Mining Very Long Sequences in Large Databases with PLWAPLong [*]

C.I. Ezeife
School of Computer Science
University of Windsor
Windsor, Ontario N9B 3P4
cezeife@uwindsor.ca

Kashif Saeed
School of Computer Science
University of Windsor
Windsor, Ontario N9B 3P4
saeed4@uwindsor.ca

Dan Zhang
School of Computer Science
University of Windsor
Windsor, Ontario N9B 3P4
zhang3d@uwindsor.ca

ABSTRACT

Position Coded Pre-order Linked Web Access Pattern (PLWAP) mining algorithm is one of the existing efficient web sequential pattern mining algorithms, which stores the frequent sequences of the entire sequential database in a compressed tree form with position coded nodes. However, for very long sequences exceeding thirty two nodes, the number of bits an integer position code can hold, the PLWAP algorithm's performance begins to degrade because it employs linked lists to store conjunctions of long position codes and the linked list traversals slow down the algorithm both during tree construction and mining. PLWAP algorithm also uses each and every node in the frequent 1-item event queue to test for that event inclusion in the suffix tree root set during mining.

This paper proposes (1) using a different position code numbering scheme where each node is assigned two numeric codes (startPosition, endPosition) instead of one, (2) using pre-knowledge of "Last Descendant" of each tree branch to lower the cost of creating the suffix tree root sets during mining. Experiments show that the proposed new scheme, the PLWAPLong outperforms the PLWAP for long sequences and large databases as well as regular databases.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: [Database Applications, Data Mining]; K.6.5 [**Management of Computing and Information Systems**]

[*]Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Editor: Bipin C. DESAI;
Copyright ©2009 ACM 978-1-60558-402-7/09/09 $5.00

IDEAS09 2009 September 16 - 18, Cetraro, Calabria [Italy]
.

General Terms

Algorithms, Design, Performance

Keywords

Data mining, Association Rule Mining, Long Sequences, PLWAP Mining

1. INTRODUCTION

Algorithms for mining frequent sequential patterns from sequential databases include GSP [1], PrefixSpan [8], SPADE [10], WAP [9] and PLWAP [2]. Discovering sequential patterns in web server logs is one application of sequential pattern mining and is helpful in predicting visiting patterns of web users, and in turn helping targeted advertisement or grouping related information based on frequent sequential web accesses. Sequential pattern mining discovers frequent subsequences as patterns in a sequence database, which stores a number of records that are ordered sequential events. Sequential pattern mining discovers frequent sequential patterns by applying association rule mining methods to discover rules of the form $X \rightarrow Y$ at a given minimum support frequency, where both X and Y are sequences of candidate 1-items from the set of possible items (events). Example applications of sequential pattern mining include: analyses of customer purchase behavior, web access patterns, scientific experiments, disease treatments, natural disaster, and protein formations [7]. Before sequential pattern mining on web log sequence database, the raw web log is first pre-processed to produce a sequential database in the form of a transactional data for mining. A line of data in the raw web log data has the format:
137.207.76.120-[30/Aug/2007:12:03:24-0500] "GET/jdk1.3/docs/relnotes/deprecatedlist.html HTTP/1.0 200 2781", where 137.207.76.120 is the host/ip, '-' represents anonymous user, [30/Aug/2007:12:03:24-0500] is the [date:time], "GET/jdk1.3/docs/relnotes/deprecatedlist.html HTTP/1.0" is the "request url", '200' is the status of the URL request and 2781 is the number of bytes requested. An example sequential database obtained after pre-processing the web log, which is used for sequential pattern mining is shown as Table 1 (columns 1 and 2). From Table 1, the 1-items representing each web page accessed by each user is the set {a, b, c, d, e, f} and there are only four user page sequence accesses in this simple example. User with transaction ID (TID) 100 has accessed page 'a' before 'b', then page 'd' before revisiting page 'a' and finally before accessing page 'c'. Given the minimum support threshold,

Table 1: Example Sequential Database for Mining

TID	DB Sequences	Frequent Sequences (Seq list)
100	abdac	abac
200	eaebcac	abcac
300	babfaec	babac
400	afbacfc	abacc

all subsequences with frequency occurrence in the database greater or equal to the given minimum support frequency are considered frequent. And from such frequent patterns, strong rules with confidence greater than or equal to the minimum confidence can be generated in the second phase of pattern mining. The relative support percentage of an *n-sequence* is given as the number of records in the database containing the n-sequence divided by the total number of records in the database times 100. The absolute support of an n-sequence is also simply taken as the number of records in the database containing the n-sequence. For example, given the minimum support threshold of 75% or 3 out of 4 transactions, since the 1-sequence *a* has a support count of 4, then, *a* is a frequent pattern. Another sequence with a support count of 4 is *aba* which shows that sequences are mined with gaps because although TID 100 has the original record as abdac, the subsequences can skip some events and can be repeated.

