

# Mining Web Access Patterns with First-Occurrence Linked WAP-Trees\*

Peiyi Tang  
Dept of Computer Science  
Univ of Arkansas at Little Rock  
Little Rock, AR 72204

Markus P. Turkia  
Dept of Computer Science  
Univ of Arkansas at Little Rock  
Little Rock, AR 72204

Kyle A. Gallivan  
School of Computational Science  
Florida State University  
Tallahassee, FL 32306

## Abstract

### Abstract

In this paper, we describe the concept of first-occurrence and present a web access pattern mining algorithm based on it using a novel first-occurrence linked WAP-tree (FLWAP-tree). The first-occurrences of all symbols in the base WAP-tree of the database can be found by a pre-order traversal of a portion of the WAP-tree. The frequent patterns and their projection databases can be found quickly by following the first-occurrence links rather than following the links of the whole WAP-tree as in the PLWAP-tree mining [1]. The performance evaluation shows that our FLWAP-tree mining outperforms the PLWAP-tree mining [1] consistently and significantly.

## 1 Introduction

With the explosive growth of Internet use, mining frequent access patterns from huge datasets of web log files becomes important. The frequent web access patterns mined from web logs can help web masters and designers to understand the behaviors of web surfers on their web sites. This understanding and knowledge is essential to further improve the business of the company and the design of the web sites.

Web access pattern mining (or web log mining) is an example of general sequential pattern mining, where an event is an access of a URL. The web access sequence database is a multi-set of web access sequences, each of which, in turn, is a sequence of events of web accesses over a period of time. Another example of frequent pattern mining is the mining of biological sequences where each sequence is a sequence of amino acids or nucleotides.

Due to the importance of the problem and its large number of applications, web access pattern mining has attracted significant attention in the recent years [1, 2, 3, 4, 5, 6, 7]. The most notable algorithms

of web access pattern mining include the apriori-based algorithm GSP [8] and two pattern-growth algorithms: the WAP-tree algorithm [2] and the PLWAP-tree algorithm [1]. A WAP-tree is an aggregate tree [9] that represents the web access sequence database. All nodes with the same label are linked when the tree is built [2]. In [1], it was proposed to link all the nodes of same label in the order determined by a pre-order traversal of the tree after it is built. Such a tree is, thus, called *Pre-Order Linked WAP-tree* (PLWAP-tree). The mining algorithms in both [2] and [1] use recursive conditional searching of projection databases to find frequent web access patterns. The difference is that the WAP-tree mining [2] grows the suffix of frequent patterns, while the PLWAP-tree mining [1] grows the prefix of the frequent patterns. It is shown in [1, 6] that the mining time of the PLWAP-tree algorithm is less than that of the WAP-tree algorithm [2] and the GSP algorithm [8].

In this paper, we first present a formal framework and the theory of conditional searching on which pattern-growth mining algorithms are based. This framework enables us to reason about and express the pattern-growth mining rigorously in the form of abstract algorithm. We then present our pattern-growth algorithm using our *First-Occurrence Linked WAP-Tree* (FLWAP-tree). A FLWAP-tree is different from a PLWAP-tree [1] in that it links only the *first-occurrences* instead of *all* occurrences of each symbol in the tree. Our experiments show that our FLWAP-tree algorithm outperforms the PLWAP-tree algorithm [1] significantly and consistently.

The organization of the rest of the paper is as follows. In Section 2, we present the formal framework and the theory of conditional searching. In Section 3, we describe the concept of first-occurrence and present our FLWAP-tree and the FLWAP-tree mining algorithm. Experimental results and concluding remarks of the paper are presented in Section 4 and Section 5, respectively.

---

\*The work was supported in part by NASA and the Arkansas Space Grant Consortium under Grant UALR11604.

