MINING OF USERS' ACCESS BEHAVIOUR FOR FREQUENT SEQUENTIAL PATTERN FROM WEB LOGS

S.Vijayalakshmi
Senior Lecturer, Department of Computer Applications
Thiagarajar College of Engineering, Madurai, Tamil Nadu, India
E-mail: svlcse@tce.edu

V.Mohan
Professor and Head Department of mathematics
Thiagarajar College of Engineering, Madurai, Tamil Nadu, India
E-mail: vmohan@tce.edu; srubi77@yahoo.com

S.Suresh Raja
Senior Lecturer, Department of Computer Applications
K.L.N College of Engineering, Madurai , Tamil Nadu, India
E-mail: suresh.csr@gmail.com

ABSTRACT:

Sequential Pattern 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. The task of discovering frequent sequences is challenging, because the algorithm needs to process a combinatorially explosive number of possible sequences. Discovering hidden information from Web log data is called Web usage mining. One common usage in web applications is the mining of users' access behaviour for the purpose of predicting and hence pre-fetching the web pages that the user is likely to visit. The aim of discovering frequent Sequential patterns in Web log data is to obtain information about the access behaviour of the users.

Finding Frequent Sequential Pattern (FSP) is an important problem in web usage mining. In this paper, we explore a new frequent sequence pattern technique called AWAPT (**A**daptive **W**eb **A**ccess **P**attern **T**ree), for FSP mining. An AWAPT combines Suffix tree and Prefix tree for efficient storage of all the sequences that contain a given item. It eliminates recursive reconstruction of intermediate WAP tree during the mining by assigning the binary codes to each node in the WAP Tree. Web access pattern tree (WAP-tree) mining is a sequential pattern mining technique for web log access sequences, which first stores the original web access sequence database(WASD) on a prefix tree, similar to the frequent pattern tree (FP-tree) for storing non-sequential data. WAP-tree algorithm then, mines the frequent sequences from the WAP-tree by recursively re-constructing intermediate trees, starting with suffix sequences and ending with prefix sequences. An attempt has been made to AWAPT approach for improving efficiency. AWAPT totally eliminates the need to engage in numerous reconstructions of intermediate WAP-trees during mining and considerably reduces execution time.

Keywords: Data Mining, Sequential pattern mining, frequent pattern mining, web usage mining, AWAPT.

1. INTRODUCTION

One of the data mining methods is sequential pattern discovery introduced in [2]. 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. In this paper, we explore a new frequent sequence pattern technique called AWAPT (**A**daptive **W**eb **A**ccess **P**attern **T**ree), for FSP mining..

1.1 Web Mining Approaches

World Wide Web Data Mining includes content mining, hyperlink structure mining, and usage mining. All three approaches attempt to extract knowledge from the Web, produce some useful results from the knowledge extracted, and apply the results to certain real-world problems. The first two apply the data mining techniques to Web page contents and hyperlink structures, respectively. The third approach, Web usage mining, the theme of this article, is the application of data mining techniques to the usage logs of large Web data repositories in order to produce results that can be applied to many practical subjects, such as improving Web sites/pages, making additional topic or product recommendations, user/customer behavior studies, etc. A Web usage mining system must be able to perform five major functions: i) data gathering, ii) data preparation, iii) navigation pattern discovery, iv) pattern analysis and visualization, and v) pattern applications.

Web mining can be categorized into three different classes based on which part of the Web is to be mined. These three categories are (i) Web content mining, (ii) Web structure mining and (iii) Web usage mining.

1.2 Web content mining is the task of discovering useful information available on-line. There are different kinds of Web content which can provide useful information to users, for example multimedia data, structured (i.e. XML documents), semi-structured (i.e. HTML documents) and unstructured data (i.e. plain text). The aim of Web content mining is to provide an efficient mechanism to help the users to find the information they seek. Web content mining includes the task of organizing and clustering the documents and providing search engines for accessing the different documents by keywords, categories, contents etc.

