent 1-sequences. The tree starts with an empty root node and builds downward by inserting each frequent subsequence as a branch from root to leaf. During the construction of the tree, a header link table is also constructed, which contains frequent 1-items (in our example a, b and e) each one with a link connecting to its first occurrence in the tree and threading its way through the branches to each subsequent occurrence of its node type as shown in Figure 3.2.1. To mine the tree, WAP-mine algorithm starts with the least frequent item in the header link table and uses it as a conditional suffix to construct an intermediate conditional WAP-tree and finds frequent items building on the suffix to get frequent k-sequences. In this example (as in Figure 3.2.1), we start with item e, which is added to the set of frequent sequences as fs={e} and follow its header links to find sequences bba:1, baba:1, ab:2, having supports of b(4) and a(4) to have a and b as frequent 1-sequences. Now, build a WAP-subtree for suffix |e as in Figure 3.2.2 for sequences bba:1, baba:1, ab:2 and mine it to obtain conditional suffix |be as shown in Figure 5-3 following the b-header link, causing sequence be to get added to the set of frequent sequences mined as fs = {e, be}. Following the header links of b in WAP-tree|e (conditional search on e) gives b:1, ba:1, b:-1, a:2, with supports of b(1) and a(3), b's support is less than min_sup so we remove it, resulting in a:1, a:2, giving the conditional WAP-tree|be as in Figure 3.2.3. Next, we add the newly discovered frequent sequence to fs building on the suffix as fs={e, be, abe} and follow its header link in WAP-tree|be to get ø (Figure 3.2.3). Now, the algorithm

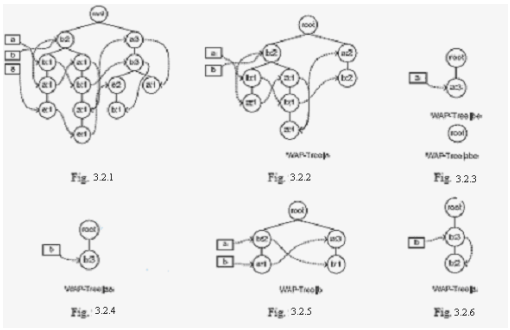

Fig 3.2 WAP-Tree for Web Log from table 3.

recursively backtracks to WAP-tree|*e* in Figure 3.2.1 to mine the *a* link for WAP-tree|*ae* (Figure 3.2.4). Complete mining of our example WAP-tree can be traced in the rest of Figure 3.2.1 with complete set of frequent sequence *fs*={*e, be, abe, ae, b, bb, ab, a, ba*}. WAP-mine algorithm is reported to have better scalability than GSP and to outperform it by a margin **[Pei, 00].** Although it scans the database only twice and can avoid the problem of generating explosive candidates as in Apriori-based and candidate generate-and-test methods, WAP-mine suffers from a memory consumption problem as it recursively reconstructs numerous intermediate WAP-trees during mining and in particular, as the number of mined frequent patterns increases.

IV. PROPOSED METHOD(HYBRID ALGORITHM)

A. Concept of the Algorithm

Our goal is to find a data structure that supports efficient FSP (Frequent Sequence Pattern) mining in terms of both memory and time. Below we propose a special data structure, for this purpose. Table 2 shows some example Web Access Sequences. To detect FSPs, our approach is based on WAP-tree, but avoids recursively re-constructing intermediate WAP-trees during mining of the original WAP tree for frequent Sequence patterns. The Hybrid algorithm is able to quickly determine the suffix of any frequent pattern prefix under consideration by comparing the assigned binary position codes of nodes of the tree. A tree is a data structure accessed starting at its root node and each node of a tree is either a leaf or an interior node. A leaf is an item with no child. An interior node has one or more child nodes and is called the parent of its child nodes. All children of the same node are siblings. Like WAP-tree mining, every frequent sequence in the database

can be represented on a branch of a tree. Thus, from the root to any node in the tree defines a frequent sequence. For any node labeled e in the WAP-tree, all nodes in the path from root of the tree to this node (itself excluded) form a prefix sequence of e. The count of this node e is called the count of the prefix sequence. Any node in the prefix sequence of e is an ancestor of e. On the other hand, the nodes from e (itself excluded) to leaves form the suffix sequences of e.