An efficient web sequential pattern mining algorithm is the PLWAP or the Pre-order Linked Web Access Pattern Tree [6], [2], [3]. The PLWAP algorithm eliminates the need for recursive re-construction of intermediate WAP-trees during mining by assigning unique binary position code to the nodes of the tree. The binary codes help in determining the suffix tree for any frequent pattern prefix as well as ancestor and descendant relationships of the nodes on the suffix tree for quick prefix-first pattern growth mining using the only one original PLWAP tree.

However, for very long sequences exceeding thirty two nodes, the number of bits an integer position code can hold, the PLWAP algorithm's performance begins to degrade because it employs linked lists to store conjunctions of long position codes and the linked list traversals slow down the algorithm both during tree construction and mining [3]. PLWAP algorithm also uses each and every node in the frequent 1-item event queue to test for that event inclusion in the suffix tree root set during mining. This is a very expensive operation since except for one node, all other nodes that are its ancestors and descendants are not included in the root set.

1.1 Contributions and Outline

This paper proposes the PLWAPLong algorithm for very long sequences, which provides the following extensions to the PLWAP algorithm: (i) replaces the PLWAP's position code numbering scheme of one code per node to two numeric position codes of startPosition and endPosition per node. (ii) using knowledge of "Last Descendant" of each tree node to lower the cost of creating the root sets and to eliminate the unwanted nodes from ancestor/descendent tests during mining. The objectives of the proposed techniques are:
1. Enabling mining of long sequences which from the literature are not effectively handled by existing sequential pattern mining algorithms.
2. Improving Mining Efficiency: by reducing the CPU execution time for mining of large databases.

Section 2 presents related work, Section 3 presents the proposed system, the PLWAPLong. Section 4 describes the experimental results, while section 5 presents conclusions and future work.

2. RELATED WORK

Existing sequential pattern mining algorithms include GSP [1], PrefixSpan [8], SPADE [10], WAP [9], and PLWAP [2]. The GSP [1] is an Apriori based sequential pattern mining algorithm [5], which generates the next level candidate (k+1)-sequences C_{k+1} from the seed frequent k-sequences L_k through the operation L_k GSP-join L_k. The GSP-join, like the Apriori-generate join requires that two sequences in L_k join with each other if two conditions are met. Here, a sequence s_1 (e.g., ba) in L_k joins with another sequence s_2 (e.g., ab) in L_k if the subsequence obtained by dropping the first (k-1) items of s_1 is the same as the subsequence obtained by dropping the last (k-1) items of s_2, the candidate sequence generated is the sequence s_1 extended with the last item in s_2 (e.g., bab is the result of joining ba and ab) and is added to C_{k+1}. After pruning step to remove candidate (k+1)-sequences not meeting the Apriori property, the frequent (k+1)-sequences are computed from candidate sequences C_{k+1} by scanning the database for their supports and keeping those with support greater or equal to minsupport.

SPADE relies on lattice theory to generate candidate sequential patterns. The database for sequence mining consists of a collection of sequences. Each input sequence has a unique record identifier called SID, and a sequence of events $e_1e_2e_3 \ldots e_n$, where each event e_i consists of a set of items from the candidate itemsets and each event has a time stamp represented as an EID. SPADE only needs to access the original database 3 times for support counting. One for computing 1-sequence, one for computing 2-sequence and another for enumerating all other sequences via breadth-first search and depth-first search.

PrefixSpan [8] recursively constructs patterns by growing on the prefix, and simultaneously, restricting the search to projected databases.

The WAP-tree algorithm or Web Access Pattern Tree was developed by [9] for mining the sequential patterns from web logs in non-Apriori like fashion similar to the FP-tree approach [4]. The WAP algorithm transforms the database into a compact WAP tree structure that stores only frequent sequences for each transaction id from the Root node to the leaf node and maintains a frequent 1-sequence header linkage in the order the sequences are inserted in the tree. Recursive mining of the original WAP tree starting from the suffix frequent 1-sequences, generating of conditional pattern bases and intermediate WAP trees at each phase of recursive mining, would result in complete list of all frequent patterns.

The PLWAP or the Pre-order Linked Web Access Pattern Tree algorithm [2] transformed the WAP tree by assigning binary position codes to the nodes of the tree and performing pre-order (visit root, then left, then right) frequent 1-sequence header linkage in order to cut off the need for recursively constructing intermediate conditional pattern bases and WAP tree during mining. Given a sequence database DB (e.g., Table 1 columns 1 and 2) and a minimum support threshold s (e.g., 75%), the steps in mining frequent

