Mining Access Patterns Efficiently from Web Logs *

Jian Pei, Jiawei Han, Behzad Mortazavi-asl, and Hua Zhu

School of Computing Science, Simon Fraser University, Canada
{peijian, han, mortazav, hzhua}@cs.sfu.ca

Abstract. With the explosive growth of data available on the World
Wide Web, discovery and analysis of useful information from the World
Wide Web becomes a practical necessity. Web access pattern, which is the
sequence of accesses pursued by users frequently, is a kind of interesting
and useful knowledge in practice.
In this paper, we study the problem of mining access patterns from Web
logs efficiently. A novel data structure, called **Web access pattern
tree**, or **WAP-tree** in short, is developed for efficient mining of access
patterns from pieces of logs. The Web access pattern tree stores highly
compressed, critical information for access pattern mining and facilitates
the development of novel algorithms for mining access patterns in large
set of log pieces. Our algorithm can find access patterns from Web logs
quite efficiently. The experimental and performance studies show that
our method is in general an order of magnitude faster than conventional
methods.

1   Introduction

With the explosive growth of data available on the World Wide Web, discovery
and analysis of useful information from the World Wide Web becomes a practical
necessity. *Web mining* is the application of data mining technologies to huge
Web data repositories. Basically, there are two domains that pertain to Web
mining: Web content mining and Web usage mining. The former is the process
of extracting knowledge from the content of Web sites, whereas the latter, also
known as *Web log mining*, is the process of extracting interesting patterns in
Web access logs.

Web servers register a Web log entry for every single access they get, in which
important pieces of information about accessing are recorded, including the URL
requested, the IP address from which the request originated, and a timestamp.
A fragment of log file is shown as follows.

```
pm21s15.intergate.bc.ca - - [06/Oct/1999:00:00:09 -0700] "GET / HTTP/1.1" 200 5258 "http://www.sfu.ca/academic_programs.htm"
"Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)"

pm21s15.intergate.bc.ca - - [06/Oct/1999:00:00:11 -0700] "GET /images/bullets/bsqs.gif HTTP/1.1" 200 489 "http://www.cs.sfu.ca/"
"Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)"
```

There are many efforts towards mining various patterns from Web logs, e.g. [4, 11, 15]. Web access patterns mined from Web logs are interesting and useful knowledge in practice. Examples of applications of such knowledge include improving designs of web sites, analyzing system performance as well as network communications, understanding user reaction and motivation, and building adaptive Web sites [5, 10, 13, 14].

Essentially, a Web access pattern is a sequential pattern in a large set of pieces of Web logs, which is pursued frequently by users. Some research efforts try to employ techniques of sequential pattern mining [2], which is mostly based on association rule mining [1], for discovering Web access patterns from Web logs.

Sequential pattern mining, which discovers frequent patterns in a sequence database, was first introduced by Agrawal and Srikant [2] as follows: *given a sequence database where each sequence is a list of transactions ordered by transaction time and each transaction consists of a set of items, find all sequential patterns with a user-specified minimum support, where the support is the number of data sequences that contain the pattern.*

Since its introduction, there have been many studies on efficient mining techniques and extensions of sequential pattern mining method to mining other time-related frequent patterns [2, 12, 8, 7, 3, 9, 6].

Srikant and Agrawal [12] generalized their definition of sequential patterns in [2] to include time constraints, sliding time window, and user-defined taxonomy and developed a generalized sequential pattern mining algorithm, GSP, which outperforms their AprioriAll algorithm [2]. GSP mines sequential patterns by scanning the sequence database multiple times. In the first scan, it finds all frequent 1-items and forms a set of 1-element frequent sequences. In the following scans, it generates (step-wise longer) *candidate sequences* from the set of frequent sequences and check their supports. GSP is efficient when the sequences are not long as well as the transactions are not large. However, when the length of sequences increase and/or when the transactions are large, the number of candidate sequences generated may grow exponentially, and GSP will encounter difficulties.

All of the above studies on time-related (sequential or periodic) frequent pattern mining adopt an Apriori like paradigm, which promotes a generate-and-test method: first generate a set of candidate patterns and then test whether each candidate may have sufficient support in the database (i.e., passing the minimum support threshold test). The Apriori heuristic is on how to generate a *reduced set* of candidates at each iteration.

