Journal of Computer Science 6 (11): 1293-1300, 2010
ISSN 1549-3636

Mining Sequential Access Pattern with Low Support From Large Pre-Processed Web Logs

[1]S. Vijayalakshmi and [2]V. Mohan
[1]Department of Computer Applications,
[2]Department of Mathematics,
Thiagarajar College of Engineering, Madurai, Tamil Nadu, India

Abstract: Problem statement: To find frequently occurring Sequential patterns from web log file on the basis of minimum support provided. We introduced an efficient strategy for discovering Web usage mining is the application of sequential pattern mining techniques to discover usage patterns from Web data, in order to understand and better serve the needs of Web-based applications. **Approach:** The approaches adopt a divide-and conquer pattern-growth principle. Our proposed method combined 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. **Results:** Our proposed Hybrid algorithm eliminated the need to store numerous intermediate WAP trees during mining. Since only the original tree was 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. **Conclusion:** An attempt had been made to our approach for improving efficiency. Our proposed method totally eliminates reconstructions of intermediate WAP-trees during mining and considerably reduces execution time.

Key words: Data mining, sequential pattern mining, frequent pattern mining, web usage mining, hybrid algorithm, WAP-tree

INTRODUCTION

One of the data mining methods is sequential pattern discovery introduced in (Ren and Zhou, 2006). 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. With increase in the use of the world wide web for E-Commerce businesses, web services and others, web usage mining has appeared in the literature as one of the most prevalent application areas of sequential pattern mining (Han *et al*., 2000; Ezeife and Lu, 2005; El-Sayed *et al*., 2004; Han *et al*., 2004; 2005; Madan and Madan, 2010) This study focuses on sequential pattern mining techniques including those applicable to web usage mining.

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).

Table 1: Web access sequence

USER ID	Web access sequence
T1	<abdac>
T2	<eaebcac>
T3	<babfaec>
T4	<abfac>

Corresponding Author: S. Vijayalakshmi, Department of Computer Applications, Thiagarajar College of Engineering, Madurai, Tamil Nadu, India

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 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. Store managers may then place promotional prices on product a's web page, that is visited a number of times in sequence, to increase the sale of other products. 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 (Ivancsy and Vajk, 2006). 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. Client-side data collection, require that a remote agent be implemented or a modified browser be used to collect single-user data, thus eliminating caching and session identification problems and is useful for web content personalization applications. A proxy server can, on the other hand, reveal the actual HTTP requests from multiple clients to multiple web servers, thus, characterizing the browsing behavior of a group of anonymous users sharing a common server (Ren *et al*., 2006). Web user navigational patterns can be mined using sequential pattern mining of the preprocessed web log. Web usage mining works on data generated by observing web surf sessions or behaviors, which are stored in the web log, from where all users' behaviors on each web server can be extracted. 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

This recorded information is in the format: host/ip user [date: time] "request url" status bytes and represents from left to right, the host IP address of the computer accessing the web page. In most cases, researchers are assuming that user web visit information is completely recorded in the web server log, which is preprocessed to obtain the transaction database to be mined for sequences. Several sequential pattern mining techniques that can be applied to web usage mining have been introduced in the literature since mid 1990's. Previous surveys looked at different mining methods applicable to web logs (Srivastava *et al*., 2000; Ivancsy and Vajk, 2006), but they lack of two important things; they fail to (i) focus on sequential patterns as a complete solution, (ii) include a deep investigation of the techniques and theories used in mining sequential patterns.

MATERIALS AND METHODS

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 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 2. 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 twice in the access sequence of user 400, sxtsuxu, but the sequence contributes only one to the count of xu.

Study of WAP-mine algorithm (Han *et al.* 2000)-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 (Han *et al*., 2000) with its WAP-tree structure.

