Pre-order linked WAP-tree mining of sequential patterns.

Yi. Lu
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/3172

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®
Pre-Order Linked WAP-tree Mining of Sequential Patterns

By

Yi Lu

A Thesis
Submitted to the Faculty of Graduate Studies and Research through the Faculty of Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Science at the University of Windsor

Windsor, Ontario, Canada

2002
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.
Yi Lu 2002

© All Rights Reserved
Abstract

Web usage mining applies data mining techniques to the discovery of usage patterns of web data. Web usage mining mines the secondary data which are recorded users' behavior generally kept in the web log. Web usage mining can be widely used to improve the system and site design, leading to better market decisions. A navigation pattern on the web is considered a sequence of web page accesses. A sequence is an ordered list of events, and sequential mining is used to find the correlation between events. WAP-tree (Web Access Pattern tree) mining is a sequential pattern mining technique for web log access sequences. The WAP-tree technique is based on a prefix tree, which first stores the original web access sequence database, and the frequent sequences are then mined from this tree by recursively re-constructing intermediate trees.

This thesis proposes a WAP-tree based algorithm for finding frequent access sequences, which eliminates the need to reconstruct intermediate trees. In order to avoid reconstructing intermediate WAP-trees, the proposed algorithm builds the frequent header node links of the original tree in a pre-ordered fashion. It also uses position codes to identify the ancestor/descendant relationships between nodes of the tree, and finds common prefix subsequences of mined sequential patterns through a condition prefix sequence search. This results in much better response time as time for reconstructing and traversing several huge trees is saved.

Key words: Sequential pattern, Web usage mining, WAP-tree mining, PLWAP algorithm, Pre-order linkage, Position code, Apriori algorithm
To my parents
To my wife and son
To my teachers
To my friends

...
Acknowledgements

I would like to take this opportunity to thank the people who gave me help during my graduate study in University of Windsor.

 Millions of thanks to my advisor, Dr. Christie Ezeife. Without your solid help in theory and practice, this work could not have been accomplished. Thanks to you for your precious consulting time during this work. Thanks for your intensive help through all graduate studies.

 I would also thank my internal reader, Dr. Ahmed Tawfik and external reader, Dr. Myron Hlynka. Your comments, questions, and criticisms helped me to improve the thesis quality. Thanks also go to Dr. Alioune Ngom for chairing my thesis committee.

 I love to thank my parents and parents-in-law. Without their encouragement and help, I could not have continued my graduate study.

 Special thanks to my wife, Cai Liuqing. Without your understanding and contribution to the family, I could not finish the work in time.

 Finally, I would like to thank all my friends for their support, advice and help during my M.Sc. study.
# Table of Contents

Abstract........................................................................................................ IV  
Acknowledgements.................................................................................... VI  
Table of Contents ..................................................................................... VII 
List of Figures ........................................................................................... IX  
1. Introduction  
   1.1 Data Mining ................................................................................... 1  
   1.2 From Data Mining to Web usage Mining ......................................... 3  
   1.3 Data Source for Web Usage Mining ............................................... 5  
   1.4 Data Preprocessing ......................................................................... 8  
   1.5 General Discussion of Web Usage Mining ....................................10 
   1.6 Application Areas of Web Usage Mining .....................................11 
   1.7 Motivation for Thesis ....................................................................12 
   1.8 Contribution of Thesis ..................................................................13 
   1.9 Outline of Thesis ..........................................................................13  
2. Previous/Related work  
   2.1 Problem Statement ......................................................................14 
   2.2 Sequential Pattern Mining ...........................................................18 
      2.2.1 GSP Algorithm ......................................................................18 
      2.2.2 The PSP Approach ...............................................................21 
      2.2.3 Graph Traversal Mining .......................................................24 
      2.2.4 G Sequence Mining .............................................................27 
   2.3 WAP-tree Mining ..........................................................................30  
3. Pre-Order Linkage WAP-tree Mining  
   3.1 Position Code ..............................................................................38  

VII
3.2 The PLWAP Algorithm ....................................................................................41
   3.2.1 Construction of the Pre-order Linked WAP-tree .................................43
   3.2.2 Mining Web Access Patterns ...............................................................49
3.3 Algorithm Analysis ....................................................................................57

4. Experiment Evaluation and Performance Analysis .............................60
   4.1 Datasets ..................................................................................................60
   4.2 Experiment 1: Execution Time for Different Support .........................61
   4.3 Experiment 2: Execution Time for Databases with Different Size ..........62
   4.4 Experiment 3: Execution Time for Different Length Sequence ............64
   4.5 Correctness of Algorithm Implementations .........................................65

5. Conclusions and Future Work .................................................................67
   5.1 Conclusions ..........................................................................................67
   5.2 Future Work .........................................................................................67

Reference .......................................................................................................69
APPENDIX: Correctness Testing Results .....................................................74
VITA AUCTORIS ..........................................................................................87
List of Figures

Figure 1.1  Customer Purchase Data.................................................................2
Figure 1.2  Sample Web Log.............................................................................8
Figure 2.1  Sample Web Access Sequences......................................................15
Figure 2.2  Sample Tree Structure.................................................................17
Figure 2.3  Candidate Generation: Example...................................................19
Figure 2.4  Data Sequence for GSP Algorithm..................................................21
Figure 2.5  A Database Example for PSP Approach........................................22
Figure 2.6  Tree Data Structure for PSP Approach...........................................23
Figure 2.7  3-candidate Sequence for PSP Example...........................................23
Figure 2.8  Inclusion of Candidates in a Data Sequence.....................................24
Figure 2.9  A Directed Graph for Graph Traversal Mining...............................25
Figure 2.10 A Database Example Graph Traversal Mining...............................25
Figure 2.11 Large Path Generation for Graph Traversal Mining: step 1............25
Figure 2.12 Large Path Generation for Graph Traversal Mining: step 2............26
Figure 2.13 Large Path Generation for Graph Traversal Mining: step 3............26
Figure 2.14 Constructing Aggregate Tree..........................................................28
Figure 2.15 A Tree of G-sequence......................................................................29
Figure 2.16 A Database of Web Access Sequences...........................................30
Figure 2.17 Constructing WAP-tree.................................................................31
Figure 2.18 A Complete WAP-tree....................................................................32
Figure 2.19 Re-construct WAP Trees for Mining Conditional Base Pattern c........35
Figure 2.20 Re-construct WAP Trees for Mining Conditional Base Pattern b and a...36
Figure 3.1  Position Code....................................................................................39
Figure 3.2  Main PLWAP Algorithm.................................................................42
Figure 3.3  Constructing PLWAP Tree Algorithm..............................................44
Figure 3.4  Database of Web Access Sequences................................................45
Figure 3.5  Constructing PLWAP Tree Using Pre-Order Traversal.....................47
Figure 3.6  Mining PLWAP Tree Algorithm.................................................................50
Figure 3.7  Mining PLWAP tree - Find Frequent Sequence Starting with $aa$........53
Figure 3.8  Mining PLWAP tree - Find Frequent Sequence Starting with $ab$ or $ac$...54
Figure 3.9  Mining PLWAP tree - Find Frequent Sequence Starting with $b$ or $c$......56
Figure 3.10 CPU Time Comparison between PLWAP and WAP-tree Mining........57
Figure 3.11 Memory Comparison between PLWAP and WAP-tree Mining............59
Figure 4.1  Execution Time (seconds) with Minimum Different Support..............61
Figure 4.2  Execution Time Trend with Different Minimum Support.....................61
Figure 4.3  Execution Time (seconds) with Different Data Size..........................63
Figure 4.4  Execution Time Trend with Different Data Size...................................63
Figure 4.5  Execution Time (seconds) with Different Length Sequences...............64
Figure 4.6  Execution Time Trend with Different Length Sequences....................64
Figure 4.7  Sample Datasets Testing the Correctness...........................................65
Figure 4.8  Results Generating from Three Algorithms to Show the Correctness.....66
Chapter 1

Introduction

Web usage mining is the automatic discovery of user access patterns from web servers. The interesting user access patterns can be extracted from web access logs which are recorded on the web server. Analysis of these access data can provide useful information for server performance enhancements, restructuring a web site, and direct marketing in e-commerce. In this thesis, we will study how to efficiently mine the sequential access patterns from web logs. This chapter is organized as follows: section 1.1 introduces data mining concepts; Section 1.2 shows the background of web usage mining. The data sources used in web usage mining, data preprocessing, are presented in section 1.3 and 1.4 respectively. Section 1.5 and 1.6 generally discuss web usage mining and its application areas. Finally, thesis motivation (Section 1.7), contribution of thesis (Section 1.8) and organization of thesis (Section 1.9) are presented.

1.1 Data Mining

Data mining, also known as knowledge discovery in database, has received a great deal of attention in recent years. The main reason is due to the wide availability of huge amounts of data and the imminent need for turning such data into useful information and knowledge [HK00]. Data Mining is the task of discovering interesting patterns from large amounts of data where the data can be stored in databases, data warehouses or other information repositories. It is possible to strike gold in unexpected places and extract patterns not previously discernable.