However, as these algorithms are level-wise, Apriori -like in nature, they encounter the same difficulty when the length of the pattern grows long, which is exactly the case in Web access pattern mining. In Web log mining, the length of Web log pieces can be pretty long, while the number of such pieces can be quite huge in practice.

In this paper, we investigate the issues related to efficiently mining Web access from large set of pieces of Web log. The main contributions are as follows. First,

a concise, highly compressed WAP-tree structure is designed and implemented which handles the sequences elegantly. Second, an efficient mining algorithm, WAP-mine , is developed for mining the complete (but nonredundant) Web access patterns from large set of pieces of Web log. Third, a performance study has been conducted which demonstrates that the WAP-mine algorithm is an order of magnitude faster than its Apriori -based counterpart for mining Web access patterns.

The remaining of the paper is organized as follows. The problem is defined in Section 2, while the general idea of our novel method is presented in Section 3. Section 4 and 5 focus on construction WAP-tree and mining the tree, respectively. We show the experimental results and conclude the paper in Section 6.

2   Problem Statement

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

Let E be a set of **events**. A **Web log piece** or **(Web) access sequence** $S = e_1 e_2 \cdots e_n$ ($e_i \in E$) for ($1 \leq i \leq n$) is a sequence of events, while n is called the **length** of the access sequence. An access sequence with length n is also called an n-**sequence**. Please note that it is not necessary that $e_i \neq e_j$ for ($i \neq j$) in an access sequence S. Repetition is allowed. For example, aab and ab are two different access sequences, in which a and b are two events.

Access sequence $S' = e'_1 e'_2 \cdots e'_l$ is called a **subsequence** of access sequence $S = e_1 e_2 \cdots e_n$, and S a **super-sequence** of S', denoted as $S' \subseteq S$, if and only if there exist $1 \leq i_1 < i_2 < \cdots < i_l \leq n$, such that $e'_j = e_{i_j}$ for ($1 \leq j \leq l$). Access sequence S' is a **proper subsequence** of sequence S, denoted as $S' \subset S$, if and only if S' is a subsequence of S and $S' \neq S$.

In access sequence $S = e_1 e_2 \cdots e_k e_{k+1} \cdots e_n$, if subsequence $S_{suffix} = e_{k+1} \cdots e_n$ is a super sequence of pattern $P = e'_1 e'_2 \cdots e'_l$, and $e_{k+1} = e'_1$, the subsequence of S, $S_{prefix} = e_1 e_2 \cdots e_k$, is called the **prefix** of S with respect to pattern P.

Given a set of access sequence $\mathcal{WAS} = \{S_1, S_2, \ldots, S_m\}$, called **Web access sequence database**, in which S_i ($1 \leq i \leq m$) are access sequences. The **support** of access sequence S in $\mathcal{WAS}$ is defined as $sup_{\mathcal{WAS}}(S) = \frac{|\{S_i | S \subseteq S_i\}|}{m}$. $sup_{\mathcal{WAS}}(S)$ is also denoted as $sup(S)$ if $\mathcal{WAS}$ is clear from the context. A sequence S is said a ξ-**pattern** or simply **(Web) access pattern** of $\mathcal{WAS}$, if $sup_{\mathcal{WAS}}(S) \geq \xi$. Please note that the access sequences in a Web access sequence database need not be of the same length. Although events can be repeated in an access sequence or pattern, any pattern can get support at most once from one access sequence.

Problem Statement. The problem of **Web access pattern mining** is: *given Web access sequence database* $\mathcal{WAS}$ *and a support threshold* ξ*, mine the complete set of* ξ*-patterns of* $\mathcal{WAS}$.

Example 1. Let $\{a, b, c, d, e, f\}$ 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.

$$\langle 100, a\rangle\langle 100, b\rangle\langle 200, a\rangle\langle 300, b\rangle\langle 200, b\rangle\langle 400, a\rangle\langle 100, a\rangle\langle 400, b\rangle\langle 300, a\rangle\langle 100, c\rangle$$

$$\langle 200, c\rangle\langle 400, a\rangle\langle 200, a\rangle\langle 300, b\rangle\langle 200, c\rangle\langle 400, c\rangle\langle 400, c\rangle\langle 300, a\rangle\langle 300, c\rangle$$

A preprocessing which divides the log file into access sequences of individual users is applied to the log file, while the resulting access sequence database, denoted as $\mathcal{WAS}$, is shown in the first two columns in Table 1.