1.3 Web structure mining is the process of discovering the structure of hyperlinks within the Web. Practically, while Web content mining focuses on the inner-document information, Web structure mining discovers the link structures at the inter-document level. The aim is to identify the authoritative and the hub pages for a given subject. Authoritative pages contain useful information, and are supported by several links pointing to it, which means that these pages are highly referenced. A page having a lot of referencing hyperlinks means that the content of the page is useful, preferable and maybe reliable. Hubs are Web pages containing many links to authoritative pages, thus they help in clustering the authorities. Web structure mining can be achieved only in a single portal or also on the whole Web. Mining the structure of the Web supports the task of Web content mining. Using the information about the structure of the Web, the document retrieval can be made more efficiently, and the reliability and relevance of the found documents can be greater. The graph structure of the web can be exploited by Web

structure mining in order to improve the performance of the information retrieval and to improve classification of the documents.

1.4 Web usage mining

There are three types of log files that can be used for Web usage mining. Log files are stored on the server side, on the client side and on the proxy servers. By having more than one place for storing the information of navigation patterns of the users makes the mining process more difficult. Really reliable results could be obtained only if one has data from all three types of log file. The reason for this is that the server side does not contain records of those Web page accesses that are cached on the proxy servers or on the client side. Besides the log file on the server, that on the proxy server provides additional information. However, the page requests stored in the client side are missing. Yet, it is problematic to collect all the information from the client side. Thus, most of the algorithms work based only the server side data. Some commonly used data mining algorithms for Web usage mining are association rule mining, sequence mining and clustering.

2. Web Access pattern Mining

Web Access pattern mining is also called as Web usage mining. Web usage mining, from the data mining aspect, is the task of applying data mining techniques to discover usage patterns from Web data in order to understand and better serve the needs of users navigating on the Web [13]. As every data mining task, the process of Web usage mining also consists of three main steps: (i) pre-processing, (ii) pattern discovery and (iii) pattern analysis. The pre-processing step contains three separate phases. Firstly, the collected data must be cleaned, which means that graphic and multimedia entries are removed. Secondly, the different sessions belonging to different users should be identified. Thirdly, a session is understood as a group of activities performed by a user when he is navigating through a given site. To identify the sessions from the raw data is a complex step, because the server logs do not always contain all the information needed. In this work, pattern discovery means applying the introduced frequent Sequential pattern discovery methods to the log data. For this reason the data have to be converted in the pre-processing phase such that the output of the conversion can be used as the input of the algorithms. Pattern analysis means understanding the results obtained by the algorithms and drawing conclusions. The motivation behind pattern analysis is to filter out uninteresting rules or patterns from the set found in the pattern discovery phase. The exact analysis methodology is usually governed by the application for which Web mining is done. The most common form of pattern analysis consists of a knowledge query mechanism such as SQL. Visualization techniques, such as graphing patterns or assigning colours to different values, can often highlight overall patterns or trends in the data. Content and structure information can be used to filter out patterns containing pages of a certain usage type, content type, or pages that match a certain hyperlink structure.

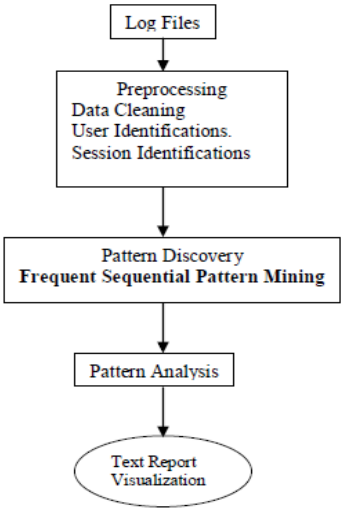

Fig-3: Process of web usage mining

3. 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.

In this paper, we focus on mining web access patterns. In general, a web log can be regarded as a sequence of pairs of user identifier and event. In this investigation, web log files are divided into pieces per mining purpose. Pre-processing can be applied to the original web log files, so that pieces of web log can be obtained. Each piece of web log is a sequence of events from one user or session in timestamp ascending order. We model pieces of web log as sequences of events, and mine the sequential patterns over certain support threshold.

3.1 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 Table 1 .