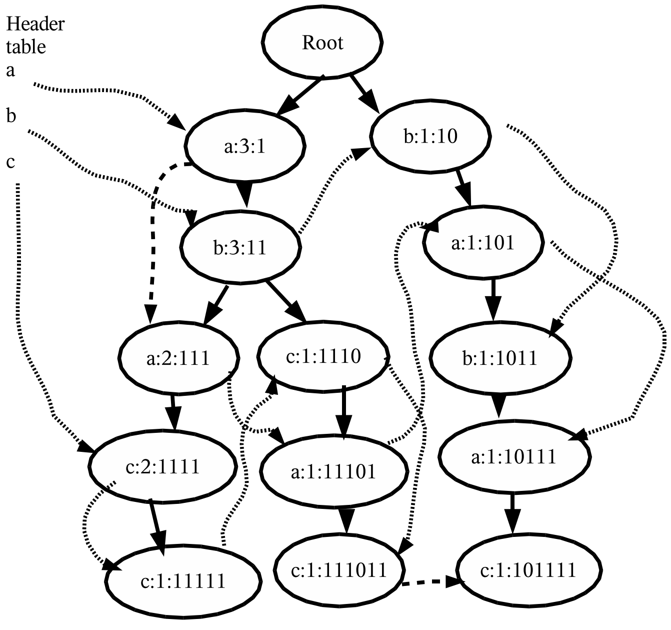

Figure 1: The PLWAP Tree for the Example Database

sequences from DB using the PLWAP algorithm are:
1. Scan the DB once to compute the frequent 1-sequence F_1 as those 1-sequence with support count greater or equal to 75% or 3 out 4 records in the example database of Table 1 columns 1 and 2. This gives F_1 = {a:4, b:4, c:4} with the counts of the items listed next to them.
2. Build the PLWAP tree by first scanning the database DB a second time to create the frequent sequence of each record FS (shown in column 3 of Table 1) by deleting all non-frequent 1-sequence from the original sequence in column 2. Then, insert each frequent sequence or FS record from Root to leaf node of the PLWAP tree where each node is labeled as (node item: count: position code). The position code of the Root node is null and thereafter, the PLWAP assigns to the leftmost child of a node, its parent node's position code with binary '1' appended to it, but assigns to any other child of a parent node, the position code of the child's nearest left sibling node with binary '0' appended to it. Thus, the immediate children of the Root node from left will have position codes of 1, 10, 100, 1000 and so on and children of the leftmost child of Root will have position codes of 11, 110, 1100 and so on. After inserting the FS in the tree, the frequent 1-sequence header linkages are built pre-order fashion (root, left, right). The PLWAP tree obtained for the example database is shown as Figure 1. The header linkages are shown with broken lines starting from the F_1 event, and are used during mining to track all events of this F_1 type. For example, a links to all a-node on the tree.
3. Mine the PLWAP tree by first mining prefix frequent 1-sequence and using the original PLWAP tree, the frequent 1-sequence header linkages, and the position codes. PLWAP Algorithm starts mining with the first element from the header linkage table. In our example it is 'a'. Following the 'a' linkage, the first occurrence of 'a' node in the two suffix trees of the Root are found at nodes a:3:1 and a:1:101. Since the sum of the counts of both these nodes is greater than the minimum support, 'a' is considered as frequent 1-sequence. Next, the algorithm looks at 2-sequence with prefix sequence 'a'. The 'a' suffix trees of a:3:1 and a:1:101 rooted at b:3;11 and b:1:1011 are mined. The first occurrences of 'a' in these suffix trees are found at nodes a:2:111, a:1:11101 and a:1:10111. Since the frequency count of these nodes at different branches of the PLWAP tree at this level being mined, is higher than the minsupport of 3, hence 'a' is appended to the last list of frequent sequence 'a' forming 'aa' frequent sequence. The algorithm next mines the suffix trees of nodes a:2:111, a:1:11101 and a:1:10111 from the last step. The roots of these suffix trees c:2:1111, c:1:111011 and c:1:101111 give 'c' frequent event to make 'aac' frequent sequence. The last suffix tree is c:1:11111 which is not frequent hence 'aacc' is not frequent and this causes this 'a' track recursive search to backtrack to look for the next frequent 1-sequence 'b' suffix trees in order to check if 'ab' is frequent. Following the 'b' header linkage, the algorithm finds b:3:11 and b:1:1011 and generates 'b' frequent event giving 'ab' frequent sequence. The algorithm progresses and finds other frequent sequences with 'ab' as their prefix sequence, i.e, 'aba', 'abac' and 'abc'. The algorithm backtracks to find frequent sequences that have 'c' as prefix event. Algorithm finds frequent event 'c' from c:2;1111, c:1:111011 and c:1:101111 to give 'ac' as the frequent sequence. This completes finding all the frequent sequences that have 'a' as their prefix. The PLWAP algorithm then finds the frequent sequences starting with 'b' and 'c'. The complete set of frequent sequences found by PLWAP are {a,aa,aac,ab,aba,,abac,abc,ac,b,ba,bac,bc,c}.

3. THE PROPOSED POSITION CODED PRE-ORDER LINKED WAP-TREE LONG (PLWAP-LONG)

Section 3.1 presents the algorithm PLWAPLong, being proposed for enabling effective and efficient mining of frequent long sequences using the PLWAP mining approach. Section 3.2 discusses the implementation details of the algorithm. section 3.3 presents an application of the proposed algorithm on our example database.