## 2 Conditional Searching

Let  $\Sigma$  be the set of symbols. Each symbol in  $\Sigma$  represents a web page. A non-empty web access sequence  $s$  is a finite sequence of symbols from  $\Sigma$ ,  $s = s_1 \cdots s_m$ , such  $s_i \in \Sigma$  for all  $1 \leq i \leq m < \infty$  and  $s_i$  and  $s_j$  are not necessarily different for  $i \neq j$ . The *length* of the sequence  $s = s_1 \cdots s_m$  is  $m$ . The empty sequence denoted as  $\epsilon$  is a special web access sequence of length 0 and we have  $\epsilon \cdot s = s \cdot \epsilon = s$  for any sequence  $s$  where  $\cdot$  is the concatenation operator. A web access database  $D$  is a multi-set of web access sequences including the possible empty sequence. A *pattern* is also a web access sequence. A web access sequence  $s' = s'_1 \cdots s'_n$  is a *subsequence* of sequence  $s = s_1 \cdots s_m$ , denoted as  $s' \subseteq s$ , if and only if  $n \leq m$  and there exist  $i_1, \dots, i_n$  such that  $1 \leq i_1 < \dots < i_n \leq m$  and  $s'_j = s_{i_j}$  for all  $1 \leq j \leq n$ . The empty sequence  $\epsilon$  is a subsequence of any sequence. A web access sequence  $s$  in  $D$  is said to *support* pattern  $p$  if  $p$  is a subsequence of  $s$ . The *support* of pattern  $p$  in  $D$ , denoted as  $Sup_D(p)$ , is the number of web access sequences in  $D$  that support  $p$ . Given a threshold  $\xi$  in interval  $(0, 1]$ , a pattern  $p$  is *frequent* with respect to  $\xi$  and  $D$  if  $Sup_D(p) \geq \xi|D|$ , where  $|D|$  is the number of web sequences in  $D$ .  $\xi|D|$  is called the *absolute threshold* and denoted as  $\eta$ . The frequent web access pattern mining problem is to find all the frequent web access patterns in  $D$  with respect to  $\xi$ .

Given a web access sequence  $s$  and a symbol  $a$  from  $\Sigma$  such that  $a \subseteq s$ , the *a-prefix* of  $s$  is the prefix of  $s$  from the first symbol (the leftmost symbol) to the *first* occurrence of  $a$  inclusive. For example, the *a-prefix* of sequence  $bc\underline{a}bad$  is  $bc\underline{a}$  rather than  $bc\underline{a}ba$ , because the first occurrence of  $a$  is the  $a$  underlined. The *a-projection* of  $s$  is what is left after the *a-prefix* is deleted. In the example above, the *a-projection* of sequence  $bc\underline{a}bad$  is  $bad$ . Note that if  $a$  occurs only once as the last symbol in  $s$ , the *a-prefix* is  $s$  and the *a-projection* is the empty sequence  $\epsilon$ . For example, the *a-projection* of  $bcab\underline{a}$  is  $\epsilon$ .

Given the database  $D$  and a symbol  $a$  in  $\Sigma$ , the *a-projection database* of  $D$ , denoted as  $D_a$ , is the multi-set of *a-projections* of the web access sequences in  $D$  that support  $a$ . That is,

$$D_a = \{a\text{-projection of } s \mid a \subseteq s \wedge s \in D\}$$

Note that the same sequence may repeat in a projection database. For example, the *a-projection database* of  $D = \{cb\underline{a}ca, bcb\underline{a}ca, ccb\underline{a}ba\}$  is  $D_a = \{ca, ca, ba\}$ .

The following theorem relates the supports of patterns in the projection database and the original database.

**Theorem 1** Given a symbol  $a$  from  $\Sigma$  and database  $D$ , the support of pattern  $p$  (including empty pattern  $\epsilon$ ) in the *a-projection database*  $D_a$  of  $D$  is equal to the support of pattern  $a \cdot p$  in the original database  $D$ . That is,

$$Sup_D(a \cdot p) = Sup_{D_a}(p) \quad (1)$$

**Proof:** ( $\geq$ ) Let  $s$  be a sequence in  $D$  that supports  $a \cdot p$ , i.e.  $a \cdot p \subseteq s$ . Sequence  $s$  must be able to be split as  $s = \hat{a} \cdot \hat{p}$  such that  $\hat{a}$  is the minimal prefix of  $s$  such that  $a \subseteq \hat{a}$ , and  $p \subseteq \hat{p}$ .  $\hat{a}$  is the *a-prefix* of  $s$ . Thus,  $\hat{p}$  is the *a-projection* of  $s$  and is in  $D_a$ . It supports  $p$  as  $p \subseteq \hat{p}$ . Therefore, we have  $Sup_{D_a}(p) \geq Sup_D(a \cdot p)$ .