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

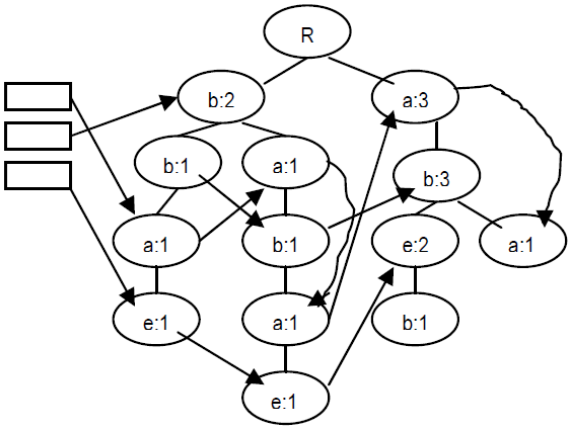

Fig. 1: Complete WAP tree

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>

Here, the sequence database is scanned only twice to build the WAPtree from frequent sequences along with their support, a "header table" is maintained to point at the first occurrence for each item in a frequent item set, which is later tracked in a threaded way to mine the tree for frequent sequences, building on the suffix. The first scan of the database finds frequent 1-sequences and the second scan builds the WAPtree with only frequent subsequences. As an example of WAP-tree, the database of Problem 1 (Table 3) 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 Fig. 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.

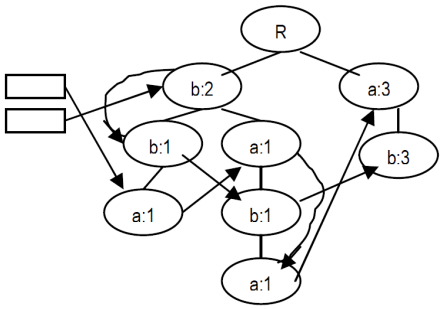

Fig. 2: WAP tree |e

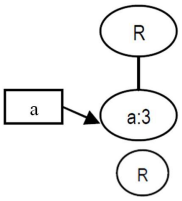

Fig. 3: WAP tree |be

In this example (Fig. 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-sub-tree for suffix |e as in Fig. 2 for sequences bba:1, baba:1, ab:2 and mine it to obtain conditional suffix |be as shown in Fig. 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 Fig. 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 (Fig. 3). Now, the algorithm recursively backtracks to WAP-tree|e in Fig. 1 to mine the link for WAP-tree|ae (Fig. 4). Complete mining of our example WAP-tree can be traced in the rest of Fig. 5 and 6 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 (Han *et al*., 2000). 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.

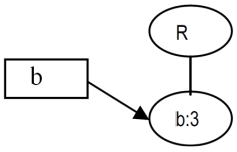

Fig. 4: WAP tree|ae

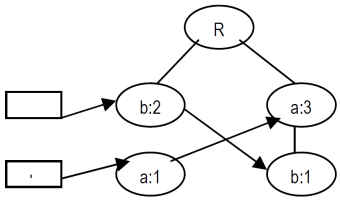

Fig. 5: WAP tree |ab

Table 4: Access sequence database D

User ID	TID	Access sequence
100	T1	<bcbae>
200	T2	<bacbae>
300	T3	<abe>
400	T4	<abebd>
500	T5	<abad>

Study of FS-miner algorithm (El-Sayed *et al*. 2004): Inspired by FP-tree (Han *et al*., 2000; 2004) and ISM (Thakur *et al*., 2006) FS-Miner is a tree projection pattern growth algorithm that resembles WAP-mine and supports incremental and interactive mining. The significance of FS-Miner is that it starts mining immediately with 2-subsequences from the second-which is also the last- scan of the database (at k = 2). It is able to do so due to the compressed representation in the FS-tree, which utilizes a header table of edges (referred to as links in El-Sayed *at al*. (2004)) rather than single nodes and items compared to WAP-tree. It is also considered a variation of trie as it stores support count in nodes as well as edges of the tree that represent 2-sequences and are required for the incremental mining process. Consider Table 1 for a running example on FS-Miner and keep in mind that it only considerscontiguous sequences (e.g., cba is a contiguous subsequence of bcbae, but ca is not). Figure 5 shows the header table generated along with FS-tree for sequences in Table 4 FS-miner and the FS-tree R maintain two kinds of support counts and minimum support, namely $MSuppC^{seq}$, the minimum frequent sequence support, similar to min_sup used throughout this study and $MSuppC^{link}$, the minimum frequent link (a frequent 2-sequence) support. A link h with support count $Supp^{link}$ (h) is considered a frequent link if $Supp^{link}(h) \geq MSuppC^{seq}$ and is considered potentially frequent link if $MSuppC^{link} \leq Supp^{link}(h) < MSuppC^{seq}$. If $Supp^{link}(h)$ does not satisfy both $MSuppC^{link}$ and $MSuppC^{seq}$ then it is a non-frequent link.