3.1 PLWAPLong for Mining Long Sequences

The proposed PLWAPLong algorithm being proposed eliminates two main problems that degrade the performance of the PLWAP and in particular, when processing long sequences with more than 32 nodes or events.
1. PLWAP algorithm's implementation represents binary position code of each node by storing its binary code in a linked list data structure. However, for very long sequences exceeding thirty two nodes, the number of bits an integer position code can hold, the PLWAP algorithm's performance begins to degrade because the linked list traversals slow down the algorithm both during tree construction and mining [3].
2. In the PLWAP algorithm's implementation, during construction of the suffix trees, in order to define an event root set (e.g., 'a' root set), which consists of all first nodes of this event on different branches of the tree, the ancestor-descendant relationship between the first found root set element and all other events in its header linkage, has to be checked. This is identified as making a lot of unnecessary

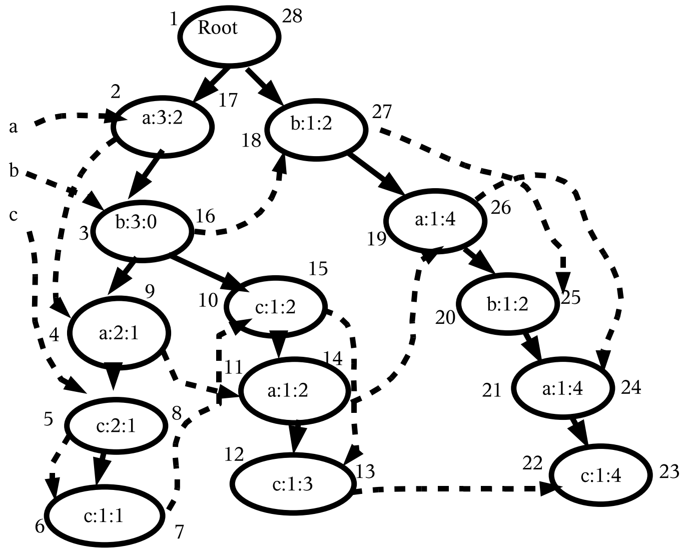

Figure 2: The PLWAPLong Tree Nodes with Two Position Codes and inside annotation as node label:count:lastDesc

checks. This is because all events, with the same label, that are descendants of the event for which suffix tree is being explored, should not be tested for this relationship as they will never be counted in the suffix tree support. When dealing with long sequences where branches have hundreds of event nodes having repeated events, the algorithm will perform relationship checks just to ignore their support count. These support checks affect the performance when we have very long sequences.

Proposed Solution
Solution for Problem 1: To overcome the first problem, the PLWAPLong proposes a new position code numbering scheme. The new approach uses two labels instead of one for each tree node as 'startPosition' and 'endPosition'. The startPosition and endPosition codes of the tree nodes are assigned by traversing the PLWAPLong tree in pre-order fashion (root, left, right) starting with the Root node. Thus, while the original example PLWAP tree is in Figure 1 for the example database, the resulting newly coded PLWAPLong tree is shown as Figure 2. From Figure 2, the startPosition code for the Root node is 1, while the Root's endPosition code is 28. The startPosition and endPosition codes of the a:3 leftchild of the Root are 2 and 17 respectively. The startPosition code is labeled on the left side of the node, the endPosition code is labeled on the right side of the node, the event label, its count and its last descendant are labeled inside the node. The rule that is used during the mining process to determine the ancestor-descendant relationship of any two nodes and relevant definitions are presented next.

Some Definitions and Rules

DEFINITION 3.1. startPosition of a PLWAPLong node p *is the unique number assigned to node p during tree descent while traversing the PLWAPLong tree pre-order fashion starting from the Root node which has a startPosition of 1. For example, the immediate left child of Root node will have a startPosition of 2.* ■

DEFINITION 3.2. endPosition of a PLWAPLong node p *is the unique number assigned to node p during tree ascent while traversing the PLWAPLong tree pre-order fashion starting from the Root node which has a startPosition of 1. For example, the immediate left child a:3:1 of Root node for the example PLWAP tree will have an endPosition of 17 while Root has an endPosition of 28.* ■

DEFINITION 3.3. lastDescendant (lastDesc) of a PLWAPLong node p *is pointer to the node with position index of the event node q in header linkage, that is of the same type as event p, such that q appears last on the same branch of the PLWAPLong tree as event p. If there is no other descendant event of p on the same branch, then, p becomes its own last-descendant. For example, the lastdescendant of node a:3:2 of the example PLWAPLong tree is a:1:2 which has position index 2 on the frequent header link and thus lastdescendant of a:3:2 is 2, while the lastdescendant of a:2:1 is 1 as shown in Figure 2 where nodes are annotated inside as node label:count:lastDesc.* ■

DEFINITION 3.4. The Suffix Root Set of a set of PLWAPLong node p *is taken to be the set of p's children nodes as in the original PLWAP.* ■

RULE 3.1. "Given two nodes n_1 and n_2, n_1 is ancestor of n_2 (or n_2 is descended of n_1) *if n_1.startPosition $<$ n_2.startPosition and if n_1.endPosition $>$ n_2.endPosition"*. ■