( $\leq$ ) On the other hand, let  $\tilde{p}$  be a sequence in  $D_a$  that supports  $p$ , i.e.  $p \subseteq \tilde{p}$ . Recall that  $D_a$  is the *a-projection database* of  $D$  and  $\tilde{p}$  is the *a-projection* of a sequence  $s'$  in  $D$ . Therefore, we have  $s' = \tilde{a} \cdot \tilde{p}$  where  $\tilde{a}$  is the *a-prefix* of  $s'$ . Since  $a \subseteq \tilde{a}$  and  $p \subseteq \tilde{p}$ , we have  $a \cdot p \subseteq \tilde{a} \cdot \tilde{p} = s'$ . Therefore, we have  $Sup_{D_a}(p) \leq Sup_D(a \cdot p)$ .  $\square$

When the same absolute threshold  $\eta = \xi|D|$  is used in the mining of both  $D$  and  $D_a$ , we can say that  $a \cdot p$  is frequent in  $D$  ( $Sup_D(a \cdot p) \geq \eta$ ) if and only if  $p$  is frequent in the *a-projection database*  $D_a$  ( $Sup_{D_a}(p) \geq \eta$ ).

Let  $p$  in (1) be the empty sequence  $\epsilon$  and we have  $Sup_D(a) = Sup_D(a \cdot \epsilon) = Sup_{D_a}(\epsilon)$ . Since empty sequence  $\epsilon$  is a subsequence of any sequence, the support of  $\epsilon$  in  $D_a$  is the total number of sequences in  $D_a$ , i.e.  $Sup_{D_a}(\epsilon) = |D_a|$ . Thus, we have

$$Sup_D(a) = |D_a| \quad (2)$$

Let  $F(D, \eta)$  be the set of frequent patterns in  $D$  with respect to  $\eta$ :

$$F(D, \eta) = \{p \mid Sup_D(p) \geq \eta\} \quad (3)$$

If  $|D| < \eta$ ,  $F(D, \eta)$  is empty as  $Sup_D(p) \leq |D| < \eta$  for any  $p$  including the empty pattern  $\epsilon$ . If  $|D| \geq \eta$ ,  $F(D, \eta)$  contains at least the empty pattern  $\epsilon$  as  $Sup_D(\epsilon) = |D| \geq \eta$ .

Let  $F'(a, D, \eta)$  be the set of non-empty frequent patterns in  $D$  that start with symbol  $a$ :

$$F'(a, D, \eta) = \{a \cdot q \mid Sup_D(a \cdot q) \geq \eta\} \quad (4)$$

Thus, we have

$$F(D, \eta) = \begin{cases} \emptyset & \text{if } |D| < \eta \\ \{\epsilon\} \cup \bigcup_{a \in \Sigma} F'(a, D, \eta) & \text{if } |D| \geq \eta \end{cases} \quad (5)$$

The  $F'(a, D, \eta)$  defined in (4) is related to  $F(D_a, \eta)$  as follows. Since  $Sup_D(a \cdot q) = Sup_{D_a}(q)$  according to (1) in Theorem 1, we have

$$\begin{aligned} F'(a, D, \eta) &= \{a \cdot q \mid Sup_D(a \cdot q) \geq \eta\} \\ &= \{a \cdot q \mid Sup_{D_a}(q) \geq \eta\} \\ &= \{a \cdot q \mid q \in F(D_a, \eta)\} \end{aligned}$$