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.*

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

Table-4.1 A web access sequence Database.

4. Related Work:

4.1 Sequential Pattern Mining:

The sequential pattern mining problem was first introduced by Agrawal and Srikant in [2]: Given a set of sequences, where each sequence consists of a list of elements and each element consists of a set of items, and given a user-specified min_support threshold, sequential pattern mining is to find all frequent subsequences, i.e., the subsequences whose occurrence frequency in the set of sequences is no less than min_support.

Sequential Pattern Mining comes in Association rule mining. For a given transaction database T, an association rule is an expression of the form $X \rightarrow Y$, where X and Y are subsets of A and $X \rightarrow Y$ holds with confidence τ, if τ % of transactions in D that support X also Y. The rule $X \rightarrow Y$ has support σ in the transaction set T if σ % of transactions in T support X U Y. Association rule mining can be divided into two steps. Firstly, frequent patterns with respect to support threshold minimum support are mined. Secondly association rules are generated with respect to confidence threshold minimum confidence.
[3] Proposed a method for discovering access patterns from web logs based on a new type of association patterns. They handle the order between page accesses, and allow gaps in sequences. They use a candidate generation algorithm that requires multiple scans of the database. Their pruning strategy assumes that the site structure is known. [2] presented an algorithm for finding generalized sequential patterns that allows user-specified window-size and user-defined taxonomy over items in the database. This algorithm required multiple scans of the database to generate candidates.

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. 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. To further improve the performance, a pseudo projection technique is

developed in PrefixSpan. A comprehensive performance study shows that PrefixSpan, in most cases, outperforms the Apriori-based algorithm GSP, FreeSpan. PrefixSpan, integrated with pseudo projection, is the fastest among all the tested algorithms. The PrefixSpan consumes a much smaller memory space in comparison with GSP. This pattern-growth methodology can be further extended to mining multilevel, multidimensional sequential patterns, and mining other structured patterns.

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. WE examine WAP tree structure for frequent sequence pattern mining in web log files.

5.1 WAP-tree:

WAP-tree, which stands for web access pattern tree. A nice data structure, WAP-tree, is devised to register access sequences and corresponding counts compactly, so that the tedious support counting can be avoided. It also maintains linkages for traversing prefixes with respect to the same suffix pattern efficiently. A WAP-tree registers all and only all information needed by the rest of mining. Once such a data structure is built, all the remaining mining processing is based on the WAP-tree. The original access sequence database is not needed any more. Because the size of WAP-tree is usually much smaller than that of the original access sequence database, the construction of WAP-tree is quite efficient by simply scanning the access sequence database twice.

An efficient recursive algorithm is proposed to enumerate access patterns from WAP-tree. No candidate generation is required in the mining procedure, and only the patterns with enough support will be under consideration. The philosophy of this mining algorithm is conditional search. Instead of searching patterns level-wise as Apriori, 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. Conditional search is a partition-based divide-and-conquer method instead of bottom-up generation of combinations. It avoids generating large candidate sets.

The main steps involved in this technique are summarized. The WAP-tree stores the web log data in a prefix tree format similar to the frequent pattern tree (FP-tree) for non-sequential data.

- The 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.

Based on the above observations, a Web access pattern tree structure, or WAP-tree in short, can be defined as follows.

1. Each node in a WAP-tree registers two pieces of information: **label** and **count,** denoted as *label: count.* 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, and is associated with a count which registers the number of occurrences of the corresponding prefix ended with that event in the Web access sequence database.

2. The WAP-tree is constructed as follows: for each access sequence in the database, filter out any non frequent events, and then insert the resulting frequent subsequence into WAP-tree. The insertion of frequent subsequence is started from the root of WAP-tree. Considering the first event, denoted as e, increment the count of child node with label e by 1 if there exists one; otherwise create a child labeled by e and set the count to 1. Then, recursively insert the rest of the frequent subsequence to the subtree rooted at that child labeled e.