One of the widely used techniques in data mining is association rule mining. The idea of association rule mining is used to discover correlation relationships among items in a transaction data. An example of transaction data is shown in figure 1.1. Transactions
(records) in these data represent items that are bought by a customer in one shopping market visit. Given a set of transactions, where each transaction consists of items, an association rule is an expression of the form \( X \rightarrow Y \), where \( X \) and \( Y \) are sets of items. A simple example is "90% of customers who purchase milk and bread, also buy eggs in the same transaction". This rule is expressed as \( \{\text{milk, bread}\} \rightarrow \{\text{egg}\} \).

![Transaction ID(TID) vs Item](image)

**Figure 1.1 Customer Purchase Data**

Two measures, **support** and **confidence**, are used for evaluating how useful association rules are. Support can be used to determine how often the rule is applied, while confidence can be used to determine how often the rule is correct.

**Definition 1.1** Given a set of items \( I \), the rule \( X \rightarrow Y \) has support \( s \) in the transaction set \( D \) means that \( s\% \) of transactions in \( D \) contains \( X \cup Y \), where \( X,Y \subseteq I \). In other words, \( s\% = \frac{\text{number of transactions containing } X \cup Y}{\text{number of transactions in the database } D} \). For example, using the simple transaction database in figure 1.1, the items set \( I \) in this example is \( \{\text{butter, bread, cheese, egg, milk, sugar}\} \). We can find that \( \{\text{bread, milk}\} \) occurs in transactions 1, 2 and 5. Thus, the support for the set of items \( \{\text{bread, milk}\} = \frac{3}{5} \times 100\% = 60\% \).

**Definition 1.2** The rule \( X \rightarrow Y \) holds in the transaction set \( D \) with confidence \( c \) if \( c\% \) of transactions in \( D \) that contains \( X \) also contain \( Y \). The confidence \( c \) can be represented as \( c\% = \frac{\text{number of transactions containing } X \cup Y}{\text{number of transactions containing } X \text{ in database}} \). In the example, \( \{\text{bread}\} \) occurs in transactions 1, 2, 3 and 5, thus, the confidence of rule \( \text{bread} \rightarrow \text{milk} \) is \( \frac{3}{4} \times 100\% = 75\% \).
An itemset is a set of items. An itemset that contains k items is called a k-itemset. For example, itemset \{milk, egg, sugar\} is a set of items \{milk\}, \{egg\} and \{sugar\}, and this is a 3-itemset. Two measures, minimum support and minimum confidence, which are given by the user, can be used to compute a frequent itemset. If the occurrence of an itemset in the database is greater than the minimum support, we call it a frequent itemset. For example, using a minimum support 3, in figure 1.1, \{bread\}, \{milk\} and \{sugar\} are a frequent 1-itemset.

While association rule mining is used to find intra-transaction pattern, sequential pattern mining is the technique to find inter-transaction patterns, which means the presence of a set of items in a time-ordered sequence of transactions. In association rule mining, the items occurring in one transaction have no order. But in sequential pattern mining, there exists order between the items (events). For example, in a video rental store, 80% of customers typically rent "Star wars", then "Empire strikes back", and then "Return of Jedi". In association rule mining, we do not care which video is rented first by the customer. But in sequential pattern mining, we are concerned with the order of items in transactions. In sequential pattern mining, the measures used in association rule mining, support and confidence are still used.

1.2 From Data Mining to Web Usage Mining

Since the 1990's, the World-Wide-Web has emerged as an important medium for transacting commerce as well as for dissemination of information related to a wide range of topics. These huge amounts of data raise a grand challenge, namely, how to turn the web into a more useful information utility [MBN+99]. Data mining technique is believed to be one of the methods to strike the useful information from these amounts of data on various web sites. Etzioni [Etz96] first coined the term web mining. He gave a definition of web mining, as the use of data mining techniques to automatically discover and extract information from web documents and service.

S. Madria et al. [MBN+99], J. Borges et al. [BL99], and J. Srivastava et al.
[SCD+00], proposed to categorize web mining into three areas of interest based on which part of the web to mine: web content mining, web structure mining and web usage mining.

Web content mining requests the discovery of useful information from the real data on web pages, such as the data that web page was designed to convey to the users. This usually consists of several types of data such as textual, image, audio, video, metadata, as well as hyperlinks. The web content data include free texts, semi-structured data like HTML documents, and more structured data like data in tables, as well as database generated HTML pages and XML pages.

Web structure mining discovers the model underlying the link structures of the web. The model is based on the topology of the hyperlinks with or without the description of the links. This model can be used to categorize web pages and is useful for generating information such as similarity and relationship between different web sites. Web structure mining could be used to discover authority sites which are sites organized for particular subjects and have many links to other related web sites based on the subject. Moreover, it can be used to find the hubs which are points to many authorities regarding one particular subject.

Web usage mining makes sense of the data generated by the web surfer's session or behavior. While the web content and structure mining utilize the real or primary data on the web, such as HTML or XML files, web usage mining mines the secondary data which is not for user browsing but for recording user's behavior. In web usage mining, what we are interested in, is to find the relationship among different users' accesses. For example, we may find the correlation: 90% of clients who accessed the page with URL /products/product.html, also accessed the page /contact/contact.html. This information give us an implication that these two pages have a close relation and they can be organized together to provide users with an easier way for browsing.
1.3 Data Source for Web Usage Mining

As we all know, the World-Wide-Web is a repository of large amounts of data. The web data used in web usage mining are secondary data in the web log. The data in the web log are derived from the results of the interactions between user and the web server. There kinds of data are not for browsing, and the customer cannot always access them. But every access or request from client to server will be recorded in the web log. Thus, the web log can be considered the repository of users requests for a particular server. All users' behavior on this server can be extracted from the web log.

In order to create some consistency in the discussion that follows, definitions of terms relevant to web usage mining are given below.

- Web server log: A file that explicitly records the browsing behavior of site visitors.
- Browser profile: Data that provide demographic information about users of the web site. This includes registration data and customer profile information.
- Page View: A page view consists of every file that contributes to the display on a user's browser at one time. Page views are usually associated with a single user action (such as a mouse-click) and can consist of several files such as frames, graphics, and scripts.
- Click-stream: A click-stream is a sequential series of page view requests. Again, the data available from the server side do not always provide enough information to reconstruct the full click-stream for a site.
- User session: A user session is the click-stream of page views for a single user across the entire Web. Typically, only the portion of each user session that is accessing a specific site can be used for analysis, since access information is not publicly available from the vast majority of web servers.
- Packet sniffer: Packet sniffing technology is an alternative method to collecting usage data through server logs. Packet sniffer monitors network traffic coming to a web server and extract usage data directly from TCP/IP packets.
- Cookies: Cookies are tokens generated by the web server for individual client browsers in order to automatically track the site visitors.
The web server log records the browsing behavior of site visitors. Normally, these users' browsing information can be stored in various formats based on the requirement and platform the server used. But the most useful and common information is provided in the common log format, which was specified by the CERN (European Organization for Nuclear Research) and the NCSA (National Center for Supercomputing Applications) as given below:

```
host/ip user [date:time] "request" status bytes
```

Normally, there are hundreds and thousands of lines with the format above, which are recorded in the web log of any web server.

An Example of a line of data in a web log is:

```
137.207.76.120 - [30/Aug/2001:12:03:24 -0500]
"GET /jdkl.3/docs/relnotes/deprecatedlist.html HTTP/1.0" 200 2781
```

- Host/IP: "137.207.76.120 ". This is the address of which computer makes the request to get the information. The server records the IP and then, if configured, will lookup the Domain Name Server (DNS). In this case, the visitor comes from the University of Windsor.

- User (or identification): Rarely used, the field was designed to identify the requestor. If this information is not recorded, a hyphen (-) holds the column in the log. In this case, "-" is shown in the log.

- Time Stamp: [30/Aug/2001:12:03:24 -0500]. The date, time, and offset from Greenwich Mean Time are recorded for each hit. The date and time format is: DD/Mon/YYYY HH:MM:SS. The example above shows that the transaction was recorded at 12:03:24 p.m. on Aug 30, 2001 at a location 5 hours behind GMT.

- Request: "GET /jdkl.3/docs/relnotes/deprecatedlist.html". One of the three types of HTTP requests is recorded in the log. GET is the standard request for getting a document or program. POST is used for uploading a document from
either the local file system or a remote HTTP server to a remote HTTP server (destination). HEAD is used for requesting only HTTP headers of the supplied URL and no document body. The specific level of HTTP protocol is also recorded. In the above example, GET is used, and the document which the user requested is "deprecatedlist.html" under the "/jdk1.3/docs/relnotes/" directory.

- **Status Code:** "200". There are four classes of codes, which are success (200 series), redirect (300 series), failure (400 series), server error (500 series). A status code of 200 means the transaction was successful. Common 300-series codes are 302, for a redirect from http://www.mydomain.com to http://www.mydomain.com/, and 304 for a conditional GET. This occurs when the server checks if the version of the file or graphic already in cache is still the current version and directs the browser to use the cached version. The most common failure codes are 401 (failed authentication), 403 (forbidden request to a restricted subdirectory), and the dreaded 404 (file not found) messages. In this case, the request is success, so, the status shows 200.