Let  $aF(D_a, \eta)$  denote  $\{a \cdot q \mid q \in F(D_a, \eta)\}$ . Thus, we have

$$F'(a, D, \eta) = aF(D_a, \eta) \quad (6)$$

By substituting  $aF(D_a, \eta)$  for  $F'(a, D, \eta)$  in (5), we have the recursive equation for  $F(D, \eta)$ :

$$F(D, \eta) = \begin{cases} \emptyset & \text{if } |D| < \eta \\ \{\epsilon\} \cup \bigcup_{a \in \Sigma} aF(D_a, \eta) & \text{if } |D| \geq \eta \end{cases} \quad (7)$$

Equations (7) and (2) form the base for the pattern-growth mining algorithm we derive next. The direct translation of (7) to a recursive function to find  $F(D, \eta)$  is:

```

function Mine(database  $D$ , int  $\eta$ ) {
1: if  $|D| < \eta$  then
2:   return  $\emptyset$ ;
3: else
4:    $F \leftarrow \{\epsilon\}$ ;
   for each  $a \in \Sigma$  do
5:     if ( $Sup_D(a) \geq \eta$ ) then
6:        $F' \leftarrow Mine(D_a, \eta)$ ;
        $F \leftarrow F \cup aF'$ ;
     endif
   endfor
7: return  $F$ ;
8: endif
}

```

Here, we take advantage of (2) and test  $(Sup_D(a) \geq \eta)$  instead of  $(|D_a| \geq \eta)$  in line 4. If  $Sup_D(a) < \eta$ , i.e.  $|D_a| < \eta$ ,  $F(D_a, \eta)$  will be empty according to (7). There is no need to call the function recursively by invoking  $Mine(D_a, \eta)$ . The function above can be further simplified by removing lines 1-3 and 6, if  $(|D| \geq \eta)$  is a pre-condition of the function call. This is certainly the case when the function is called in line 5.

Since the recursive calls find the symbols of the frequent patterns one at a time, we can accumulate them as their prefixes as the recursive calls go deeper. In essence, the frequent patterns can grow as the recursive calls progress. Based on this, function  $Mine(D, \eta)$  above can be reformatted to the final pattern growth algorithm shown in Figure 1. A pre-condition of function call  $Pattern-Grow(q, D, \eta)$  is  $|D| \geq \eta$ . Parameter  $q$  is the frequent pattern prefix found so far ( $q$  is a frequent pattern itself and it must have been recorded because  $\epsilon \in F(D, \eta)$  according to (7)).

$Pattern-Grow(q, D, \eta)$  returns the set of frequent patterns that have  $q$  as the prefix and are longer than  $q$ . After  $(Sup_{D_a} \geq \eta)$  is tested to be true at line 1,  $q \cdot a$  is a frequent pattern (see line 2) because  $F(D_a, \eta)$  contains at least  $\epsilon$  according to (7). The further growth from  $q \cdot a$  is found by the recursive call with prefix  $q \cdot a$  on the  $a$ -projection database,  $D_a$ . The pre-condition  $|D_a| \geq \eta$

```

function Pattern-Grow(pattern  $q$ , database  $D$ , int  $\eta$ ) {
   $F \leftarrow \emptyset$ ;
  for each symbol  $a$  in  $\Sigma$  do
1:   if ( $Sup_D(a) \geq \eta$ ) then
2:      $F \leftarrow F \cup \{q \cdot a\}$ ;
3:     Construct the  $a$ -projection database  $D_a$  of  $D$ ;
4:      $F \leftarrow F \cup Pattern-Grow(q \cdot a, D_a, \eta)$ ;
   endif
  endfor
5: return  $F$ ;
}

```

Figure 1: Pattern-Growth Mining Algorithm

is satisfied because  $|D_a| = Sup_D(a) \geq \eta$  according to (2). If  $Sup_D(a) < \eta$ ,  $q \cdot a$  is not a frequent pattern and the growth of  $q$  stops there. If  $Sup_D(a) < \eta$  for all  $a$  in  $\Sigma$ , the function returns the empty set  $\emptyset$ .