There are totally 4 access sequences in the database. They are not with same length. The first access sequence, *abdac*, is a 5-sequence, while *ab* is a subsequence of it. In access sequence of user 200, both *e* and *eaebc* are prefixes with respect to *ac*. *fc* is a 50%-pattern because it gets supports from access sequence of user 300 and 400. Please note that even *fc* appears twice in the access sequence of user 400, *afbacfc*, but the sequence contributes only one to the count of *fc*.

User ID	Web Access Sequence	Frequent subsequence
100	*abdac*	*abac*
200	*eaebcac*	*abcac*
300	*babfaec*	*babac*
400	*afbacfc*	*abacc*

Table 1. A database of Web access sequences.

3   WAP-mine : Mining Access Patterns Efficiently from Web Logs

Access patterns can be mined using sequential pattern mining techniques. Almost all previously proposed methods for sequential pattern mining are based on a sequential pattern version of *Apriori heuristic* [1], stated as follows.

Property 1. (**Sequential Pattern Apriori**) Let $\mathcal{SEQ}$ be a sequence database, if a sequence G is not a ξ-pattern of $\mathcal{SEQ}$, any super-sequence of G cannot be a ξ-pattern of $\mathcal{SEQ}$.

For example, "*f*" is not a 75%-pattern of $\mathcal{WAS}$ in Example 1, thus any access sequence containing "*f*", cannot be a 75%-pattern.

The sequential pattern Apriori property may substantially reduce the size of candidate sets. However, because of the combinatorial nature of the sequential

pattern mining, it may still generate a huge set of candidate patterns, especially when the sequential pattern is long, which is exactly the case of Web access pattern mining.

This motivates us to study alternative structures and methods for Web access pattern mining. The key consideration is how to facilitate the tedious support counting and candidate generating operations in the mining procedure.

Our novel approach for mining Web access patterns is called WAP-mine . It is based on the following heuristic, which follows Property 1.

Property 2. (**Suffix heuristic**) If e is a frequent event in the set of prefixes of sequences in $\mathcal{WAS}$, w.r.t. pattern P, sequence eP is an access pattern of $\mathcal{WAS}$.

For example, b is a frequent event within the set of prefixes w.r.t. ac in Example 1, so we can claim that bac is an access pattern.

Basically, the general idea of our method can be summarized as follows.

- A nice data structure, WAP-tree , is devised to register access sequences and corresponding counts compactly, so that the tedious support counting can be avoided. It also maintains linkages for traversing prefixes with respect to the same suffix pattern efficiently. A WAP-tree registers all and only all information needed by the rest of mining. Once such a data structure is built, all the remaining mining processing is based on the WAP-tree . The original access sequence database is not needed any more. Because the size of WAP-tree is usually much smaller than that of the original access sequence database, as shown later, the mining becomes easier. As shown in Section 4, the construction of WAP-tree is quite efficient by simply scanning the access sequence database twice.
- An efficient recursive algorithm is proposed to enumerate access patterns from WAP-tree . No candidate generation is required in the mining procedure, and only the patterns with enough support will be under consideration. The philosophy of this mining algorithm is *conditional search*. Instead of searching patterns level-wise as Apriori , conditional search narrows the search space by looking for patterns with the same suffix, and count frequent events in the set of prefixes with respect to condition as suffix. Conditional search is a partition-based divide-and-conquer method instead of bottom-up generation of combinations. It avoids generating large candidate sets.

The essential structure of the WAP-mine algorithm is as follows. The algorithm scans the access sequence database twice. In the first pass, it determines the set of frequent events. An event is called a **frequent event** if and only if it appears in at least ($\xi \cdot |\mathcal{WAS}|$) access sequences of $\mathcal{WAS}$, in which $|\mathcal{WAS}|$ and ξ denotes the number of access sequences in $\mathcal{WAS}$ and the support threshold, respectively. In the next scan, WAP-mine builds a tree data structure, called WAP-tree , using frequent events, to register all count information for further mining. Then, WAP-mine recursively mine the WAP-tree using *conditional search* to find all Web access patterns. An overview of the algorithm is given in Algorithm 1.