- **Bytes:** "2781". For GET HTTP transactions, the last field is the number of bytes transferred. For other commands, this field will be a hyphen (-) or a zero (0). In the above example, the bytes transferred to user are 2781.

Although most projects extract the information from the common log, the common log cannot reflect every user's visit behavior one hundred percent. The cached page views cannot be recorded in server log, and the information passed through the POST method will not be available in server log. Thus, in order to gain the proper web usage data source, we may need some extra techniques, such as packet sniffer and cookies. In most cases, researchers are assuming user visit information is completely recorded in the web server log like in Figure 1.2.
<table>
<thead>
<tr>
<th>#</th>
<th>IP address</th>
<th>Userid</th>
<th>Time</th>
<th>Request(URL)</th>
<th>Status</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123.456.78.9</td>
<td>-</td>
<td>[18/Oct/2001:08:09:41-0500]</td>
<td>&quot;GET A.HTML HTTP/1.0&quot;</td>
<td>200</td>
<td>3218</td>
</tr>
<tr>
<td>2</td>
<td>123.456.78.9</td>
<td>-</td>
<td>[18/Oct/2001:08:10:34-0500]</td>
<td>&quot;GET B.HTML HTTP/1.0&quot;</td>
<td>200</td>
<td>2049</td>
</tr>
<tr>
<td>3</td>
<td>123.456.78.9</td>
<td>-</td>
<td>[18/Oct/2001:08:11:25-0500]</td>
<td>&quot;GET C.HTML HTTP/1.0&quot;</td>
<td>200</td>
<td>590</td>
</tr>
<tr>
<td>4</td>
<td>123.456.78.9</td>
<td>-</td>
<td>[18/Oct/2001:08:11:50-0500]</td>
<td>&quot;GET D.HTML HTTP/1.0&quot;</td>
<td>200</td>
<td>790</td>
</tr>
<tr>
<td>5</td>
<td>123.456.78.9</td>
<td>-</td>
<td>[18/Oct/2001:08:12:23-0500]</td>
<td>&quot;GET A.HTML HTTP/1.0&quot;</td>
<td>200</td>
<td>3218</td>
</tr>
<tr>
<td>6</td>
<td>123.456.78.9</td>
<td>-</td>
<td>[18/Oct/2001:08:13:48-0500]</td>
<td>&quot;GET F.HTML HTTP/1.0&quot;</td>
<td>200</td>
<td>2789</td>
</tr>
<tr>
<td>7</td>
<td>123.456.78.9</td>
<td>-</td>
<td>[18/Oct/2001:08:14:15-0500]</td>
<td>&quot;GET G.HTML HTTP/1.0&quot;</td>
<td>200</td>
<td>4190</td>
</tr>
<tr>
<td>8</td>
<td>123.456.78.9</td>
<td>-</td>
<td>[18/Oct/2001:08:15:50-0500]</td>
<td>&quot;GET B.HTML HTTP/1.0&quot;</td>
<td>200</td>
<td>2049</td>
</tr>
</tbody>
</table>

Figure 1.2 Sample Web Log

As [SCD+00] said: "The usage data collected at different sources will represent the navigation patterns of different segments of the overall web traffic, ranging from single-user, single-site browsing behavior to multi-user, multi-site access patterns".

Today, most web usage mining projects take single-site, multi-user, server-side usage data (web server logs) as input.

1.4 Data Preprocessing

Basically, a data mining process consists of 7 steps, which are data cleaning, data integration, data selection, data transformation, data mining, pattern evaluation, and knowledge presentation [HK00]. Simply said, it can be viewed as three groups, data preprocessing, pattern discovery, and pattern presentation. The data preprocessing step is used to transform the raw data to a proper format for data mining. The pattern discovery step is used to extract the data patterns from data source. And the pattern presentation step is used to present the mined knowledge to the user.

The main goal of data preprocessing for web usage mining is to create a minable object. Generally, when a user visits the web site, a variety of files are accessed as the result of the requests from client side. These files include image, sound, video,
executable cgi file, and HTML file. Some of them have nothing to do with the pattern discovery process, and should be removed from the logs, which will be applied to the discovery process. Thus, we should preprocess the web server log before necessary data are involved in the pattern discovery step. This preprocess step is first performed by checking the suffix of the URL name. All log entries with suffix gif, jpeg, jpg, wav, will be removed from the log. Another important step in preprocessing is how to identify a user transaction. As we can see, in the common log, the only part we can use for identifying every user is Host/IP. But sometimes, one IP may represent several users who share one IP address.

In the best case, we can rely on the value in the field userid to accurately identify a user. But in most cases, the field userid is empty. In the absence of such information, host/IP is the only available choice to identify a user. This is assuming every user has a unique IP address within which only one type of browser is operated. However, this is not necessarily correct. As stated in [SCD+00], the following are cases that break the assumption.

- Several users may access a server through a single proxy potentially at the same time.
- Some ISPs (Internet Service Providers) or privacy tools randomly assign an IP address to each user's request.
- Some repeat users access the web each time from a different machine.
- A user may operate many browsers of different types at the same machine and potentially at the same time.

Once a user has been identified or approximated, the ordered click-stream of each user must be divided into server-sessions or visits. It is done by identifying the last page view of each visit. Without the presence of an explicit sign-out event or access to a complete user-session, page view time can be used to determine if a user has continued the same visit by selecting another page. Catledge and Pitkow in [CP95] have studied user page view time over WWW and have recommended 30 minutes of inactivity as an indication of a sign-out event. Note that since a user may not be interested only in the
pages of one site or potentially leave and re-enter the same site at different intervals, session identification may also become a difficult task. In such case, another technique, cookie is widely used to identify a particular user by storing a piece of information in the user's computer. The cookie is stored in the user's computer at the first time that the user logs into the web site. After that, when the user requests any web pages from web server, the cookie is sent with the request to the web server to identify the user. When the cookie is used, the web log needs an extra field to store the user's information from the cookie.

1.5 General Discussion of Web Usage Mining

In [SCD+00], J.Srivastava et al. enumerated possible methods, which can be used in web usage mining, such as statistical analysis, association rule, clustering, classification, sequential patterns and dependency modeling. In practice, association rule, clustering and sequential patterns are the three most common methods in web usage mining.

Association rule discovery using the Apriori algorithm [AS94] can relate pages that are most often referenced together in a single user session. Thus, the presence or absence of such rules can help web designers to restructure their web site. The association rule method used in web usage mining is mostly the same as used in database or data warehouse.

Clustering is a technique to group together a set of items having similar characteristics. In web usage domain, there are two kinds of interesting clusters to be discovered: usage clusters and page clusters. Clustering of users [FSS99] tends to establish groups of users exhibiting similar browsing patterns. Clustering of pages will discover groups of pages having related content. [YJG+96] is believed to be the first paper that focused on clustering users based on their access pattern.

In sequential pattern mining, web marketers can predict future visit patterns which will be helpful in placing advertisements aimed at certain user groups. For example, starting from Yahoo!'s home page, users can locate information on University in Canada
by following either Home → Education → Higher Education → Colleges and Universities → By Region → Countries → Canada or Home → Regional → Countries → Canada → Education → Higher Education. Thus, a university that wants to attract prospective students can place an advertisement on any of the pages along the path.

In web usage mining, the sequential pattern method looks more straightforward than others do. Normally, the users browse the web page by page, in a sequential pattern. Most of the papers in web usage mining [BS00][NM00][Spi99] focus on the area of sequential mining.

1.6 Application Areas of Web Usage Mining

Web usage mining was first used to help improve system design and make better marketing decisions. After that, a wide range of applications was introduced by extracting usage patterns from web data. They can roughly be divided into 5 categories [SCD+00]:

1. Personalization

   Personalization of web-based applications is widely used in individualized marketing for e-commerce. Making dynamic recommendations to a web user is based on his/her profile in addition to the usage behavior [MDL+00][JFM97].

2. System Improvement and Protection

   Web usage mining provides the key to understanding web traffic behavior, which can be used for developing policies for web caching, network transmission, load balancing or data distribution. Also, it can provide patterns which are useful for detecting intrusion, fraud, attempted break-in [LS98][MPT99].

3. Site Modification

   Web usage mining provides detailed feedback on user behavior, providing the web site designer information on which to use redesign decisions. The usage pattern discovered from server logs could lead automatically to changing the structure of a site. Clustering of pages is used to determine which page should be directly linked
[PE97][SPF99][MCS99].

4. Business Intelligence

Information on how customers are using a web site is critical information for marketers of retailing businesses. Web server logs are used to generate beliefs about the access patterns of web pages at a given web site [BM98].

5. Usage characterization

Web usage mining can be used to predict the probability distribution for various pages a user might visit on a given site [MCS00].