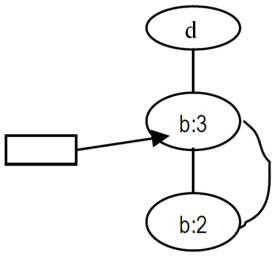

Fig. 6: WAP tree | b

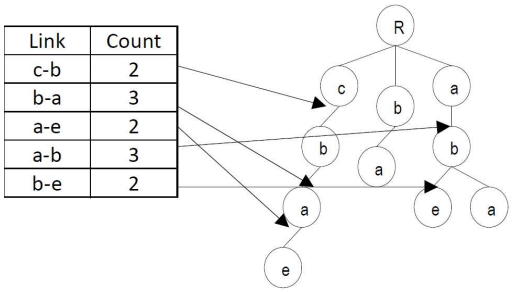

Fig. 7: FS-Tree and header table HT for Web Log from Table 4

Potentially frequent links are maintained to support and enable the incremental and interactive mining capability of FS-miner.

Frequent links and potentially frequent links are marked in the Header Table (HT) in Fig. 7. Only frequent links are used in mining. Assume $MSuppC^{link}$ = 2 and $MSuppC^{seq}$ = 3 for our example. The sequence database (Table 4) is scanned once to find counts for links and insert them in the header table. The FS-tree is built during the second scan of the database in a manner similar to WAP-tree and PLWAP-tree except that a sequence which contains non-frequent links is split and each part is inserted in the tree separately. Pointer links from the header table are built to connect occurrences of header table entries as they are inserted into the FS-tree similar to WAP-tree linkage (i.e., first occurrence based and not pre-ordered). As only frequent links may appear in frequent sequences, only frequent links are considered during mining (El-Sayed *et al*., 2004) mention four important properties of the FS-tree as follows:

- Any input sequence that has non-frequent link (s) is pruned before being inserted into the FS-tree
- If MSuppClink<MSuppCseq, the FS-tree is storing more information than required. This is a drawback as discussed in the features earlier in introduction but is required by FS-miner to support incremental and interactive mining in a manner similar to ISM (Thakur *et al*., 2006)

- All possible subsequences that end with a given frequent link h can be obtained by following the pointer of h from the header table to correct FS-tree branches
- In order to extract a sequence that ends with a certain link h from an FS-tree branch, we only need to examine the branch prefix path that ends with that link backward up to the tree root

Proposed method (hybrid 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 FSP.

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. 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).

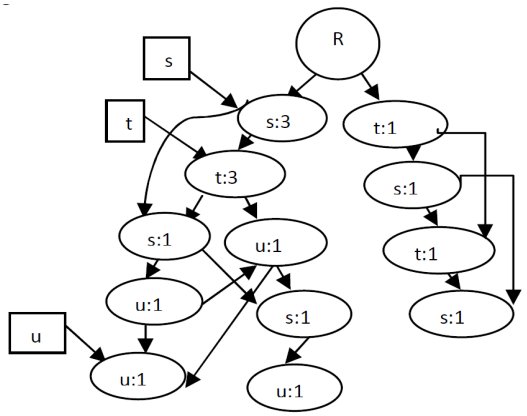

Fig. 8: Position code assignment with node position in its complete tree (binary tree)

Construction of tree: The tree data structure Fig. 8, 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.

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 an 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:

- The leftmost child of this node in the general tree is the left child of the node in the binary tree
- 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 Fig. 1. It can be transformed into its binary tree equivalent shown in Fig. 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 Fig. 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 lefttree of another node. From Fig. 8, 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).

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. 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 en is frequent. It continues the search for each next prefix event along the path, using subsequent suffix trees of some en (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 en nodes in the suffix sub-trees 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 sub-trees. 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 sub-trees 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.

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 $S^1 = S_1\ S_2$ ……S_n , WHERE
$S_1(1<=I<=n)$ are events in S1.Let current node point to the root of T.
b) for i=1 to n do ,
if cuurent_node has a child labeled S_i by 1 and make cuurent_node point Si,
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: WAP 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 ei 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 hybrid-Mine and passing R^1 and F^1.
End // Hybrid algorithm //

RESULTS

We report our experimental results on the performance of hybrid algorithm in comparison with WAP Tree and FS-Tree. It shows that our proposed algorithm is outperforms other previously proposed methods and is efficient and scalable for mining sequential patterns in large databases.