Algorithm 1 (WAP-mine : mining access patterns in Web access sequence database)
Input: access sequence database $\mathcal{WAS}$ and support threshold ξ $(0 < \xi \leq 1)$.
Output: the complete set of ξ-patterns in $\mathcal{WAS}$.
Method:

1. Scan $\mathcal{WAS}$ once, find all frequent events.
2. Scan $\mathcal{WAS}$ again, construct a WAP-tree over the set of frequent events for using Algorithm 2, presented in Section 4;
3. Recursively mine the WAP-tree using *conditional search*, which will be presented in Section 5.

There are two key techniques in our method, constructing WAP-tree and mining access patterns from WAP-tree . They are explored in detail in the following two sections. Section 4 focuses on the concept and the construction of WAP-tree , while Section 5 investigates the mining of access patterns from WAP-tree .

4   Construction of WAP-tree

The following observations may help us design a highly condensed Web access pattern tree.

1. Of all the 1-sequences, only the frequent ones will be useful in the construction of frequent k-sequences for any $k > 1$. Thus, if an event e is not in the set of frequent 1-sequences, there is no need to include e in the construction of a Web access pattern tree.
2. If two access sequences share a common prefix P, the prefix P can be shared in the WAP-tree . Such sharing can bring some advantages. It saves some space for storing access sequences and facilitates the support counting of any subsequence of the prefix P.

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

1. Each node in a WAP-tree registers two pieces of information: label and count, denoted as *label* : *count*. The root of the tree is a special virtual node with an empty label and count 0. Every other node is labeled by an event in the event set E, and is associated with a *count* which registers the number of occurrences of the corresponding prefix ended with that event in the Web access sequence database.
2. The WAP-tree is constructed as follows: for each access sequence in the database, filter out any nonfrequent events, and then insert the resulting *frequent subsequence* into WAP-tree . The insertion of frequent subsequence is started from the root of WAP-tree . Considering the first event, denoted as e, increment the count of child node with label e by 1 if there exists one; otherwise create a child labeled by e and set the count to 1. Then, recursively insert the rest of the frequent subsequence to the subtree rooted at that child labeled e.

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

Example 2. Let's consider the access sequence database in Example 1. Suppose the support threshold is set to 75%, i.e. it is required to find all Web access patterns supported by at least three access sequences in the database.

One scan of the database derives the set of frequent 1-events: $\{a, b, c\}$. For convenience, the frequent subsequences are listed in the rightmost column of Table 1.
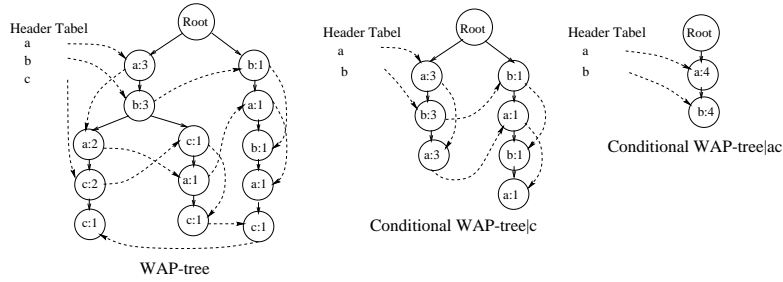
Fig. 1. The WAP-tree and conditional WAP-tree for frequent subsequences in Table 1.

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

The algorithm for constructing a WAP-tree for Web access sequences is given in Algorithm 2.

Algorithm 2 (WAP-tree Construction for Web access sequences)
Input: A Web access sequence database $\mathcal{WAS}$ and the set of frequent events FE (which is obtained by scanning $\mathcal{WAS}$ once).
Output: an WAP-tree $\mathcal{T}$.
Method:

1. Create a root node for $\mathcal{T}$;
2. For each access sequence S in the access sequence database $\mathcal{WAS}$ do
   (a) Extract frequent subsequence S' from S by removing all events appearing in S but not in FE. Let $S' = s_1 s_2 \cdots s_n$, where s_i $(1 \leq i \leq n)$ are events in S'. Let *current_node* point to the root of $\mathcal{T}$.
   (b) For $i = 1$ to n do, if *current_node* has a child labeled s_i, increase the count of s_i by 1 and make *current_node* point to s_i, else create a new child node $(s_i : 1)$, make *current_node* point to the new node, and insert it into the s_i-queue.
3. Return($\mathcal{T}$);

Analysis: The WAP-tree registers all access sequence counts. As will be shown in later sections, the mining process for all Web access patterns needs to work on the WAP-tree only, instead of on the original database any more. Therefore, WAP-mine needs to scan the access sequence database only twice. It is easy to show that the height of the WAP-tree is one plus the maximum length of the frequent subsequences in the database. The width of the WAP-tree , i.e. the number of leaves of the tree, is bounded by the number of access sequences in the database. Therefore, WAP-tree may not generate explosive number of nodes. Access sequences with same prefix will share some upper part of path from root. Statistically, considering the factor of prefix sharing, the size of WAP-tree is much smaller than the size of access sequence database.