1.7 Motivation for Thesis

The use of World Wide Web as the means for marketing and selling has increased dramatically in recent years. As the e-commerce activities become more important, organizations must spend more time to provide the right level of information to their customers. Web usage mining is the application of established data mining techniques to analyze web site usage. For an e-commerce company this means detecting future customers likely to make a large number of purchases, or predicting which online visitors will click on what commercials or banners based on observation of prior visitors who have behaved both positively and negatively to the advertisement banners.

During past 6 years, techniques for mining sequential pattern from web logs fall into Apriori or non-Apriori. The Apriori-like algorithms generate a huge set of candidate patterns, especially when the sequential pattern is long. WAP-tree (Web Access Pattern tree) mining uses a different approach which stores the web access pattern into a compact tree. Thus, it can avoid generating a large amount of candidate sets. However, the WAP-tree algorithm has the drawback of recursively constructing the WAP-tree which is time-consuming. This thesis proposes an algorithm to solve this problem and achieve better performance.
1.8 Contribution of Thesis

This thesis presents a new algorithm, which is called PLWAP (Pre-order Linked WAP-tree mining), for efficiently mining sequential patterns from web logs. The algorithm inherits the prefix tree data structure from WAP-tree mining. The mining process is based on the prefix tree data structure. However, the occurrence of any frequent event (item) on the PLWAP tree is defined using a pre-order linking of these frequent event nodes. The original WAP tree would simply link them in the order the events arrive. With this pre-order linkage and the assignment of position codes for identifying each node's ancestor and descendants, PLWAP is able to compute frequent sequential patterns from prefix sequence to suffix without numerous recursive reconstruction of intermediate WAP trees as done by the WAP techniques.

1.9 Outline of Thesis

The rest of the thesis is organized as follows. Chapter 2 reviews existing related work to the thesis. Chapter 3 presents a detailed description of the new algorithm PLWAP for mining sequential patterns on WAP tree. Chapter 4 describes system implementation and performance analysis. Chapter 5 gives conclusions and discusses future work.
Chapter 2

Previous/Related work

The term web usage mining was introduced by Chen et al. [CPY96] in 1996. After that, several methods [MTV95][AS95] which were first used in data mining were applied to this area. Finding the sequential pattern in web logs is one of the most important issues in web usage mining. The sequential mining was introduced in 1995 [AS95], and it inherits the main idea of association rule mining [AS94], but it is more sophisticated than the association rule mining. After that, a lot of work had been proposed, which included incremental updating [PZO+99][MPT00], parallel algorithm [SK98][JKK00] of web logs. The GSP [SA96], PSP [MPC99], G sequence [Spi99] and graph traversal [NM01] algorithm were considered most important algorithms in developing sequential patterns in web log mining. In this chapter, we will go through these works and discuss another novel algorithm called WAP-tree mining [PHM+00] on which this thesis is based.

2.1 Problem Statement

A web log can be regarded as a sequence of events (or items), which include the user identifier, access time and access content. After preprocessing, the web log can be grouped into a set of access sequences based on user identifier.

For example, given a fragment of a web log shown below (for simplicity, every event is composed of user identifier and access content <User ID, Access content>):

- `<100,a><100,b><200,e><200,a><300,b><200,e><100,d><200,b><400,a><400,f>`
- `<100,a><400,b><300,a><100,c><200,c><400,a><200,a><300,b><200,c>`
- `<300,f><400,c><400,f><400,c><300,a><300,e><300,c>`

During preprocessing, this web log is converted to a web access sequence in the form of a transaction database, where each tuple represents a transaction ID and a
sequence of its web accesses. Thus, for example, user ID 100, from the given web log, has accessed contents, a then b,d,a, and c. The transaction web access sequences from these given web log data are shown as figure 2.1.

<table>
<thead>
<tr>
<th>User ID</th>
<th>Web Access Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>abdac</td>
</tr>
<tr>
<td>200</td>
<td>eaebcace</td>
</tr>
<tr>
<td>300</td>
<td>babfaec</td>
</tr>
<tr>
<td>400</td>
<td>afbacfc</td>
</tr>
</tbody>
</table>

*Figure 2.1 Sample Web Access Sequence Database*

The problem of mining sequential pattern from web logs is now based on the database shown in figure 2.1.

**Definition 2.1:** Given a set of events $E$, the access sequence $S$ can be represented as $e_1e_2...e_n$, where $e_i \in E$ $(1 \leq i \leq n)$. That means the access sequence is composed of a series of events which are member of event set $E$. It is not necessary that $e_i \neq e_j$ for $(i \neq j)$ in an access sequence $S$. Repetition is also allowed. For example, in figure 2.1, $E$ = \{a,b,c,d,e,f\} and an $S$ is $abdac$.

**Definition 2.2:** $|S| = n$ is called the length of an access sequence. An access sequence with length $n$ is called an $n$-sequence. The sequence with the length 1 can be considered as 1-sequence which is composed of a single event. For example, $abdac$ is a 5-sequence.

**Definition 2.3:** WASD (*Web Access Sequences Database*) = \{$S_1,S_2,..,S_m$\}, where $S_i$ $(1 \leq i \leq m)$ are access sequences. Also, we use $|WASD|$ to indicate the number of access sequence in the database. For example, figure 2.1 is a WASD with 4 access sequences $abdac$, $eaebcace$, $babfaec$ and $afbacfc$ in the database.

**Definition 2.4:** An Access sequence $S' = e_1'e_2'...e_l'$ is called a subsequence of the access sequence $S$ = $e_1e_2...e_n$, and $S$ is a super-sequence of $S'$, denoted as $S' \subsetneq S$, if and only if there exists $1 \leq i_1 < i_2 < ... < i_l \leq n$, such that $e_i' = e_j$ for $(1 \leq j \leq l)$. That means, every event $e_i'$ in $S'$ should be equal to one of the events $e_j$ in $S$, while the order that events which occur in $S$ should follow the order of events in $S'$. For example, if $S' = ab$, $S = babcd$, we can say that $S'$ is subsequence of $S$. We can also say that $ac$ is a subsequence of $S$, although there
is $b$ occurring between $a$ and $c$ in $S$.

**Definition 2.5:** A Pattern is an access sequence which will be discovered during the process as being frequent or having support that is higher than minimum support.

**Definition 2.6:** In the access sequence $S = e_1 e_2 \ldots e_k e_{k+1} \ldots e_n$, if the subsequence $S_{\text{suffix}} = e_{k+1} \ldots e_n$ is a super sequence of the pattern $P = e'_1 e'_2 \ldots e'_l$, and if $e_{k+l} = e'_l$, then the subsequence of $S$, $S_{\text{prefix}} = e_1 e_2 \ldots e_k$, is called the prefix of $S$ with respect to the pattern $P$. For example, in the sequence $eaebcac$, we can say $eae$ is a prefix of $bcac$, while the $bcac$ is a suffix of $eaebcac$.

**Definition 2.7:** The support of pattern $S$ in WASD is defined as

$$\text{Support}(S) = \frac{|S| \ S \preceq S_i, S_i \subseteq \text{WASD}}{|\text{WASD}|}$$

**Definition 2.8:** Given a sequence $S = S_1 S_2$, the confidence of sequential pattern $S_1 \rightarrow S_2$ is defined as:

$$\text{Confidence}(S_1 \rightarrow S_2) = \frac{|S_i | S_1 S_2 \preceq S_i, S_i \subseteq \text{WASD}}{|S_i | S_1 \preceq S_i, S_i \subseteq \text{WASD}}$$

In other words, the support of sequence $S$ = the number of sequence $S_i$, which contain the subsequence $S$, divided by numbers of transactions in the database WASD. The confidence of pattern $S_1 \rightarrow S_2$ = the number of sequence $S_i$ which contain the subsequence $S_1 S_2$, divided by the number of sequence $S_i$ which contain the subsequence $S_1$.

Although events can be repeated in an access sequence, any pattern can get support at most once from one access sequence. For example, from figure 2.1, $fc$ is a pattern which gets 50% support from user 300 and 400. $fc$ appears twice in user 400, but only one can contribute to the count of $fc$.

Similar to the association rule mining definition, the **minimum support** of sequence pattern mining is a number between 0 and 1 that is set by the user to identify frequent sequences.
**Problem Statement:** Given a web access sequence database WASD and a minimum support threshold \( \lambda \), the problem of web usage mining is to find all patterns which have support greater than \( \lambda \).

As we can see, many works in sequential pattern mining are based on the tree data structure. A tree is a data structure accessed beginning at the root node (see figure 2.2(a)). Each node is either a leaf or an interior node. A leaf is an item with no child. An interior node has one or more child nodes and is called the parent of its child nodes. All children of the same node are siblings, like nodes B, C and D in figure 2.2.