The Main PLWAPLong Algorithm
Given a sequence database DB, the process of finding all frequent sequential patterns at a minimum support threshold of s with the proposed PLWAPLong algorithm, has the following four main sequence of steps.
1. Scan DB once, find all frequent F_1 events with support $\geq$ s.
2. Scan DB again, build PLWAPLong tree with seq list (which is, frequent sequence version of DB, e.g., column 3 of Table 1), assigning two position codes and preordered F_1 header linkage after pre-order traversal.
3. Build Last Descendant, by calling buildDesc(PLWAPLong, F_1 header linkage), for each event node using its suffix trees and the F_1 header linkages.
4. Mine the PLWAPLong tree using lastDescendant.
Solution for Problem 2:
This solution addresses the second problem, which requires avoiding unnecessary ancestor-descendant checks of all tree nodes in the F_1 event frequent header linkage, during mining.
Build Last Descendant
After constructing the linked list PLWAPLong tree, which has nodes with two unique codes of startPosition and endPosition, buildDesc() method is called to create the last descendant of each node. Formal definition of this method is given as Algorithm 1. Rule 1 is applied to check if a node q like a:1:4 with startPosition and endPosition codes of 19 and 26, is a descendant of another node p, like a:3:2 with startPosition and endPosition codes of 2 and 17 as follows. Node p is an ancestor of node q only if p.startPosition $<$ q.startPosition and p.endPosition $>$ q.endPosition. For nodes a:3:2 with (2,17) and a:1:4 with (19,26), we can see

that while $2 < 19$, 17 is not > 26, and thus, we conclude that node a:1:4 is not a descendant of node a:3:2 as also can be seen in the tree of Figure 2.

Algorithm 1. *(buildDesc(): Compute the lastDescendant of Tree nodes)*

```
Algorithm buildDesc()
Input:    PLWAPLong , $F_1$ header linkages
Output: All nodes in PLWAPLong assigned lastDescendant
Other variables: unassignedNode stack to track nodes not
          assigned lastDesc.
begin
/* Compare each event $e_j$ in PLWAPLong with other events in */
/* its $F_1$ header linkages so that the index of the last event of */
/* its kind on its subtree is filled as $e_j$'s lastDesc. */
1. for each event, $e_i$, in $F_1$ link header list
    1.1. for index j=0 to number of $e_i$ events
        1.1.1. If $j + 1 > numberofe_i$ events, set $e_j$.lastDesc = j
        1.1.1.1. if stack is not empty
            Set (stack.pop).lastDesc = j
        1.1.2. If $e_j$ is ancestor of $e_{j+1}$, push ($e_j$) to stack
        1.1.3. Else set $e_j$.lastDesc = j
        1.1.3.1. if stack is not empty
        while stack.top not ancestor of $e_{j+1}$)
        Set (stack.pop).lastDesc = j
end
```

The PLWAPLong-Mine Algorithm

The step four of the PLWAPLong algorithm involves mining the PLWAPLong tree using the lastDescendant. The mining process proceeds recursively from Root the same way the PLWAP-Mine occurs, except that the proposed PLWAPLong-Mine now finds next suffix root set of node by adding 1 to the lastDescendant of node p, so that the total support count of a pattern being mined is quickly obtained. The formal algorithm PLWAPLong-Mine algorithm is presented as Algorithm 2.

Algorithm 2. *(PLWAPLong-Mine: Mining with PLWAPLong Tree)*

```
Algorithm PLWAPLong-Mine()
Input:    PLWAP tree T, header linkage table L,
          minimum support $\lambda$ ($0 < \lambda \leq 1$),
          Frequent m-sequence F)
          suffix tree roots set R (R includes root and F is
          empty first time algorithm is called).
          Frequent m-sequence F=$\emptyset$, Rootset R={Root},
Output: Frequent (m+1)-sequences, F'
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
(1) If R is empty, return
(2) For each event, $e_i$ in L, find the suffix tree of
$e_i$ in T (i.e, $e_e|suffixtree$), do
    (2a) Save first event in $e_i$-queue to S.
    (2b) 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 $R'$
                    Jump to the last descendant of $e_i$
                    Add count of $e_i$ to C
                    Replace the S with $e_i$
    (2c) If C is greater than $\lambda$
            Append $e_i$ after F to $F'$ and output $F'$
            Call Algorithm PLWAPLong-Mine of Figure 2
            passing $R'$ and $F'$.
end // of PLWAPLong-Mine //
```

Application of the PLWAPLong Algorithm

EXAMPLE 1: Given the example database of Table 1, mine frequent patterns from this database at minimum support threshold of 75% using the PLWAPLong algorithm.

SOLUTION 1: Applying step 1 of the PLWAPLong algorithm discussed in section 3.1 scans the database of Table 1 (columns 1 and 2) to generate the frequent F_1 list as {a:4, b:4, c:4}. In step 2, the algorithm scans the database to build the PLWAPLong tree using the frequent transactions (which consist of original sequence with only F_1 events retained) shown in Table 1 (column 3). This construction is done the same way the PLWAP tree is built except that each frequent event is assigned two position codes of startPosition indicated on the left side of the node and endPosition indicated on the right side of the node. First the PLWAPLong tree is built and then, it is traversed pre-order fashion to assign the startPosition and endPosition codes. The Root is always assigned the startPosition code of 1, then, during tree descent, the startPosition codes are assigned, such that in PLWAPLong tree of Figure 2, the startPosition of 2 is assigned to the "a:3:2" left child of Root and the same node is assigned the endPosition code of 17.
In step 3 of the PLWAPLong algorithm, the tree is traversed pre-order fashion using the F_1 header linkage list, in order to compute the last descendant of each event, which is a pointer to the last event of its kind on the same subtree and which has its position index computed. If the event has no descendant, then, its last descendant index becomes its own index and it points to itself. The algorithm for computing the last descendant is given as Algorithm 1. The F_1 header linkages of the PLWAPLong tree of Figure 2 is shown in Table 2 in the format $< event >< count >< startPosition >< endPosition >< parent >< leftchild >< rightsibling >< lastDesc >$. This representation more clearly shows how the last descendant of each node is derived. Note that the null pointers are represented on the table as zero (0) and the Root is represented as *Rt*.