From Algorithm 2, the construction of WAP-tree , one can observe an important property of WAP-tree stated as follows.

Lemma 1. *For any access sequence in an access sequence database $\mathcal{WAS}$, there exists a unique path in the* WAP-tree *starting from the root such that all labels of nodes in the path in order is exactly the same as the events in the sequence.*

This lemma ensures that the number of distinct leaf nodes as well as paths in an WAP-tree cannot be more than the number of distinct frequent subsequences in the access sequence database, and the height of the WAP-tree is bounded by one (for the root) plus the maximal number of instances of frequent 1-events in an access sequence.

It is easy to show that a WAP-tree can be partitioned and structured in the form similar to B+-tree, and can be implemented even in pure SQL. Therefore, WAP-tree as well as mining using WAP-tree are highly scalable.

5 Mining Web Access Patterns from WAP-tree

The WAP-tree structure constructed by Algorithm 2 provides some interesting properties which facilitate mining Web access patterns.

Property 3. (**Node-link property**) For any frequent event e_i, all the frequent subsequences contain e_i can be visited by following the e_i-queue, starting from the record for e_i in the header table of WAP-tree .

The property facilitates the access of all the pattern information related to frequent event e_i by following the all branches in WAP-tree linked by e_i-queue only once. For any node labeled e_i in an WAP-tree , all nodes in the path from root of the tree to this node (excluded) form a **prefix sequence** of e_i, The count of this node labeled e_i is called the **count** of the prefix sequence.

Please note that a path from root may have more than one node labeled e_i, thus a prefix sequence of e_i may contain another prefix sequence of e_i. For example, sequence aba is a prefix sequence of "b" in $abab$, it contains another prefix sequence of "b", a. when counting ab in sequence $abab$, we must make sure no double counting, i.e. $abab$ contributes only 1 to the count of ab. It is achieved by the concept of *unsubsumed count* as follows.

Let G and H be two prefix sequences of e_i, and G is also formed by the sub-path from root of that H is formed by, H is called a **super-prefix sequence** of G, and G is a **sub-prefix sequence** of H. For instance, aba is a super-prefix sequence of a.

For a prefix sequence of e_i without any super-prefix sequences, we define the **unsubsumed count** of that sequence as the count of it. For a prefix sequence of e_i with some super-prefix sequences, the *unsubsumed count* of it is the count of that sequence minus unsubsumed counts of all its super-prefix sequences. For example, let $S = (a : 6) \rightarrow (b : 5) \rightarrow (a : 2) \rightarrow (b : 2)$ be one path from root, the unsubsumed count of the first a, a prefix sequence of b, in the path should be 3 instead of 5, since two of the totally five counts in the first b node devotes to the super-prefix sequence aba of a.

Property 4. (**Prefix sequence unsubsumed count property**) The count of a sequence G ended with e_i is the sum of unsubsumed counts of all prefix sequences of e_i which is a super-sequence of G.

Based on the above two properties, we can apply **conditional search** to mine all Web access patterns using WAP-tree . What "conditional search" means, instead of searching *all* Web access patterns at a time, it turns to search Web access patterns with same **suffix**. Suffix is used as the condition to narrow the search space. As the suffix becomes longer, the remaining search space becomes smaller potentially.

The conditional search paradigm has some advantages against Apriori -like ones. The *node-link property* of WAP-tree ensures that, for any frequent event e_i, all sequences with suffix e_i can be visited efficiently using the e_i-queue of the tree. On the other hand, the *prefix sequence unsubsumed count property* makes sure that to count all frequent events in the set of sequences with same suffix, only caring the unsubsumed count is sufficient. That simplifies the counting operations. These two properties of WAP-tree make the conditional search efficient.