3. Auxiliary node linkage structures are constructed to assist node traversal in a WAP-tree as follows. All the nodes in the tree with the same label are linked by shared-label linkages into a queue, called event-node queue, The event node queue of with label e_i is also called e_i queue. There is one header table *H* for a WAP-tree, and the head of each event-node queue is registered in *H.*

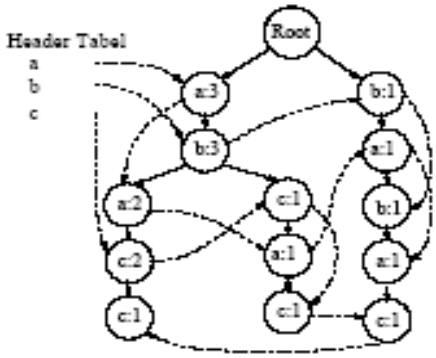

Fig 5.4.1 Complete WAP Tree.

The WAP-tree is shown in Figure 5.4.1, which is constructed as follows. First, insert the sequence abac into the initial tree with only one virtual root. It creates a new node (a: 1) (i.e., labeled as a, with count set to 1) as the child of the root, and then derives the a-branch \ (a: 1)! (B: 1)! (A: 1)! (c: 1)", in which arrows point from parent nodes to children ones. Second, insert the second sequence abcac. It starts at the root. Since the root has a child labeled a, a's count is increased by 1, i.e., (a : 2) now. Similarly, we have (b : 2). The next event, c, does not match the existing node a, and a new child node c : 1

is created and inserted. The remaining sequence insertion process can be derived accordingly.

WAP-tree algorithm scans the original database only twice and avoids the problem of generating explosive candidate sets as in Apriori-like algorithms. Mining efficiency is improved sharply, but the main drawback of WAP-tree mining is that it recursively constructs large numbers of intermediate WAP-trees during mining and this entails storing intermediate patterns, which are still time consuming operations.

5. AWAPT (Adaptive Web Access Pattern Tree)

5.1 Concept of AWAPT

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, **AWAPT**, for this purpose. Table 4.1 shows some example Web Access Sequences. To detect FSPs.

The AWAPT approach is based on WAP-tree, but avoids recursively re-constructing intermediate WAP-trees during mining of the original WAP tree for frequent patterns. The AWAPT 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.

5.2 Rule of AWAPT:

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).

5.3 Construction of AWAPT

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 AWAPT. 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.

In data structure, when implementing a general tree data structure, a tree is usually transformed into its equivalent binary tree, which has a fixed number of child nodes. 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 5.4.1, it can be transformed into its binary tree equivalent shown in figure 6.3, 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.

For example, in figure 6.3, (s: 1:1110) is an ancestor of (u: 1:111011) because the position code of (u: 1:1110) is 1110, and after appending 1 at the end of 1110, we get 11101, which is equal to the first 5 (i.e., length of u + 1) bits of (u: 1:111011). On the other hand, (u: 1:1110) is not the ancestor of (u: 1:101111), since after appending 1, the code will be 11101 and is not equal to the first 5 bits position code of (u: 1:101111). Not only can we use the position code to find the ancestor and descendant relationships between nodes, but we can also find whether one node belongs to the right-tree or left-tree of another node. From figure 6.1, it can be seen that node (u: 1:1111) and node (u: 1:111011) are two nodes that belong to two sub trees, which are rooted at (s: 2:111) and (s: 1:1110) respectively. The node (s: 1:1111) belongs to a left-tree of (s: 1:111011) since the fourth bit of (s: 1:111011) is 0, which means the node is extended from the node with

position code 1110. The node with position code 1110 is a right sibling of node with 111, which is an ancestor of node (s: 1:1111). Thus, (s: 1:111011) is a right-tree of (s: 1:1111).