The pre-condition of  $|D| \geq \eta$  for the original database  $D$  is always satisfied, because  $0 < \xi \leq 1$  and  $\eta = \xi|D| \leq |D|$ . Therefore, the non-empty frequent patterns in  $D$  can be found by calling  $Pattern-Grow(\epsilon, D, \eta)$ . The empty pattern  $\epsilon$  is also frequent by default, because  $|D| \geq \eta$ .

Let  $L$  be the minimum of the lengths of the  $\eta$  longest sequences in database  $D$ . The number of possible patterns that need to be considered in the search space is  $1 + |\Sigma| + \dots + |\Sigma|^L$ . The construction of projection databases (line 3) and finding the support of a symbol (line 1) is proportional to  $|D| \cdot L$ . Therefore, the complexity of frequent web access pattern mining algorithms is  $O(|D| \cdot |\Sigma|^{L+1})$ .

### 3 Mining with FLWAP-Tree

The aggregate tree [9] or base WAP-tree [2] is a compact data structure to represent a web access sequence database. A path from the root to a node of the tree is the common prefix of sequences in the database. Each node has a label for the name of the symbol and a count for the number of sequences that share this node. We use the same running example of the database as in [1, 2]:  $D = \{abac, abcac, babac, abacc\}$  with  $\Sigma = \{a, b, c\}$ . The threshold  $\xi$  is set to 0.75 and, thus,  $\eta = \xi|D|$  is 3. The count of any node in the base WAP-tree is equal to or greater than the sum of the counts of its children. The difference between them is the number of sequences in  $D$  that start with the root and end with the symbol of the node. Figure 2(a) shows the base WAP-tree of our example.

A node is a *first-occurrence* if none of its ancestors has the same label. For example, the base WAP-tree in Figure 2(a) has two first-occurrences of  $a$ :  $a:3$  in the