Given a WAP-tree with some nodes, the binary 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. 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).

B. CONSTRUCTION OF TREE

The tree data structure, similar to WAP-tree, is used to store access sequences in the database, and the corresponding counts of frequent events compactly, so that the tedious support counting is avoided during mining. A Binary code is assigned to each node in our tree. These codes are used during mining for identifying the position of the nodes in the tree. The header table is constructed by linking the nodes in sequential events fashion. Here the linking is used to keep track of nodes with the same label for traversing prefix sequences. This mining algorithm is prefix sequence search rather than suffix search.

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 3.2.1, it can be transformed into its binary tree equivalent shown in figure 4.2, where every node has at most two links, one is its left child, and the other is its sibling.

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.

C. HYBRID ALGORITHM

The 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 greater than or equal to the minimum support are frequent.

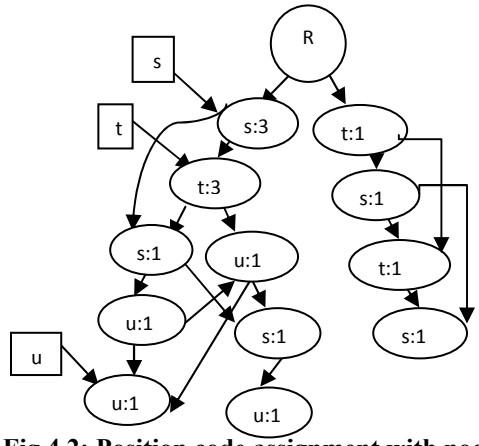

Fig 4.2: Position code assignment with node position in its Complete Tree (binary tree)

Each node in a tree registers three pieces of information: node label, node count and node code, denoted as label: count: position.The root of the tree is a special virtual node with an empty label and count 0. Every other node is labeled by an event in the event set *E*. Then it scans the database a second time to obtain the frequent sequences in each transaction. The non-frequent

events in each sequence are deleted from the sequence. This algorithm also builds a prefix tree data structure by inserting the frequent sequence of each transaction in the tree the same way the WAP-tree algorithm would insert them. Once the frequent sequence of the last database transaction is inserted in the tree, the tree is traversed 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. Then, the algorithm recursively mines the 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, which confirms an *en* node in the root set of e_i, frequent only if the count of all current suffix trees of e_n 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. To mine the tree, the algorithm starts with an empty list of already discovered frequent patterns and the list of frequent events in the head linkage table. Then, for each event, e_i, in the head table, it follows its linkage to first mine 1- sequences, which are recursively extended until the *m*-sequences are discovered. The algorithm finds the next tree node, e_n; to be appended to the last discovered sequence, by counting the support of e_n in the current suffix tree of e_i (header linkage event). Note that e_i and e_n could be the same events. The mining process would start with an e_i event and given the tree, it first mines the first event in the frequent pattern by obtaining the sum of the counts of the first e_n nodes in the suffix subtrees of the Root. This event is confirmed frequent if this count is greater than or equal to minimum support. To find frequent 2-sequences that start with this event, the next suffix trees of e_i are mined in turn to possibly obtain frequent 2-sequences respectively if support thresholds 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. Once the mining of the suffix subtrees near the leaves of the tree are completed, it recursively backtracks to the suffix trees towards the root of the tree until the mining of all suffix trees of all patterns starting with all elements in the header link table are completed.

V. ALGORITHM

Algorithm 1 (Tree Construction for Web access sequences)

Input: Access sequence database D (i), min support MS ($0< MS \leq 1$)
Output: frequent sequential patterns in D (i).
Variables: C_n 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 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 a new childnode(S_1:1),make current_node point to the new node,and insert it into the S_i queue
3. Return (T);

Algorithm 2 (Hybrid Algorithm - Mining the Binary coded WAP Tree)

Input: AWAP tree T, header linkage table L,
Minimum support ξ (0< ξ<1), Frequent m-sequence F).
Suffix tree roots set R(R includes root and F is empty first time algorithm is called.
Output: Frequent (m+1) sequence, F^1.
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
If R is empty, return
For each even e_i in L, find the suffix tree of e_i in T, do
Save first event in e_i -queue to S.
Following the e_i queue
If event e_i is the descendant of any event in R, and is not descendant of S,
Insert it into suffix-tree-header set R^1
Add count of e_i to C.
Replace the S with e_i
If C is greated then ξ
Append e_i after F to F^1 and output F^1
Call algorithm AWAPT-Mine and passing R^1 and F^1.
End // of AWAPT-Mine //

VI. EXPERIMENTAL EVALUATION AND PERFORMANCE STUDY

In this section, we report our experimental results on the performance of proposed method in comparison with WAP Tree. It shows that our proposed method is outperforms other previously proposed methods and is efficient and scalable for mining sequential patterns in large databases.