![Tree structure](image)

**Figure 2.2 Sample Tree Structure**

For sequential pattern mining, every sequence in the database can be represented on the tree. A path from the root to any node in the tree defines a sequence. For any node labeled \( e_i \) in a WAP-tree, all nodes in the path from the root of the tree to this node (excluding the node) form a **prefix sequence** of \( e_i \). The count of this node \( e_i \) is called the **count** of the prefix sequence. Any node in the prefix sequence of \( e_i \) can be the **ancestor** of \( e_i \). On the other hand, the nodes from \( e_i \) (excluded) to the leaves form the **suffix sequences** of \( e_i \). Any node in the suffix sequence can be the **descendant** of \( e_i \). The suffix sequence of \( e_i \) is not unique. Normally, there are several children of \( e_i \) in the tree. Here, we use **suffix tree (forest)** to represent them. The suffix tree (forest) are rooted with several nodes. These nodes are stored into the **roots set**, which can represent a suffix tree (forest) virtually. As we can see, figure 2.2(b) presents the suffix tree (forest) of root "A". \( \{B, C, D\} \) is the roots set of the suffix tree. We use **left-tree** and **right-tree** to indicate the
relationship of nodes in suffix trees. We can see in figure 2.2(a), "B" is left-tree of "C" or "D". And "D" is right-tree of "B" or "C". "E" belongs to the left-tree of "C" or "D".

2.2 Sequential Pattern Mining

Most early sequential pattern mining research [AS95][SA96][MTV95] appeared before the web usage mining term came out. Hence, these algorithms focused on finding sequential patterns from large databases. The Maximal Reference Sequences algorithm [CPY96] was the first method used for web usage mining. In this section, we introduce some important issues on both sequential pattern mining in a large database and a web log.

2.2.1 GSP algorithm

The problem of mining sequential patterns was brought up by R. Agrawal and R. Srikant [AS95]. In [AS95], they proposed three algorithms (Apriori, AprioriAll, AprioriSome) to handle this problem. After that, the same authors proposed the GSP (Generalized Sequential Patterns) [SA96] algorithm which is 20 times faster than the Apriori algorithm in [AS95].

The GSP Algorithm makes multiple passes over the data. The first pass determines the frequent 1-item patterns ($L_1$). Each subsequent pass starts with a seed set: the frequent sequences found in the previous pass ($L_{k-1}$). The seed set is used to generate new potentially frequent sequences, called candidate sequences ($C_k$). Each candidate sequence has one more item than a seed sequence. So, all the candidate sequences in a pass will have the same number of items. The supports for these candidate sequences are found during the pass over the data. At the end of the pass, the algorithm determines which of the candidate sequences are actually frequent ($L_k$). These frequent candidates become the seed for the next pass. The algorithm terminates when there are no frequent sequences at the end of a pass, or when there are no candidate sequences generated. There are two key
details, which are candidate generation and counting candidates.

For **candidate generation**, there are 2 steps listed below:

1. Join phase: In order to obtain k-sequence candidate \(C_k\), the frequent sequence \(L_{k-1}\) joins with itself Apriori-gen way. This requires that every sequence \(s\) in \(L_{k-1}\) joins with other sequences \(s'\) in \(L_{k-1}\) if the last elements of \(s\) are the same as the first elements of \(s'\).

   For example, in figure 2.3, the frequent 3-sequence set \(L_3\) has 6 sequences. In order to obtain frequent 4-sequences, every frequent 3-sequence should join with the other 3-sequences that have the same first elements. Sequence \(s = \langle(1,2)(3)\rangle\) can join with \(s' = \langle(2)(3,4)\rangle\) to generate a candidate 4-sequence because the last elements of \(s\), \((2)(3)\), are the same as the first elements of \(s'\). Then, we can add element \((4)\) to the sequence \(\langle(1,2)(3)\rangle\). And because the element \((4)\) is part of last element \((3,4)\) of \(s'\) \(\langle(2)(3,4)\rangle\), the new sequence should be \(\langle(1,2)(3,4)\rangle\). Also, \(\langle(1,2)(3)\rangle\) can join with \(\langle(2)(3)(5)\rangle\) to form \(\langle(1,2)(3)(5)\rangle\). The remaining sequences cannot join with any sequence in \(L_3\). \(\langle(1,2)(4)\rangle\) does not join with any sequence since there is no sequence of the form \(\langle(2)(4\ x)\rangle\) or \(\langle(2)(4)(x)\rangle\).

2. Prune phase: The candidate sequences that have a contiguous \((k-1)\)-subsequence with support counts less than the minimum support are dropped. This means that there exist \((k-1)\) subsequences of the candidate sequence, not in the \(L_{k-1}\). In figure 2.3, the \(\langle(1,2)(3)(5)\rangle\) will be dropped since its contiguous subsequence \(\langle(1)(3)(5)\rangle\) is not in \(L_3\).

<table>
<thead>
<tr>
<th>Frequent 3-Sequences</th>
<th>Candidate 4-Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>\langle(1,2)(3)\rangle</td>
<td>\langle(1,2)(3)(4)\rangle</td>
</tr>
<tr>
<td>\langle(1,2)(4)\rangle</td>
<td>\langle(1,2)(3)(5)\rangle</td>
</tr>
<tr>
<td>\langle(1)(3,4)\rangle</td>
<td></td>
</tr>
<tr>
<td>\langle(1,3)(5)\rangle</td>
<td></td>
</tr>
<tr>
<td>\langle(2)(3)(4)\rangle</td>
<td></td>
</tr>
<tr>
<td>\langle(2)(3)(5)\rangle</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 2.3 Candidate Generation: Example*

After the pruning phase in **candidate generation** process, we have the candidate
sequence for testing. The testing process in GSP algorithm is called counting candidates. There are 2 steps in solving the problem:

1. Use a hash-tree structure to reduce the number of candidates in C that are checked for data-sequence.

   The author adopts the hash-tree data structure which was introduced in [AS94]. A node of the hash-tree either contains a list of candidate sequence (leaf node) or a hash table (interior node). In the interior node, each non-empty bucket of the hash table points to another node.

   When finding the candidates contained in a data-sequence, we start from the root node. Applying the hash function to each item in the data sequence, only the sequences not starting with an item in the data sequence are ignored. Then, recursively apply this procedure to the next depth nodes, until the leaf is reached. When the leaf is reached, whether the data sequence contains a candidate sequence is based on the procedure listed below.

2. Transform the representation of the data-sequence $d$ so that we can efficiently find whether a specific candidate is a subsequence of $d$.

   When we check whether the candidate sequence is a subsequence of the data sequence $d$, there are two phases, forward phase and backward phase, during the checking. First of all, performing the forward phase requires holding an element and then finding successive elements of the candidate sequence $s$ in the data sequence $d$. If the time difference between the previous element and next element is more than max-gap, then the algorithm switches to the backward phase which holds an element and tries to find a previous element.

   For example, a data sequence is presented in figure 2.4. We want to check whether the data sequence can contribute a count to candidate sequence $<(1,2)(3)(4)>$. Assume the max-gap of transaction time is 30, min-gap of transaction time is 5, which means the time difference between any two elements is within the range 5 to 30. We first find (1,2) at transaction-time 10, and find (3) at time 45. The gap between these two
elements is 35 and more than max-gap 30. So, we start backward phase. Hold the element (3) at time 45, we just try to find the (1,2) before time 40 (min-gap is 5) and after the time 15 (max-gap is 30), and no one satisfied. Then, switch to forward phase, we find (1,2) at 50, and (3) at 65. The time difference of these two elements is 15, and meets the min-gap and max-gap requirement. Then, we hold the element of (3) at time 65, and keep finding (4) at 90, the gap between (3) and (4) is 25 which is greater than min-gap and less than max-gap. So, the data sequences in figure 2.4 can contribute count 1 to the candidate sequence <(1,2)(3)(4)> . Thus, the count of candidate sequence <(1,2)(3)(4)> adds 1. After checking all data sequences in the database, we will obtain the count of candidate sequence <(1,2)(3)(4)> . We can finally decide whether the candidate sequence is frequent or not based on the count we get.

<table>
<thead>
<tr>
<th>Transaction Time</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1,2</td>
</tr>
<tr>
<td>25</td>
<td>4,6</td>
</tr>
<tr>
<td>45</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>1,2</td>
</tr>
<tr>
<td>65</td>
<td>3</td>
</tr>
<tr>
<td>90</td>
<td>2,4</td>
</tr>
<tr>
<td>95</td>
<td>6</td>
</tr>
</tbody>
</table>

*Figure 2.4 Data Sequence for GSP Algorithm*

Limitations:

GSP is efficient when the sequences are not long. When the length of sequences increases and/or when the numbers of transactions is large, the number of candidate sequences generated may grow exponentially, and GSP will encounter difficulties.

2.2.2 The PSP Approach

The PSP (PSP : Prefix Tree For Sequential Patterns) approach [MPC99] proposed by F. Masseglia et al. is similar to the GSP algorithm [SA96]. At each step k, the database is browsed for counting the support of current candidates. Then, the frequent sequence set \( L_k \) can be built. From this set, new candidates are exhibited for
consideration at the next step. The algorithm stops when the longest frequent sequence is found.

The only difference between the PSP algorithm and the GSP is that PSP introduces a prefix-tree to handle the procedure. At the kth step, the tree has a depth of X. Any branch, from the root to a leaf stands for a candidate sequence, and considering a single branch, each node at depth l (k ≥ l) captures the kth item of the sequence. Meanwhile, along with an item, a terminal node provides the support of the sequence from the root to the considered leaf (included).