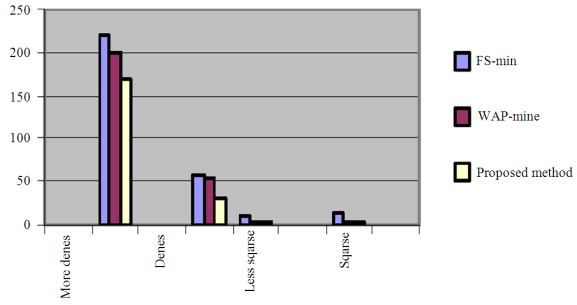

Fig. 9: Sequence Database density Vs algorithm execution time (sec), at minimum support of 1%

Table 5: Sequence database density Vs algorithm execution time (in sec), at minimum support of 1%

	Fsmine	WA pmine	Proposed method
More dense C15T10SBN20D200K	220	200	170
Dense C12T8S6N60D200K	58	55	32
Less sparse C10T6SNB0D200K	9	5	2
Sparse C8T5S4N100D200K	12	3	2

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 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 proposed algorithm always uses less runtime than the WAP algorithm (Fig 9 and Table 5). 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.

DISCUSSION

Our performance study shows that our proposed method 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. 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. This algorithm also eliminates the need to store and scan intermediate conditional pattern bases for reconstructing intermediate WAP trees.

CONCLUSION

In this study, 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 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 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

El-Sayed, M., C. Ruiz and E.A. Rundensteiner, 2004. FS-miner: Efficient and incremental mining of frequent sequence patterns in web logs. Proceedings of the 6th Annual ACM International Workshop on Web Information and Data, Nov. 12-13, ACM Press, Washington, DC., USA., pp: 128-135. DOI: 10.1145/1031453.1031477

Ezeife, C.I. and Y. Lu, 2005. Mining web log sequential patterns with position coded pre-order linked WAP-tree. Data Min. Knowl. Discov., 10: 5-38. DOI: 10.1007/s10618-005-0248-3

Han, J., J. Pei and Y. Yin, 2000. Mining frequent patterns without candidate generation. Proceeding of the 2000 ACM SIGMOD International Conference on Management of Data, May 15-18, ACM Press, Dallas, Texas, United States, pp: 1-12. DOI: 10.1145/342009.335372

Han, J., J. Pei, Y. Yin and R. Mao, 2004. Mining frequent patterns without candidate generation: A frequent pattern tree approach. Data Min. Knowl. Discov., 8: 53-87. DOI: 10.1023/B:DAMI.0000005258.31418.83

Han, J. M. Kamber and J. Pei, 2005. Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA., ISBN: 10: 1558609016, pp: 800.

Ivancsy, R. and I. Vajk, 2006. Frequent pattern mining in web log data. Acta Polytech. Hungarica, 3: 77-90. http://bmf.hu/journal/Ivancsy_Vajk_5.pdf

Madan, M. and S. Madan, 2010. Convalesce optimization for input allocation problem using hybrid genetic algorithm. J. Comput. Sci., 6: 413-416. http://www.scipub.org/fulltext/jcs/jcs64413-416.pdf

Ren, J., X. Zhang and H. Peng, 2006. MFTPM: Maximum frequent traversal pattern mining with bidirectional constraints. J. Comput. Sci., 2: 704-709. http://www.scipub.org/fulltext/jcs/jcs29704-709.pdf

Ren, J.D. and X.L. Zhou, 2006. A new incremental updating algorithm for mining sequential patterns. J. Comput. Sci., 2: 318-321. http://www.scipub.org/fulltext/jcs/jcs24318-321.pdf

Srivastava, J., R. Cooley, M. Deshpande and P.N. Tan, 2000. Web usage mining: Discovery and applications of usage patterns from Web data. ACM SIGKDD Explorat. Newslett., 1: 12-23. DOI: 10.1145/846183.846188

Thakur, R.S., R.C. Jain and K.R. Pardasani, 2006. Mining level-crossing association rules from large databases. J. Comput. Sci., 2: 76-81. http://www.scipub.org/fulltext/jcs/jcs2176-81.pdf