The basic structure of mining all Web access patterns in WAP-tree is as follows. If the WAP-tree has only one branch, all (ordered) combinations events in the branch are all the Web access patterns in the tree. So what needs to be done is just to return the complete set of such combinations. Otherwise, for each frequent event e_i in the WAP-tree , by following the e_i-queue started from

header table, an e_i-**conditional access sequence base** is constructed, denoted as $\mathcal{PS} \mid e_i$, which contains all and only all prefix sequences of e_i. Each prefix sequence in $\mathcal{PS} \mid e_i$ carries its count from the WAP-tree . For each prefix sequence of e_i with count c, when it is inserted into $\mathcal{PS} \mid e_i$, all of its sub-prefix sequences of e_i are inserted into $\mathcal{PS} \mid e_i$ with count $-c$. It is easy to show that by accumulating counts of prefix sequences, a prefix sequence in $\mathcal{PS} \mid e_i$ holds its unsubsumed count. Then, the complete set of Web access patterns which are prefix sequence of e_i can be mined by concatenating e_i to all Web access patterns returned from mining the conditional WAP-tree , and e_i itself.

The algorithm is given as follows.

Algorithm 3 (Mining all Web access patterns in a WAP-tree)
Input: a WAP-tree $\mathcal{T}$ and support threshold ξ.
Output: the complete set of ξ-patterns.
Method:

1. if the WAP-tree $\mathcal{T}$ has only one branch, return all the unique combinations of nodes in that branch.
2. initialize Web access pattern set $\mathcal{WAP} = \emptyset$. Every event in WAP-tree $\mathcal{T}$ itself is a Web access pattern, insert them into $\mathcal{WAP}$.
3. for each event e_i in WAP-tree $\mathcal{T}$,
   (a) construct a conditional sequence base of e_i, i.e. $\mathcal{PS} \mid e_i$, by following the e_i-queue, count conditional frequent events at the same time.
   (b) if the the set of conditional frequent events is not empty, build a conditional WAP-tree for e_i over $\mathcal{PS} \mid e_i$ using algorithm 2. Recursively mine the conditional WAP-tree
   (c) for each Web access pattern returned from mining the conditional WAP-tree , concatenate e_i to it and insert it into $\mathcal{WAP}$
4. return $\mathcal{WAP}$.

Example 3. Let us mine the Web access patterns in the WAP-tree in Figure 1. Suppose the support threshold is set to 75%. We start the conditional search from frequent event c. The conditional sequence base of c is listed as follows.

$$aba : 2, ab : 1, abca : 1, ab : -1, baba : 1, abac : 1, aba : -1$$

To be qualified as a conditional frequent event, one event must have count 3. Therefore, the conditional frequent events are $a(4)$ and $b(4)$. Then, a conditional WAP-tree , WAP-tree $\mid c$, is built, as also shown in Figure 1.

Recursively, the conditional sequence base of ca is built. It is $ab : 3, b : 1, ab : 1, b : -1$. The WAP-tree $\mid a$ is built, also shown in Figure 1. There is only one branch in the conditional tree, so all combinations are generated. Thus, the Web access patterns with suffix ac are $aac, bac, abac, ac$.

Then, we can construct the conditional sequence base for b in WAP-tree $\mid c$, and mine the conditional WAP-tree . The frequent patterns abc, bc can be found.

At this point, the conditional search of c is finished. We can use other frequent events in turn, to find all the other Web access patterns.

Following the properties presented ahead and considering the enumerating of access patterns in Algorithm 3, the correctness of WAP-mine can be shown.

Theorem 1. WAP-mine *returns the complete set of access patterns without redundancy.*

As can be seen in the example, and shown in our experiments, mining Web access patterns using WAP-tree has significant advantages. First, the WAP-tree is an effective data structure. It registers all count information for pattern mining, and frees the mining process from counting candidates by pattern matching. Secondly, the conditional search strategies narrow the search space efficiently, and make best uses of WAP-tree structure. It avoids the overwhelming problems of generating explosive candidates in Apriori -like algorithms.