Using the database in figure 2.5 as the example, where URL entries are mapped into integers according to the sort phase, assume the minimum support value is 50%. After scanning the database once, we found frequent 1-sequence set \( L_1 = \{10,20,30\} \). Then, we use the Apriori method to generate the candidate 2-sequence set \( C_2 = \{<10,20>, <10,30>, <10>(20), <10>(30), <(20),30>, <(20)(10), <(20)(30), <(30)(10), <(30)(20)\}. \) After scanning database twice, we can have the frequent 2-sequence set: \( L_2 = \{<10>(30), <(10)(20), <(20)30>, <(30)(20)\}. \)

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Time</th>
<th>URL accessed</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP_1</td>
<td>01/01/1999</td>
<td>10,30,40</td>
</tr>
<tr>
<td>IP_1</td>
<td>02/02/1999</td>
<td>20,30</td>
</tr>
<tr>
<td>IP_2</td>
<td>11/01/1999</td>
<td>10</td>
</tr>
<tr>
<td>IP_2</td>
<td>12/01/1999</td>
<td>30,60</td>
</tr>
<tr>
<td>IP_2</td>
<td>23/01/1999</td>
<td>20,50</td>
</tr>
<tr>
<td>IP_3</td>
<td>01/01/1999</td>
<td>10,70</td>
</tr>
<tr>
<td>IP_3</td>
<td>12/01/1999</td>
<td>30</td>
</tr>
<tr>
<td>IP_3</td>
<td>15/01/1999</td>
<td>20,30</td>
</tr>
</tbody>
</table>

*Figure 2.5 A Database Example for PSP Approach*

It is organized according to tree structure as depicted in figure 2.6. Each terminal node contains an item and a counting value. If we consider the node having the item 20, its associated value 2 means that two occurrences of the sequence \( \{<10)(20)\} \) have been detected so far. And if the item occurred during different transactions, it can be drawn as line, otherwise it should be a dash line.
The tree represented by figure 2.7 illustrates how the $k$-candidates and the frequent $l$-sequences (with $l \leq [1..(k-1)]$) are simultaneously managed by the structure. It is obtained after the generation of the candidates of length 3 from the tree represented by figure 2.6. We can extend the tree from level 2 to level 3 as follows: first, we extend the item 30 at leftmost side in figure 2.6, and what we can extend with the item 30 is based on the level 1's extension. In level 1, we find that item 30 can only be followed by item 20. So, we extend the item 30 with 20 using a line. Similarly, item 20 can occur with 30 at the same time (which is indicated with a dash line), we extend 20 with item 30 using a dash line. The final 3-sequence tree is shown in figure 2.7.
After the tree was extended, it can be used to count the support in the next step. It still needs the database scan, and the data-sequence with respect to every IP as the input for counting. As we see in Figure 2.8, the IP \( 10 \ 30 \ 40 \ 20 \ 30 \ 30 \) is input. Then, we can scan every branch in the tree. For any branch that occurred in the data sequence, the support of that branch will be incremented. In the figure 2.8, two branches in the tree, \( (10)(20 \ 30) \) and \( (30)(20 \ 30) \), occurred in the input data sequence. So, their count is increased by 1. After counting every data sequence, all leaves not satisfying the minimum support are removed.

2.2.3 Graph Traversal Mining

The main idea of Graph Traversal mining which is proposed by Nanopoulos and Manolopoulos [NM00, NM01] takes into account the relationship between the web pages of the particular site. They introduce a simple unweighted graph which reflects the relationship between the pages in site (see, Figure 2.9).
Pre-Order Linked WAP-tree Mining of Sequential Patterns

![Directed Graph for Graph Traversal Mining](image)

**Figure 2.9 A Directed Graph for Graph Traversal Mining**

After constructing the traversal graph as in figure 2.9, we can perform the Graph Traversal algorithm which is similar to Apriori algorithm.

<table>
<thead>
<tr>
<th>ID</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;A,B,C&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;B,D,E,C,A&gt;</td>
</tr>
<tr>
<td>3</td>
<td>&lt;C,A,B&gt;</td>
</tr>
<tr>
<td>4</td>
<td>&lt;D,C,A&gt;</td>
</tr>
<tr>
<td>5</td>
<td>&lt;B,C,A&gt;</td>
</tr>
</tbody>
</table>

*Figure 2.10 A Database Example Graph Traversal Mining*

Suppose we have a web log sequence database shown in figure 2.10. First of all, we can find all candidate $C_1$, and after counting the support, we have the large 1-sequence as $L_1$ (see Figure 2.11).

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;A&gt;</td>
<td>5</td>
</tr>
<tr>
<td>&lt;B&gt;</td>
<td>4</td>
</tr>
<tr>
<td>&lt;C&gt;</td>
<td>5</td>
</tr>
<tr>
<td>&lt;D&gt;</td>
<td>2</td>
</tr>
<tr>
<td>&lt;E&gt;</td>
<td>1</td>
</tr>
</tbody>
</table>

$C_1$

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;A&gt;</td>
<td>5</td>
</tr>
<tr>
<td>&lt;B&gt;</td>
<td>4</td>
</tr>
<tr>
<td>&lt;C&gt;</td>
<td>5</td>
</tr>
<tr>
<td>&lt;D&gt;</td>
<td>2</td>
</tr>
</tbody>
</table>

$L_1$

*Figure 2.11 Large Path Generation for Graph Traversal Mining: Step 1*

Secondly, we try to find the 2-sequences whose sequence length are 2. By visiting all graph paths, if a candidate $L = \langle l_1, \ldots, l_k \rangle$ is large, then the adjacency list of $N^+(l_k)$ is retrieved. For each vertex $v$ in the adjacency list, if it does not belong to path $L$ and subpath $L' = \langle l_2, \ldots, l_k, v \rangle$ is large, then, a possible candidate $C = \langle l_1, \ldots, l_k, v \rangle$ of length $k + 1$ is formed by appending $v$ at the end of $L$. At this point, the authors generate the candidate sequence only based on the traversal graph shown in figure 2.9. For generating
candidate 2-sequence, first of all, we pick up the events from \(L_1\) in figure 2.11, which are A, B, C, and D. Then, we check with the traversal graph in figure 2.9. We can find "A" has two adjacent vertices "B" and "C", and "A" also has direct arrow point to these two vertices. So, it is possible that \(<A,B>\) and \(<A,C>\) can be added to the 2-sequence candidate set. Since any subpaths of \(<A,B>\) and \(<A,C>\) are large (A, B and C are all large in \(L_1\)), we can add these two sequences to the candidate set. After checking other events B, C and D in 2.11, we can form the 2-sequence candidate in figure 2.12.

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;A,B&gt;)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;A,C&gt;)</td>
<td>1</td>
</tr>
<tr>
<td>(&lt;B,C&gt;)</td>
<td>3</td>
</tr>
<tr>
<td>(&lt;B,D&gt;)</td>
<td>1</td>
</tr>
<tr>
<td>(&lt;C,A&gt;)</td>
<td>4</td>
</tr>
<tr>
<td>(&lt;D,C&gt;)</td>
<td>2</td>
</tr>
</tbody>
</table>

\(C_2\)

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;A,B&gt;)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;B,C&gt;)</td>
<td>3</td>
</tr>
<tr>
<td>(&lt;C,A&gt;)</td>
<td>4</td>
</tr>
<tr>
<td>(&lt;D,C&gt;)</td>
<td>2</td>
</tr>
</tbody>
</table>

\(L_2\)

**Figure 2.12 Large Path Generation for Graph Traversal Mining: Step 2**

Now, after discarding the non-frequent event \(E\), we can form at most 12 2-sequences. By using graph traversal method, we have 6 2-sequences in candidate \(C_2\). The number of candidates which have their supports counted is reduced greatly. As the algorithm keeps running, we can have the final \(L_3\) in Figure 2.13.

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;A,B,C&gt;)</td>
<td>1</td>
</tr>
<tr>
<td>(&lt;B,C,A&gt;)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;C,A,B&gt;)</td>
<td>1</td>
</tr>
<tr>
<td>(&lt;D,C,A&gt;)</td>
<td>2</td>
</tr>
</tbody>
</table>

\(C_3\)

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;B,C,A&gt;)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;D,C,A&gt;)</td>
<td>2</td>
</tr>
</tbody>
</table>

\(L_3\)

**Figure 2.13 Large Path Generation for Graph Traversal Mining: Step 3**

In [NM01], the transaction-corrupted problem is also indicated. For example, a transaction \(T = \langle A, E, B, C, F, G, D \rangle\) occurred. If \(P = \langle A, B, C, D \rangle\) is the pattern, the support in [CPY96] will not be increased. As in Graph Traversal algorithm, it can be
counted, and those are not of a pattern that will be considered as the noise which is called corruption. The idea is much similar to the G sequence proposed in [Spi99]. Thus, the support count algorithm is much different from the [CPY96]. In [NM01], the authors proposed the prefix tree instead of hash-tree as the structure for storing the path.

2.2.4 G Sequence Mining

M. Spiliopoulou [Spi99] proposed an approach which is called G sequence mining. She introduced several ideas to handle the mining process.