All the experiments are performed on a 2.20 GHz core2duo laptop with 3 GB memory, running Microsoft Windows/NT. The synthetic datasets we used for our experiments were generated using standard procedure. The proposed 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. In summary our performance study shows that our proposed method is more efficient and scalable than WAP Tree, Whereas WAP tree is faster than FS -tree when the support threshold is low, and there are many long patterns. Our proposed Hybrid 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.

Table-4: Sequence Database density vs. algorithm execution time (in sec), at minimum support of 1%.

	SPAM	Prefix Span	FS-mine	WAP-mine	Proposed method
More Dense C15T10SBN20D200K	200	252	220	200	170
Dense C12T8S6N60D200K	40	43	58	55	32
Less Sparse C10T6SNB0D200K	10	15	9	5	2
Sparse C8T5S4N100D200K	8	14	12	3	2

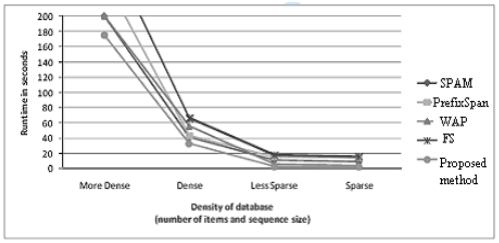

Fig-6. Sequence Database density vs. algorithm execution time (in sec), at minimum support of 1%.

VII. CONCLUSIONS

In this paper, we have developed a novel, scalable, and efficient frequent sequential pattern mining method. Our systematic performance study shows that our proposed method mines the complete set of patterns and is efficient and runs considerably faster than both based WAP Tree. 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. The Proposed Hybrid algorithm is able to quickly determine the suffix of any frequent pattern prefix under consideration by comparing the assigned binary position codes of nodes of the tree.

REFERENCES

[1] [Agrawal, 95] Agrawal, R. and Srikant, R. Mining sequential patterns. In Proc. 1995 Int. Conf. Data(ICDE'95), p.3–14, March 1995.

[2] [Agrawal, 99] 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] [Srivastava, 00] Srivastava, J., Cooley, R., Deshpande, M. and Tan, P. Web usage mining: Discovery and applications of usage patterns from web data. SIGKDD Explorations, 2000.

[4] [Han, 01] 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.

[5] [Srivastava, 00] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan, "Web usage mining: Discovery and applications of usage patterns from web data," *SIGKDD Explorations*, Vol. 1, No. 2, pp. 12-23,2000

[6] [Cooley, 97] R. Cooley, B. Mobasher, and J. Srivastava. Web Mining: Information and Pattern Discovery on the World Wide Web. In *8th IEEE Intl. Conf. on Tools with AI*, 1997.

[7] [Vijayalakshmi, 08] S.Vijayalakshmi, Dr.V.Mohan and S.Suresh Raja." Optimization Of Constraint-Based Multidimensional Frequent Sequential Pattern in Web Usage Mining Using Association Rule Mining Techniques" in International conference of Data management [ICDM 2008], New Delhi.

[8] [Wang, 04] WANG, J., AND HAN, J. 2004. BIDE: Efficient Mining of Frequent Closed Sequences. In *Proceedings of the 20th International Conference on Data Engineering,* , Boston, MA, April 2004, 79-90.

[9] **S.Vijayalakshmi** –Currently she is a Ph.D., research scholar of Anna University, Chennai. She is currently working as a Asst.Professor in MCA Department. Her papers were published in various International journals and Conferences. Her research interests include Data mining, Artificial Intelligence, and Web Development.

[10] **Dr.V.Mohan** - received Doctoral degree in Mathematics from Madurai Kamaraj University, Tamil Nadu, India. He is currently working as a Professor and Head in Mathematics Department.