Since from this table, the last descendant of node a:3:2 left child of Root is the node a:1:2 left child of c:1:2, then, the last descendant of a:3:2 is the array index of this a:1 node, which is 2.
In step 4, the algorithm PLWAPLong-Mine(PLWAPLong, Frequent m-sequence F) is called to compute the frequent patterns with the PLWAPLong tree. The mining is done the same way the PLWAP tree is mined, except that the PLWAPLong algorithm quickly uses the already computed last descendant of each node to get the set of suffix tree root set, whose supports should be added to confirm a pattern frequent or not. For example, from Figure 2, mining starts with Root, following the a-header linkage. The two suffix tree Root sets are for a:3:2 and b:1:2. To confirm that the 1-sequence a is a frequent pattern (FP), the total support counts of all the first a nodes on different branches of these 2 suffix tree root sets, have to be greater than or equal to the minimum support count of 3. Since a:3:2 and a:1:4 produce a total count of 4, then, a is confirmed an FP. The PLWAPLong algorithm uses the last descendant of a:3:2 to jump to the a node at index position 2+1 = 3 and this a node happens to be a:1:4. This way, it has skipped checking all chain of a-events in all suffix trees to find the relevant events on different suffix trees that should count. This same process is applied recursively to retrieve all frequent patterns.

Table 2: PLWAPLong Tree F_1 Header Linkages

It	Index				
em	0	1	2	3	4
a	$< a >$	$< a >$	$< a >$	$< a >$	$< a >$
	$< 3 >$	$< 2 >$	$< 1 >$	$< 1 >$	$< 1 >$
	$< 2 >$	$< 4 >$	$< 11 >$	$< 19 >$	$< 21 >$
	$< 17 >$	$< 9 >$	$< 14 >$	$< 26 >$	$< 24 >$
	$< Rt >$	$< b : 3 >$	$< c : 1 >$	$< b : 1 >$	$< b : 1 >$
	$< b : 3 >$	$< c : 2 >$	$< c : 1 >$	$< b : 1 >$	$< c : 1 >$
	$< b : 1 >$	$< c : 1 >$	$< 0 >$	$< 0 >$	$< 0 >$
	$< 2 >$	$< 1 >$	$< 2 >$	$< 4 >$	$< 4 >$
b	$< b >$	$< b >$	$< b >$		
	$< 3 >$	$< 1 >$	$< 1 >$		
	$< 3 >$	$< 18 >$	$< 20 >$		
	$< 3 >$	$< 18 >$	$< 20 >$		
	$< 16 >$	$< 27 >$	$< 25 >$		
	$< a : 3 >$	$< Rt >$	$< a : 1 >$		
	$< a : 2 >$	$< a : 1 >$	$< a : 1 >$		
	$< 0 >$	$< 0 >$	$< 0 >$		
	$< 0 >$	$< 2 >$	$< 2 >$		
c	$< c >$	$< c >$	$< c >$	$< c >$	$< c >$
	$< 2 >$	$< 1 >$	$< 1 >$	$< 1 >$	$< 1 >$
	$< 5 >$	$< 6 >$	$< 10 >$	$< 12 >$	$< 22 >$
	$< 8 >$	$< 7 >$	$< 15 >$	$< 13 >$	$< 23 >$
	$< a : 2 >$	$< c : 2 >$	$< b : 3 >$	$< a : 1 >$	$< a : 1 >$
	$< c : 1 >$	$< 0 >$	$< a : 1 >$	$< 0 >$	$< 0 >$
	$< 0 >$	$< 0 >$	$< 0 >$	$< 0 >$	$< 0 >$
	$< 1 >$	$< 1 >$	$< 3 >$	$< 3 >$	$< 4 >$

Another difference with the PLWAPLong is its use of two codes to efficiently handle long sequences.

4. EXPERIMENTS AND PERFORMANCE ANALYSIS

To test the proposed PLWAPLong algorithm, we conducted experiments to
(1) determine performance gain in terms of CPU execution time of the PLWAPLong algorithm in comparison with the PLWAP algorithm for long sequences of over 32 events in a sequence for large databases.
(2) determine performance gain in terms of CPU execution time of the PLWAPLong algorithm in comparison with the PLWAP algorithm for long sequences in regular sized databases.
(3) determine memory usages of the proposed PLWAPLong in comparison with the PLWAP algorithm.