First of all, the wildcard was introduced in her approach. This is why her approach is called g (generalized) sequence mining. Let \( U \) be a set of elements, the root \( ^\wedge \) and the wildcard \( * \) be two special symbols. Let \( U^+ = U \cup \{^\wedge, *\} \). A "generalized sequence" or "g-sequence" is a vector \( g \subseteq U^+ \) such that: (1) at least one element of \( g \) belongs to \( U \), (2) the first element is either the \( ^\wedge \) or an element of \( U \), (3) no other but the first element of \( g \) can be equal to the root, (4) no two adjacent elements of \( g \) are wildcards.

Secondly, the template was introduced in the paper. A template is a vector of variables with distinct names, which can be mapped to a g-sequence by some binding function. A sequence \( s \) "satisfies" \( t \), if and only if there is a function \( f \) that binds the named variables of \( t \) to elements of \( s \) in such a way that \( f(t[1])f(t[2])\ldots f(t[n]) \) is a g-sequence matched by \( s \). For example, \( A^*[1,2]B \) is a template means a sequence of 3 or 4 elements, while \( A,B \) represent the elements of \( U \). After using the binding function \( f(A) = a, f(B) = d \), we can produce the g-sequence \( a^*[1,2]d \) which is matched \( abcd \). And it is said that \( abcd \) satisfies the template. With this template, sequences have 1 or 2 elements between \( a \) and \( d \) that are satisfied. On the other hand, \( ad \) does not satisfy the template.

Thirdly, she introduced the aggregate tree into the solution. An aggregate tree is a tree transformed from the original web log, each node is annotated with the number of sequences having same prefix up to and including this node.

In Figure 2.14, \( b \) is the first event in \( s_2, s_3, s_6 \) and \( s_7 \). \( (b, 1) \) denotes the first occurrence
of b; (b, 2) denotes the reoccurrence in s_2, s_6. By adding up the appearances of the sequences s_2, s_3, s_5, s_7, 14 is the support of the (b, 1) as the first sequence element. Similarly, the support of (a, 1) as first sequence element is 21.

**Figure 2.14 Constructing Aggregate Tree**

After transforming, the tree contains the same information as the initial log. So, the original log will not be used in the next steps. All mining steps after this are based on the aggregate tree.

It first traverses the aggregated log to find nodes satisfying the first template variable. All t-nodes contain bindings of the first variable from level 1.

In each subsequent step, a binding of the \( i \)th variable is possible only in the subtrees below the nodes to which the \((i-1)\)th variable have been found.

For example, using the aggregate tree in figure 2.15, we want to find all templates of the form \( X*Y*Z \), for which the following conditions hold:
X.support $\geq 20$ and Y.support $\geq 10$ and Z.support $\geq 4$.

Figure 2.15 A Tree of G Sequence

In the first step, all t-nodes of level 1 are built. From figure 2.15, we can see that there exist 7 distinct elements, resulting in the t-nodes $\langle a,1 \rangle$, $\langle b,1 \rangle$, $\langle e,1 \rangle$, $\langle f,1 \rangle$, $\langle d,1 \rangle$, $\langle c,1 \rangle$, $\langle b,2 \rangle$. Each of them is built, tested against the predicate for variable X and discarded immediately if the predicate is not satisfied. So, only three t-nodes remain.

For level 2 we traverse each subtree below a t-node of level 1. For $\langle a,1 \rangle$ there are four t-nodes, of which the one containing the appearances of $\langle f,1 \rangle$ after $\langle a,1 \rangle$ rejected because its support is 4 and less than the support of Y. For $\langle b,1 \rangle$, four t-nodes are built and three of them are discarded. For $\langle e,1 \rangle$, there is only one t-node. Since the only element appearing after $\langle e,1 \rangle$ is $\langle f,1 \rangle$, and its support is lower than the threshold for variable Y, it is therefore discarded.

For level 3 and level n, the processing keeps going the same way until no more level appears.
2.3 WAP-tree Mining

Pei et al. [PHM+00] proposed an algorithm using WAP-tree which stands for web access pattern tree. This approach is quite different from the Apriori-like algorithms. Following are the main steps involved in this technique.

Step 1: Scan web log once, find all frequent individual events.

Step 2: Scan web log again, construct a WAP-tree over the set of frequent individual events.

Step 3: Find conditional suffix patterns.

Step 4: Construct a intermediate conditional WAP-tree using the pattern found in the previous step.

Step 5: Go back to repeat steps 3 and 4 until the constructed conditional WAP-tree has only one branch or is empty.

Finding all frequent events in the web log is quite straightforward. The key point is to construct the WAP-tree and mine the access pattern from the WAP-tree.

We use the web log access sequence database in figure 2.16 to show how to construct the WAP-tree and do WAP-tree mining. Suppose the minimum support threshold is set at 75% which means an access sequence, $s$ should have a count of 3 out of 4 records in our example to be considered frequent.

<table>
<thead>
<tr>
<th>User ID</th>
<th>Web Access Sequence</th>
<th>Frequent subsequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>$abdac$</td>
<td>$abac$</td>
</tr>
<tr>
<td>200</td>
<td>$eaebcac$</td>
<td>$abcac$</td>
</tr>
<tr>
<td>300</td>
<td>$babfaec$</td>
<td>$babac$</td>
</tr>
<tr>
<td>400</td>
<td>$asbacfc$</td>
<td>$abacc$</td>
</tr>
</tbody>
</table>

*Figure 2.16 A Database of Web Access Sequences*

Constructing WAP-Tree

After scanning database once, we know which events are frequent. When constructing the WAP-tree, the non-frequent part of every sequence is discarded. Only the frequent subsequences are considered as input. For example, in figure 2.16, the list of
all events is \{a, b, c, d, e, f\}, and the support of a is 4, b is 4, c is 4, d is 1, e is 2, and f is 3. With the minimum support of 3, only a, b, c are frequent events. Then, all non-frequent event (like d, e, f) are deleted from each transactional sequence to obtain the frequent subsequence shown in column three of figure 2.16.

![Diagram of WAP-tree construction](image)

*Figure 2.17 Constructing WAP-tree*

When constructing WAP-tree, a virtual root (Root) in figure 2.17(a) is inserted first. All other sequences will be the children of the root.

1. Insert sequence *abac* of transaction ID 100. It creates a new branch in the tree which is started from the first event of sequence *abac*. Four nodes "(a:1) → (b:1) → (a:1) → (c:1)" are inserted into the tree as the children of the root (see figure 2.17(a)). Every node is composed of the event label and the count number. Here, we can see that (a:1) indicates that the event label is a and the count is 1. At the same time, the linkage with same frequent event label of a is created. The linkage is inserted in the order that the nodes are inserted in the tree.

2. Insert the sequence *abacac* of the next transaction with ID 200, again starting from the virtual root (figure 2.17(b)). Since the root has a child labeled a, the node a’s count is increased by 1 to obtain (a:2) now. Similarly, (b:2) is also in the tree. The next event, c,
does not match the next existing node $a$, and new child node $c:1$ is created and inserted as another child of tree $b$ node.

(3) The third sequence $babac$ of ID 300 is inserted next. Since there is no node labeled $b$ under root, a new node $b:1$ is created and inserted after the root node. The subsequence after $b$ ($abac$) are all new in the tree, and should be newly inserted one after the other.

(4) The fourth sequence $abacc$ of transaction ID 400 is inserted with same method. The final tree is shown in figure 2.18.

![WAP-tree diagram]

**Figure 2.18 A Complete WAP-tree**

**Mining Access Patterns from WAP-tree**

Mining the access pattern is based on the WAP-tree. We start the conditional search from last frequent event of linkage head table in figure 2.18 which is $c$.

1. Step 1
Using figure 2.18 for WAP-tree mining, we first find all sequences whose last event is c. It can be called the conditional sequence base of c:

\[ \text{aba} : 2; \, \text{ab} : 1; \, \text{abca} : 1; \, \text{ab} : -1; \, \text{baba} : 1; \, \text{abac} : 1; \, \text{aba} : -1 \]

The conditional sequence list is obtained by following the linkage of the event and reading the path from the root to each node (excluding it). The count for each path is the same as the count on the node itself. The first sequence above, \( \text{aba} \), represents the path to the first c node in the WAP-tree. When a conditional sequence in a branch of WAP-tree, has a prefix subsequence that is also a conditional sequence of same base, the count of this new subsequence is subtracted because it has contributed before. In the above sequence, when we add \( \text{abca} \) to the conditional sequence of c, we find its subsequence ab is also conditional pattern of c. Thus, the event ab with count -1 has to be added to the list to prevent it from contributing twice.

To qualify as a conditional frequent event, one event must have count 3. Therefore, after counting the events in sequences above, we find that the conditional frequent events are \( a(4) \) and \( b(4) \). Note that c with 2 counts which is less than minimum support is discarded. Then, after discarding the non-frequent part c in the above sequences, the conditional sequences based on c are listed below:

\[ \text{aba} : 2; \, \text{ab} : 1; \, \text{ab} : 1; \, \text{ab} : -1; \, \text{baba} : 1; \, \text{aba} : 1; \, \text{aba} : -1 \]

We still can not obtain the frequent sequence based these conditional sequences. The frequent events are widely scattered in different conditional sequences. We only known that frequent subsequences exist in these conditional sequences. In order to find the frequent sequences in these conditional sequences, we need to construct the conditional WAP tree and recursively obtain the frequent sequences.