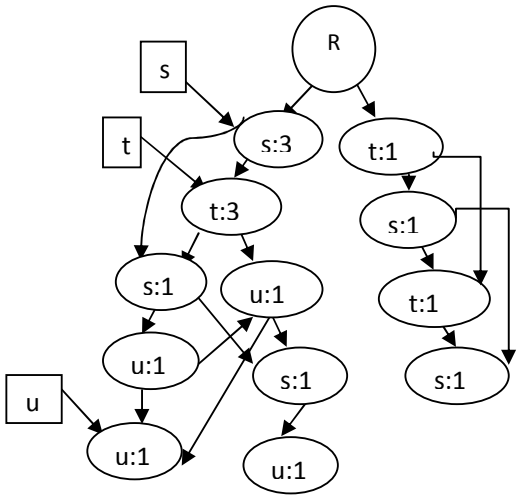

Fig 6.3: Position code assignment with node position in its Complete AWAPT (binary tree)

5.4 **Algorithm of AWAPT**

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. Each node in a AWAPT 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, *en*; to be appended to the last discovered sequence, by counting the support of *en* in the current suffix tree of e_i (header linkage event). Note that e_i and *en* 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.

6. Algorithm

Algorithm 1 (WAP-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 **AWAPT** 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);

7. Experimental Evaluation and performance study.

In this section, we report our experimental results on the performance of AWAPT in comparison with WAP Tree and FS-Tree. It shows that AWAPT 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 described in [2]. The same data generator has been used in most studies on sequential pattern mining, such as [11, 6]. We refer readers to [2] for more details on the generation of data sets.

The execution time of every algorithm decreases as the minimum support increases. This is because when the minimum support increases, the number of candidate sequence decreases. Thus, the algorithms need less time to find the frequent sequences. The AWAPT 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. It is however, more realistic to assume that such techniques are run in regular systems available in many environments, which are not memory only, but could be multiple processor systems sharing memories and CPU's with virtual memory support. As the minimum support threshold decreases, the number of events that meet minimum support increases. This means that WAP-tree becomes larger and longer, and the algorithm needs much more I/O work during mining of WAP tree. As minimum support decreases, the execution time difference between WAP-tree and AWAPT increases.

Algorithms	Time in sec's at different supports				
	2	3	4	5	10
WAP	750	510	330	280	150
AWAPT	230	160	110	95	48

Table 8.1 Execution times for dataset at different Minimum supports.

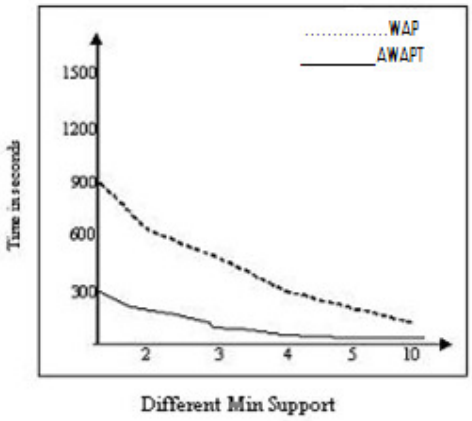

Figure 8.1. Execution times trend with different minimum supports.

In summary our performance study shows that AWAPT is more efficient and scalable than WAP Tree and FS-Tree, Whereas WAP tree is faster than FS -tree when the support threshold is low, and there are many long patterns. The AWAPT 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. This algorithm also eliminates the need to store and scan intermediate conditional pattern bases for reconstructing intermediate WAP trees.

9. Conclusions:

In this paper, we have developed a novel, scalable, and efficient frequent sequential pattern mining method, called AWAPT. Our systematic performance study shows that AWAPT mines the complete set of patterns and is efficient and runs considerably faster than both based WAP Tree and FS-Tree algorithms. This algorithm uses the pre-order linking of header nodes to store all events e_i in the same suffix tree closely together in the linkage, making the search process more efficient. 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 Extended version of Web Access Pattern approach is based on AWAPT, and avoids recursively re-constructing intermediate WAP-trees during mining of the original WAP tree for frequent patterns. The AWAPT 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, 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.

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

[3] A. Nanopoulos and Y. Manolopoulos. Mining patterns from graph traversals. Data and Knowledge Engineering, 37(3):243– 266, 2001.

[4] Etzioni, O. The world wide web: Quagmire or gold mine. Communications of the ACM, p.65 – 68, 1996.