Comparing PLWAPLong and PLWAP Execution Times
Since a recent extensive sequential pattern mining survey [7] shows the PLWAP algorithm outperforming other sequential pattern mining algorithms including PrefixSpan [8] in most circumstances, thus, the two algorithms PLWAP and our proposed PLWAPLong algorithm are compared in this study to demonstrate the performance gain over the PLWAP by the new version of the algorithm. Both the PLWAP and our proposed PLWAPLong algorithm were implemented in C++ with the same data structure. Then, the CPU execution times for the two algorithms were tested for transactional databases of different sizes ranging from 400K to 1 million records generated for both long sequences of over 32 events and regular sized databases of between 2K and 14K for long sequences of more than 32 events, with the IBM quest synthetic data generator publicly available at at http://www.almaden.ibm.com/cs/quest/ and used by other pattern mining research. The characteristics of the generated data are described as follows: $|D|$: Number of records in the database, $|C|$: Average length of the records, $|S|$: Average length of maximal potentially frequent itemset, $|N|$: number of items (attributes).
For example, C10.S5.N20.D600K describes dataset with $|C| = 10$, $|S| = 5$, $|N| = 20$, and $|D| = 600K$. All experiments are performed on a more powerful multiuser UNIX SUN microsystem with a total of 16384 MB memory and 8 x 1200 MHz processor speed, which generally produces faster execution times than when run on microcomputer environment. The minimum support for the experiments range between 10% and 50% and the lower the minimum support, higher the number of frequent patterns found and more work the algorithms will be doing. The result of the experiments are presented in Table 3 and Figure 3 for long sequences and very large database of C39.S10.N45.D1M. Table 4 and Figure 4 compare the performances of the two algorithms for varying database sizes of between 400K to 900K records at a fixed minimum support threshold of 50%. Table 5 and Figure 5 present the comparative performances for smaller sized databases with long sequences of around C39.S10.N45.D10K at a low minimum support of 15%.

The proposed PLWAPLong algorithm provides between 30% to 41% improvement on the performance of the PLWAP algorithm when processing long sequences and in particular at low minimum support when many frequent patterns are present. This is because it has eliminated endless chaining of position codes to handle long sequences and also irrelevant ancestor/descendant node search during mining. It can be seen that difference in performance of the two compared algorithms increases as the size of data increases and as minimum support threshold decreases. Experiments on memory usages also show the proposed PLWAPLong algorithm uti-

Table 3: Execution times for dataset 1M at different MinSupports(Long Sequences)

Algorithms	Runtime (in secs)						
MinSupport	30%	35%	40%	45%	50%	55%	60%
Number of FPs	282	124	61	47	32	16	7
PLWAP	13749	6799	3512	2622	1571	570	250
PLWAPLong	8095	4281	2369	1852	1029	462	235

Table 4: Execution times for varying data sizes at Minsupport 50%(Long Sequences)

Algorithms	Runtime (in secs)					
DB size	400K	500K	600K	700K	800K	900K
PLWAP	525	876	1057	1207	1420	1581
PLWAPLong	376	600	714	815	960	1060

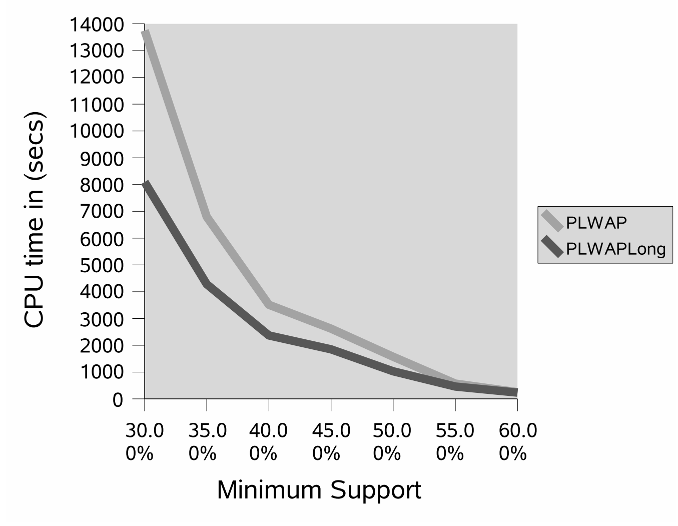

Figure 3: Execution times for dataset 1M at different MinSupports(Long Sequences)

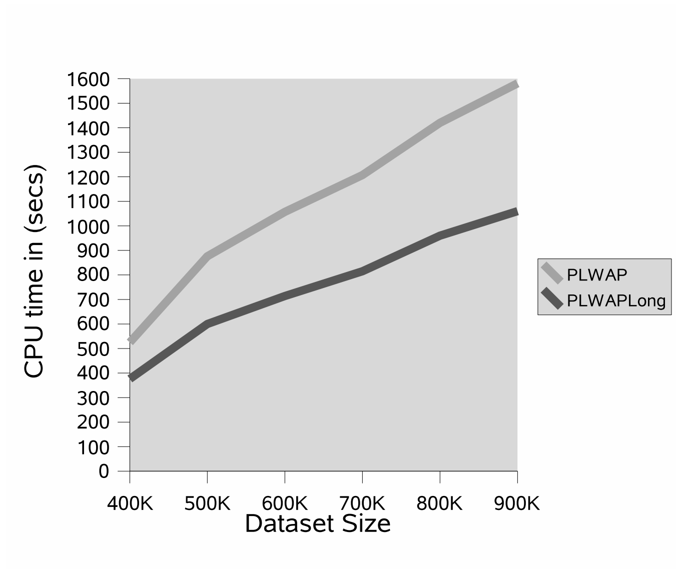

Figure 4: Execution times for varying data sizes at Minsupport 50%(Long Sequences)