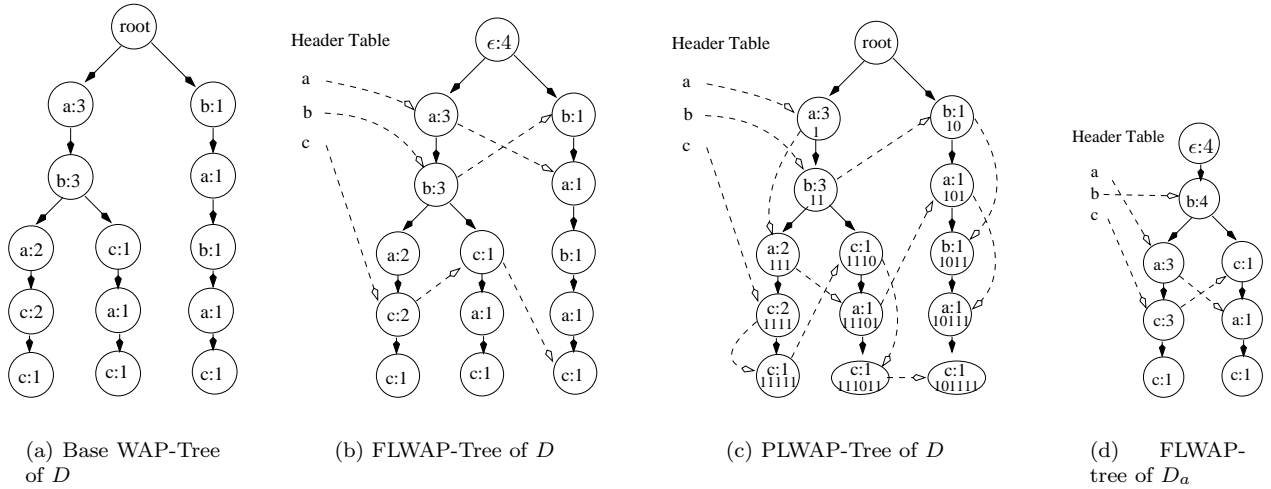


Figure 2: Base WAP-tree, FLWAP-tree and PLWAP-tree

left subtree and  $a:1$  in the right subtree. The count of a first-occurrence of a node with label  $a$  is the number of sequences in  $D$  that share the common  $a$ -prefix represented by the path from the root node to this first-occurrence. For example,  $b:3$  is a first-occurrence of  $b$  in Figure 2(a). The count 3 tells that there are three sequences in  $D$  that share the common  $b$ -prefix  $ab$  represented by the path from the root node to  $b:3$ . These sequences are  $abac$ ,  $abcac$  and  $abacc$ . Therefore, the sum of the counts of all the first-occurrences of a symbol  $a \in \Sigma$  is the number of sequences in  $D$  that contain at least one occurrence of  $a$ , i.e. the support of  $a$  in  $D$ ,  $Sup_D(a)$ . As we discussed in Section 2, the support of  $a$  in  $D$  also equals to the number of sequences (including possibly empty sequences  $\epsilon$ ) of the  $a$ -projection database of  $D$ ,  $|D_a|$ .

Consequently, the  $a$ -projection database of  $D$ ,  $D_a$ , can be represented by the subtrees rooted in the children of the first-occurrences of  $a$ , plus possible empty sequences. These subtrees are called  $a$ -projection trees, because they contain all the non-empty  $a$ -projections. We can build a base WAP-tree for the  $a$ -projection database  $D_a$  out of these  $a$ -projection trees.

The first-occurrences for each symbol can be linked to form the *First-Occurrence Linked WAP-tree* (FLWAP-tree). Figure 2(b) shows the FLWAP-tree of our example. Our mining algorithm using the FLWAP-tree is shown in Figure 3. It is an implementation of the abstract pattern-growth algorithm in Figure 1 using the FLWAP-tree. Our FLWAP-tree mining algorithm differs from the PLWAP-tree mining algorithm [1] in line 1 (how to find first-occurrences) and line 2 (build FLWAP-tree for the projection database). These differences are discussed next.

In order to find the first-occurrences of a symbol, it was suggested in [1] to attach a position code to each node and link all the nodes of the same label in a pre-order traversal of the tree. The linked tree is, thus, called the *Pre-Order Linked WAP-tree* (PLWAP-tree). Figure 2(c) shows the PLWAP-tree of our example. The mining algorithm in [1] follows the pre-order linkage and uses position codes to find first-occurrences. The PLWAP-tree links *all* the occurrences of the same label and uses position codes to determine whether a node along the linked list is a first-occurrence or not. It uses the original PLWAP for the entire mining and this forces it to go through the nodes not in the projection databases to find the first-occurrences.

```

FLWAP-mine(pattern  $q$ , FLWAP-tree  $T$ , int  $\eta$ ) {
   $F \leftarrow \emptyset$ ;
  for each  $a$  in  $\Sigma$  do
    1: Follow first-occurrence link of  $a$  to find
       its first-occurrences;
       if (the sum of the counts of first-occurrences  $\geq \eta$ )
         then
            $F \leftarrow F \cup \{q \cdot a\}$ ;
           Let  $Q$  be the set of the subtrees rooted
             at the children of the first-occurrences;
    2: Build FLWAP-tree  $T'$  for  $D_a$  out of  $Q$ ;
        $F \leftarrow F \cup FLWAP\text{-mine}(q \cdot a, T', \eta)$ ;
  endif
endfor
return  $F$ ;
}

```

Figure 3: Mining Algorithm Using FLWAP-tree

In fact, we do not need to link all the nodes with the same label in order to find the first-occurrences of a label. They can be found by traversing in pre-order a portion of the base WAP-tree only using the algorithm<sup>1</sup> shown in Figure 4.

```

procedure visit(tnode n) {
  if (stacks[n.label] is empty) then
    push n to stacks[n.label];
    link n into the first-occurrence list of n.label
  endif
  if (there is a label l such that stacks[l] is empty) then
    for each child c of n do
      visit(c);
    endfor
  endif
  if (stacks[n.label] is not-empty) then
    pop stacks[n.label];
  endif
}

```

Figure 4: Linking First-Occurrences

The second difference of our mining algorithm from that of [1] is in line 2 in Figure 3. Since all the non-empty sequences in the  $a$ -projection database  $D_a$  of database  $D$  are included in the subtrees rooted at the children of the first occurrences of a symbol  $a$ , the FLWAP-tree for  $D_a$  can be built from these subtrees. The detailed algorithm of building the FLWAP-tree for the projection database from these subtrees can be found in [10]. Figure 2(d) shows the FLWAP-tree of  $a$ -projection database  $D_a$  of our example.