6 Performance Evaluation and Conclusions

Experiments are pursued to compare the efficiency of WAP-mine and GSP, the algorithm proposed in [12]. The dataset for experiment is generated based on the principle introduced in [2]. All experiments are conducted on a 450-MHz Pentium PC machine with 64 megabytes main memory, running Microsoft Windows/NT. All the programs are written in Microsoft/Visual C++ 6.0.
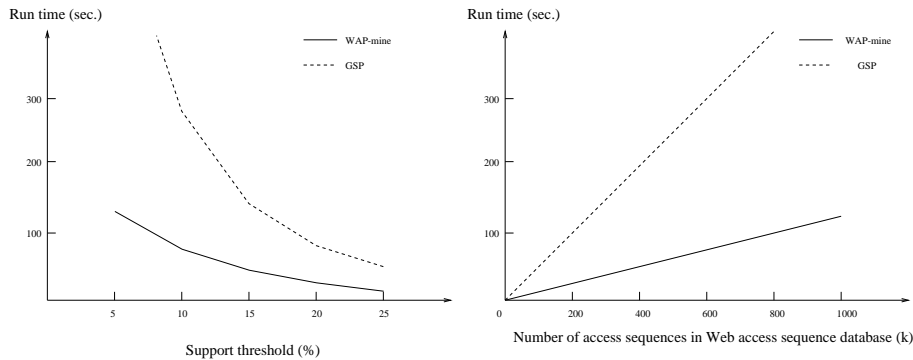
Fig. 2. Experimental results.

The experimental results are shown in Figure 2. We compare the scalabilities of our WAP-mine and GSP, with threshold as well as the number of access sequences in the database. The experimental result shows that WAP-mine outperforms GSP in quite significant margin, and WAP-mine has better scalability than GSP. Both WAP-mine and GSP show linear scalability with the number of access sequences in the database. However, WAP-mine outperforms GSP.

In conclusion, WAP-tree is an effective structure facilitating Web access pattern mining, and WAP-mine outperforms GSP based solution in a wide margin.

The success of WAP-tree and WAP-mine can be credited to the compact structure of WAP-tree and the novel *conditional search* strategies.

We believe that, with certain extensions, the methodology of WAP-tree and WAP-mine can be applied to attack many data mining tasks efficiently such as sequential pattern mining and episode mining.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 487–499, Santiago, Chile, September 1994.
2. R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering*, pages 3–14, Taipei, Taiwan, March 1995.
3. C. Bettini, X. Sean Wang, and S. Jajodia. Mining temporal relationships with multiple granularities in time sequences. *Data Engineering Bulletin*, 21:32–38, 1998.
4. R. Cooley, B. Mobasher, and J. Srivastava. Data preparation for mining World Wide Web browsing patterns. In *Journal of Knowledge & Information Systems*, Vol.1, No.1, 1999.
5. J. Graham-Cumming. Hits and misses: A year watching the Web. In *Proc. 6th Int'l World Wide Web Conf.*, Santa Clara, California, April 1997.
6. J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Proc. 1999 Int. Conf. Data Engineering (ICDE'99)*, pages 106–115, Sydney, Australia, April 1999.
7. H. Lu, J. Han, and L. Feng. Stock movement and n-dimensional inter-transaction association rules. In *Proc. 1998 SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'98)*, pages 12:1–12:7, Seattle, Washington, June 1998.
8. H. Mannila, H Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.
9. B. Özden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proc. 1998 Int. Conf. Data Engineering (ICDE'98)*, pages 412–421, Orlando, FL, Feb. 1998.
10. M. Perkowitz and O. Etzioni. Adaptive sites: Automatically learning from user access patterns. In *Proc. 6th Int'l World Wide Web Conf.*, Santa Clara, California, April 1997.
11. M. Spiliopoulou and L. Faulstich. WUM: A tool for Web utilization analysis. In Proc. 6th Int'l Conf. on Extending Database Technology (EDBT'98), Valencia, Spain, March 1998.
12. R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 1–12, Montreal, Canada, June 1996.
13. T. Sullivan. Reading reader reaction: A proposal for inferential analysis of Web server log files. In *Proc. 3rd Conf. Human Factors & The Web*, Denver, Colorado, June 1997.
14. L. Tauscher and S. Greeberg. How people revisit Web pages: Empirical findings and implications for the design of history systems. In *Int'l Journal of Human Computer Studies, Special Issue on World Wide Web Usability*, 47:97-138, 1997.
15. O. Zaiane, M. Xin, and J. Han. Discovering Web access patterns and trends by applying OLAP and data mining technology on Web logs. In *Proc. Advances in Digital Libraries Conf. (ADL'98)*, Melbourne, Australia, pages 144-158, April 1998.