Table 5: Execution times for Small Data Sizes at 15% Minsupport(Long Sequences)

Algorithms	Runtime (in secs)						
DB Size	2K	4K	6K	8K	10K	12K	14K
# of FPs	5159	4707	4633	4417	4708	4686	4413
PLWAP	225	583	942	1248	1754	2163	2388
PLWAPLong	148	310	560	746	990	1228	1372

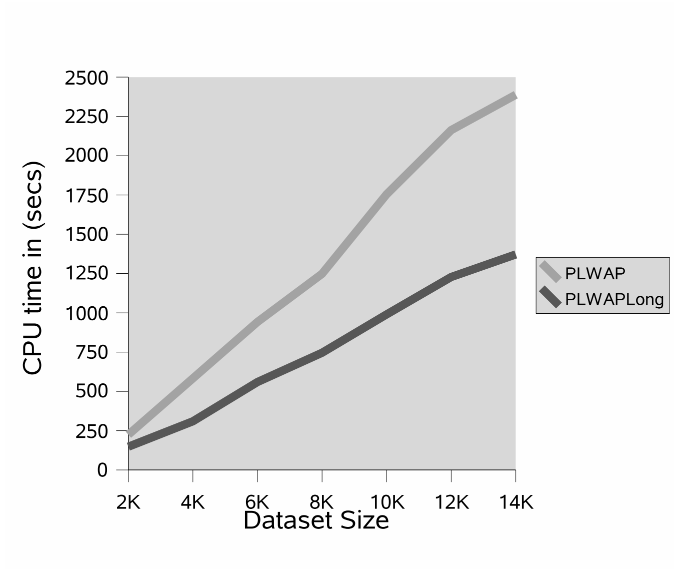

Figure 5: Execution times for Small Data Sizes at 15% Minsupport (Long Sequences)

lizing less memory than the PLWAP algorithm since it just needs only two codes for each node while PLWAP may need a chain with 10 codes for one node.

5. CONCLUSIONS AND FUTURE WORK

This paper presents a new algorithm PLWAPLong that is an extension of the PLWAP algorithm for efficiently mining very long sequences. PLWAPLong adapts the PLWAP tree structure to use two position codes for each node for purposes of identifying ancestor/descendant as well as sibling relationships during mining. In order to avoid expensive and useless comparisons of event nodes to determine the suffix trees as done by PLWAP algorithm, PLWAPLong algorithm uses 'Last Descendant' technique that quickly eliminates the unwanted nodes from ancestor/descendant comparisons and jumps to the next suffix root set member to continue finding the suffix tree. Experiments show that mining sequences using PLWAPLong algorithm is more efficient than PLWAP algorithm, especially when the sequences become longer and the web access sequence database becomes larger. Conventional sequential pattern mining algorithms do not work well with biosequences because they have small alphabet and very large sequence length. Future work involving PLWAPLong algorithm could explore the possibilities of applying PLWAPLong algorithm with such biosequences.

6. ACKNOWLEDGMENTS

This research was supported by the Natural Science and Engineering Research Council (NSERC) of Canada under an operating grant (OGP-0194134) and a University of Windsor grant.

7. REFERENCES

[1] R. S. R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the Fifth International Conference On Extending Database Technology (EDBT '96) Avignon France*, pages 3–17, 1996.

[2] C. Ezeife and Y. Lu. Mining web log sequential patterns with position coded pre-order linked wap-tree. *International Journal of Data Mining and Knowledge Discovery (DMKD) Kluwer Publishers*, 10(1):5–38, 2005.

[3] C. Ezeife, Y. Lu, and Y. Liu. Plwap sequential mining: Open source code. In *Proceedings of the ACM SIGKDD's Open Source Data Mining Workshop on Frequent Pattern Mining Implementations, Chicago*, pages 26–29. ACM, August 2005.

[4] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *International Journal of Data Mining and Knowledge Discovery*, 8(1):53–87, Jan 2004.

[5] T. Imielinski, A. Swami, and R. Agarwal. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD conference on management of data*, pages 207 – 216. ACM, 1993.

[6] Y. Lu and C. Ezeife. Position coded pre-order linked wap-tree for web log sequential pattern mining. In *Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2003)*, pages 337–349. Springer, May 2003.

[7] N. Mabroukeh and C. Ezeife. Taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys Journal*, 2009.

[8] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *2001 International Conference on Data Engineering (ICDE'01), Heidelberg, Germany*, pages 215–224, 2001.

[9] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. 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.

[10] M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42:32–60, 2001.