[5] Han, J., Pei, J. et al. FreeSpan: Frequent pattern projected sequential pattern mining.In SIGKDD, p.355–359, Aug. 2000.

[6] Han, J., Pei, J., Yin, Y. and Mao, R. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. International Journal of Data Mining and Knowledge Discovery, p.53– 87, Jan 2004.

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

[8] 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.

[9] Han, J., Pei, J., Mortazavi-Asl, B. and Zhu, H. Mining access patterns efficiently from web logs. In Proceedings of the Pacific- Asia Conference on Knowledge Discovery and Data Mining (PAKDD'00) Kyoto Japan, 2000. Jian Pei, Jiawei Han, Behzad Mortazavi-asl, and Hua Zhu

[10] Han, J., Pei, J., Mortazavi-Asl, B., and Pinto, H. 2001. PrefixSpan: Mining sequential patterns efficiently by prefixprojected pattern growth. In Proceedings of the 2001 International Conference on Data Engineering (ICDE'01). Germany, Heidelberg, p. 215–224.

[11] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu, "Mining access patterns efficiently from web logs," in PADKK '00: Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications. London, UK: Springer-Verlag, 2000, pp.396-407

[12] R. Cooley, B. Mobasher, and J. Srivastava, "Data preparation for mining world wide web browsing patterns," *Knowledge and Information Systems*, Vol. 1, No. 1, pp. 5-32, 1999

[13] 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

[14] M. S. Chen, J. S. Park, and P. S. Yu, "Data mining for path traversal patterns in a web environment," in *Sixteenth International Conference on* Distributed *Computing Systems*, 1996, pp. 385-392

[15] J. Punin, M. Krishnamoorthy, and M. Zaki, "Web usage mining:Languages and algorithms," in *Studies in Classification, Data Analysis, and* Knowledge *Organization*. Springer-Verlag, 2001

[16] 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.

[17] M. Spiliopoulou and L.C. Faulstich. WUM: A Tool for Web Utilization Analysis. In *EDBT Workshop WebDB'98, LNCS 1590*. Springer Verlag, March 1998.

[18] R. Srikant and R. Agrawal. Mining generalized association rules. In *21st VLDB Conf.*, 1995.

[19] 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.

[20] E. Cohen, B. Krishnamurthy, and J. Rexford. Improving end-to-end performance of the web using server volumes and proxy filters. In Proc. ACM SIGCOMM, pages 241{253, 1998.

[21] M. Eirinaki and M. Vazirgiannis, "Web mining for web personalization,"*ACM* Trans. *Inter.Tech.*, Vol. 3, No. 1, pp. 1-27, 2003

[22] S.Vijayalakshmi, Dr.V.Mohan and S.Suresh Raja." Mining Constraint-based multidimensional Frequent Sequential Pattern in Web Logs" in European Journal of Scientific Research , ISSN 1450-216X Vol.36 No.3 (2009), pp 480-490 © EuroJournals Publishing, Inc. 2009.

[23] Zaki, M. SPADE: An efficient algorithm for mining frequent sequences. Machine Learning, p.31–60, 2001.

[24] Ezeife, C. and Lu, Y. Mining web log sequential patterns with position coded preorder linked wap-tree. International Journal of Data Mining and Knowledge Discovery (DMKD) Kluwer Publishers, p.5–38, 2005.

[25] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu, "Mining access patterns efficiently from weblogs," in PADKK '00: Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications. London, UK: Springer-Verlag, 2000, pp.396-407

[26] A. Nanopoulos, D. Katsaros and Y. Manolopoulos, "A data mining algorithm for generalizedweb prefetching," ISEEE Transactions on Knowledge and Data Engineering, Vol. 15, No. 5,pp. 1155-1169, 2003.

[27] C.I. Ezeife. and Mostafa Monwar, "A Plwap-Based Algorithm For Mining Frequent Sequential Stream Patterns, International Journal of Information" Technology and Intelligent Computing (ITIC), Vol.2, No.1, 2007, pp.89-116, endorsed by IEEE Computing Intelligence Society (http://itic.wshe.lodz.pl).