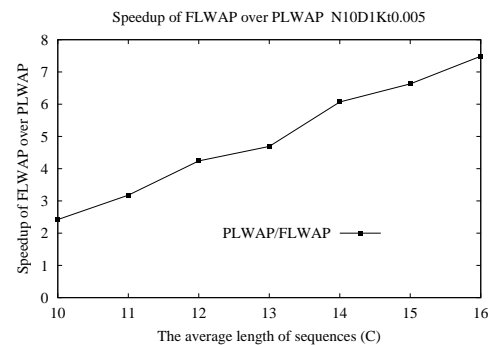
Building new trees for projection databases will use more memory. However, the sizes of the projection databases decrease as the recursion progresses. Due to the exponential complexity of pattern growth algorithms (see Section 2), our major concern is the time of the mining. In essence, our FLWAP-tree mining is a trade-off of memory for high performance. One possible way to reduce the memory use is to use the quasi-FLWAP-trees for projection databases. A quasi-FLWAP-tree can be obtained by making the children of the first-occurrences of a symbol the children of the new root node and applying function `visit()` in Figure 4 to form the new first-occurrence links. However, as the subtrees are not merged, the first-occurrence links will be longer than the FLWAP-tree and this could slow

<sup>1</sup>The algorithm uses an array of stacks `stacks[]`, one for each label. Each stack stores at most one node pointer and is initially empty. The function `visit(tnode n)` basically traverses in pre-order a portion of the base WAP-tree pointed to by  $n$ . If the stack of the label of the node visited is empty before we push the node pointer to it, we have found a first-occurrence of that label.

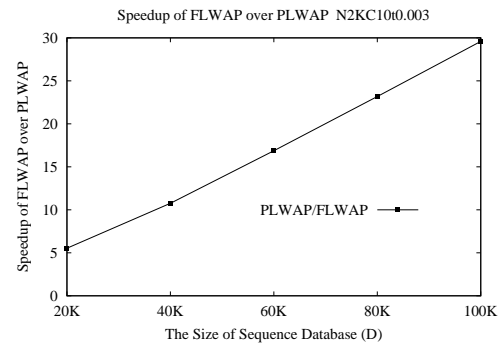
down the mining. We plan to look at this aspect in a future study. In this paper we study how performance can be improved by using the FLWAP-tree.

## 4 Experimental Results

In order to compare the performance of our FLWAP-tree mining with that of PLWAP-tree mining, we use the IBM data generator [8] to generate two sets of datasets of different characters. The first set, named N10D1KS4, has  $N=10$  symbols,  $D=1000$  sequences in each database and the average length  $C$  of the sequences in a database varying from 10 to 16. This set has large  $C/N$  ratios ranging from



(a) N10D1KS4 with  $t=0.005$



(b) N2KC10S5 with  $t=0.003$

Figure 5: Speedup of FLWAP over PLWAP

1.0 to 1.6. The second set, named N2KC10S4, has  $N=2000$  symbols, the average length  $C=10$  and the size of database varying from 20,000 to 100,000. This set has large number of symbols, large sizes of the databases and a very small  $C/N$  ratio (0.005). This set of datasets is the same as used in the experimental evaluation for the PLWAP-tree

mining in [1]. We run both our FLWAP-tree mining and the PLWAP-tree mining on these two sets of datasets. The PLWAP source code [6] is obtained from <http://sol.cs.uwindsor.ca/~cezeife/codes.html>. All the tests are run on a Pentium III processor of 997 MHz with 256KB cache and 512MB RAM.

Figure 5 shows the speedup of our FLWAP code over the PLWAP code calculated as follows:

$$Sp = \frac{T_{PLWAP}}{T_{FLWAP}} \quad (8)$$

where  $T_{PLWAP}$  and  $T_{FLWAP}$  are the execution times<sup>2</sup> of the PLWAP code [6] and our FLWAP code on the same dataset, respectively. Figure 5(a) is the speedup of FLWAP over PLWAP when mining dataset N10D1KS4 with threshold 0.005. It shows that our FLWAP-tree mining outperforms the PLWAP-tree mining consistently with  $C$  varying from 10 to 16. The speedup increases linearly as  $C$  increases. When  $C=16$  with  $C/N=1.6$ , the FLWAP-tree mining is 7.48 times as fast as the PLWAP-tree mining. Figure 5(b) shows the speedup of FLWAP over PLWAP when mining dataset N2KC10S4 with threshold 0.003. The speedup also increases linearly as the size of the databases increases. The speedup of FLWAP over PLWAP reaches 29.58 for  $D=100,000$ .

The reasons why our FLWAP-tree mining is faster than the PLWAP-tree mining are two-fold: (1) The FLWAP-tree mining builds new FLWAP trees for the projection databases whose sizes are increasingly smaller as the recursion progresses. Our FLWAP-tree mining has an efficient algorithm to build the new FLWAP-tree for the projection database. (2) The FLWAP-tree mining does not go through unnecessary nodes in the original WAP tree to find the first-occurrences. The first-occurrences of all the labels in our FLWAP-tree are found by following the first-occurrence links which are established by traversing only the upper part of the tree of the increasingly smaller projection databases.

## 5 Conclusion

We have presented a formal framework and the theory of conditional searching which enables us to reason about and prove pattern-growth mining algorithms rigorously.

We have defined the concept of first-occurrence and presented a web access pattern mining algorithm based on it using a novel first-occurrence linked

WAP-tree (FLWAP-tree). The performance evaluation shows that our FLWAP-tree mining outperforms the PLWAP-tree mining consistently and significantly.

## References

- [1] C.I. Ezeife and Yi Lu. Mining web log sequential patterns with position coded pre-order linked wap-tree. *International Journal of Data Mining and Knowledge Discovery*, 10:5–38, 2005.
- [2] Jian Pei, Jiawei Han, Behzad Mortazavi-asl, and Hua Zhu. Mining access patterns efficiently from web logs. In *Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'00)*, pages 396–407. Lecture Notes in Computer Science, Vol. 1805, 2000.
- [3] Q. Yang and H. H. Zhang. Web-log mining for predictive web caching. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):1050–1053, 2003.
- [4] C.I. Ezeife and Min Chen. Mining web sequential patterns incrementally with revised PLWAP tree. In *Proceedings of the 5th International Conference on Web-Age Information Management(WAIM 2004)*, pages pp. 539–548. Lecture Notes of Computer Science 3129, Springer Verlag, 2004.
- [5] J. Wang and J. Han. BIDE: Efficient mining of frequent closed sequences. In *Proceedings of the 20th International Conference on Data Engineering (ICDE'04)*, pages 79–90, 2004.
- [6] C.I. Ezeife and Yi Lu. PLWAP sequential mining: open source code. In *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, pages 26 – 35, 2005.
- [7] Jian-Chin Ou, Chang-Huang Lee, and Ming-Syan Chen. Web log mining with adaptive support thresholds. In *Proceedings of 2005 International World Wide Web Conference*, pages 1188–1189, 2005.
- [8] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the International Conference on Extending Database Technology*, pages 3–17, 1996.
- [9] Myra Spiliopoulou and Lukas C. Faulstich. WUM: A tool for web utilization analysis. In *Proceedings of EDBT Workshop Web DB'98*. Springer Verlag, LNCS 1590, 1998.
- [10] Peiyi Tang, Markus P. Turkia, and Kyle A. Galivan. Mining web access patterns with first-occurrence linked WAP-tree. Technical Report [titus.compsci.ualr.edu/~ptang/papers/flwap-rpt.pdf](http://titus.compsci.ualr.edu/~ptang/papers/flwap-rpt.pdf), Department of Computer Science, University of Arkansas at Little Rock, 2006.

<sup>2</sup>The execution times of both FLWAP and PLWAP mining are plotted in [10].