Using the conditional sequences above, a conditional WAP-tree, WAP-tree|c, is built using the same method as shown in figure 2.17. And the new intermediate WAP tree conditional pattern based on "c" is shown in figure 2.19(a).

Recursively, based on the WAP-tree in figure 2.19(a), the program keeps running,
the conditional sequence base of $b$ in WAP-tree$|c$ are $b:3$; $ba:1$. Then, the conditional WAP-tree$|bc$ is built in figure 2.19(b). And the frequent patterns $abc$; $bc$ are found. Since there is only one branch in figure 2.19(b), there is no need to construct the WAP-tree again. We finish this leg and come back to find other frequent sequences in figure 2.19(a).

From figure 2.19(a), we finished finding the sequence based on $bc$, and keep to find the frequent sequences whose conditional base pattern are $ac$. The conditional sequence base of $ac$ are $ab:3$; $b:1$; $bab:1$; $b:-1$. The conditional frequent events are $a(4)$, $b(4)$. Then, the WAP-tree$|ac$ is built, shown in figure 2.19(c). The WAP-tree$|ac$ is not empty or only one branch, we keep to build the intermediate tree for conditional pattern $bac$ and $aac$. It finds the conditional sequence base of $bac$ as $a:3$; $ba:1$. The conditional frequent events are $a(4)$. Then, the conditional WAP-tree$|bac$ is built as shown in figure 2.19(d). Now, there is only one branch in the conditional WAP tree, so all combinations are generated which are $bac$, $abac$. The conditional sequence base of $aac$ is $b:1$. The count of $b$ is 1 less than the threshold and should be discarded. The conditional WAP-tree with the suffix $aac$ is shown in figure 2.19(e), which is empty. From figure 2.19(e), we know the only frequent sequence in this conditional WAP-tree is $aac$. Thus, the algorithm stops this leg of the recursive method and returns to the previous round in figure 2.19(c), which is based on the suffix of $ac$. There is no more frequent event in the linkage header table to be mined in figure 2.19(c), we go back to figure 2.19(a). In 2.19(a), all frequent events in the linkage header table have been mined, we can stop this leg.
This concludes the mining of frequent sequences from the original WAP tree of figure 2.18 that have suffix frequent event "c". The frequent sequences obtained for suffix "c" are \{c, bc, abc, ac, bac, abac, aac\}. The bc and abc are from figure 2.19(b). ac is from figure 2.19(c), bac, abac are from figure 2.19(d), and aac is from figure 2.19(e).
2. Step 2

After finding all frequent sequences with conditional base pattern "c", we start to find the frequent sequence with conditional base pattern "b" in figure 2.18. Using the same idea as in step 1, we find the conditional sequence of b: a:3; ba:1. Then, we can construct the WAP shown in figure 2.20(a). Since this tree only has one branch, all frequent sequences {b, ab} can be found.

3. Step 3

We start step 3 in figure 2.18 to find the frequent sequence with base pattern "a". After finding all conditional sequences which finished with "a", we can construct the intermediate tree in figure 2.20(b). Then, we can recursively find the conditional pattern with "ba" and "aa", finally, we find the frequent sequence {a, ba, aba, aa}.

Combining all frequent sequence sets from 3 steps above, we have the final frequent sequence set: {c, bc, abc, ac, bac, abac, aac, b, ab, a, ba, aba, aa}.

A WAP-tree based algorithm has a significant difference from the Apriori-like algorithm. As the example shown above, when we use GSP algorithm to find the frequent sequence, it scans the database 5 times. But for WAP-tree, it scans the original
database only twice. All mining work is based on the WAP-tree. Thus, it avoids the problem of generating explosive candidate sets as in Apriori-like algorithms. The mining efficiency is improved sharply. The main drawback of WAP-tree mining is that it should recursively construct the WAP-tree. The process of constructing the intermediate WAP tree composed several steps, which need I/O operation and checking the frequent event in conditional patterns. These operations in WAP-tree mining are considered time-consuming.
Chapter 3

Pre-Order Linkage WAP-tree Mining

After reviewing the previous work, we find that mining sequential patterns from the web log are based on two basic approaches, the Apriori and non-Apriori methods. Generating and testing with the Apriori algorithm is costly, and most work on generating candidate sets takes a lot of time. The WAP-tree mining is one of the non-Apriori algorithms which overcomes the drawbacks of Apriori algorithm. But as introduced in the previous chapter, the WAP-tree mining still has some aspects that can be improved on such as recursively constructing the tree. In this chapter, we present a new sequential pattern-mining algorithm which is based on the WAP-tree and can avoid recursively constructing the WAP-tree, called Pre-order Linked WAP-tree mining (PLWAP).

3.1 Position Code

As introduced in chapter 2.3, the entries in the web log can be constructed into one compact tree called WAP-tree. The main problem of WAP-tree mining is that the algorithm should construct the WAP tree recursively. This phase can be considered as the main costly part in WAP-tree mining. Can we find a method to avoid constructing the tree repeatedly when mining the original WAP tree? Unlike the condition search in WAP-tree mining, which is based on finding a common suffix sequence, we will introduce a method to find a common prefix sequence. For example, if \(abcd\) is a frequent pattern to be discovered, the WAP technique would start by finding the suffix event \(d\), then it will find the next suffix subsequence \(cd\), then \(bcd\) and finally \(abcd\). And for each of these suffix subsequences, an intermediate WAP-tree is constructed. The PLWAP algorithm on the other hand, would find the prefix event a first, then using the suffix tree of node \(a\), it will find the next prefix subsequence \(ab\) and continuing with the suffix tree of \(b\), it will find the next prefix subsequence \(abc\) and \(abcd\). The main idea of PLWAP is to use the suffix trees of the last frequent event in an \(n\)-prefix sequence to recursively
extend the subsequence to n+1 sequence by adding a frequent event that occurred in the suffix sequence. For a sequence in the WAP-tree, the events occurring at the beginning of the sequence are presented in the upper part of the WAP-tree, the remaining events of the sequence are only represented by the nodes in the lower part of the WAP-tree. So, if we know what part of the WAP-tree can be used to find the next event in frequent sequence, we need not construct the WAP-tree again. In other words, what we need to do is find the frequent event in the suffix subtree in the WAP-tree. Since we already know the frequent sequence obtained in previous steps, we obviously know the last event (say ei) of this frequent sequence. If we can find the suffix tree of ei, we can find the next frequent event of the sequence in this suffix tree. But the main problem is how to identify which sequences are suffix sequences of the last event ei? Here, we introduce a method called position code to identify the position of every node in the WAP tree. Using this method, we can find the suffix tree of a particular event. And thus, we need not reconstruct the tree again.

![Diagram of WAP-tree](image)

**Figure 3.1 Position Code**

A position code is a kind of code to indicate the position of the nodes in the WAP-tree. In data structure, a general tree is always represented as a binary tree [Sha00], which
has 2 child nodes. For a node, two pointers, LeftChild and RightSibling are used to link to other nodes. Using the LeftChild point to the leftmost son of that node and the RightSibling point to the next brother of the node. For example, given a general tree shown as figure 3.1(a), it can be transformed to the binary tree shown in figure 3.1(b).

Then, we can assign the code to the node just like Huffman coding. Here, we assign 1 for the left child of the node, and 0 for its sibling. After this assignment, we have the position code for every node as shown in figure 3.1(a). The position code of a node on the WAP tree is defined as the concatenation of all codes of its ancestors from the root to the node itself in the transformed binary tree equivalent of the tree. For example, in figure 3.1(a), the position code of node (c:1) at the leaf of the rightmost branch is obtained by concatenating all codes from path (b:1) to (c:1) of the rightmost branch of figure 3.1(b) to obtain 101111.

After observing the position code with nodes in figure 3.1(a), we have the property given below:

**Property 1:** A node $\alpha$ is an ancestor of another node $\beta$ if and only if the position code of $\alpha$ attaching 1 at the end, equals to the first ($|\text{position code of } \alpha| + 1$) bits position code of $\beta$.

For example, in figure 3.1(a), (c:1:1110) is the ancestor of (c:1:111011) Since the position code of (c:1:1110) is 1110, after attaching 1 at the end of 1110 (11101), it will be equal to the first 5 bits of (c:1:111011). On other hand, (c:1:1110) is not the ancestor of (c:1:101111), since after attaching 1, the code will be 11101 and is not equal to the first 5 bits position code of (c:1:101111).

Similarly, we can find all nodes' relationship within the WAP-tree by using position codes. Not only can we find the ancestor and descendant relationship, but also we can find whether one node belongs to the right-tree or left-tree of another node. From figure 3.1(a), we can see node (c:1:1111) and node (c:1:111011) are two nodes belonging to two trees which are rooted at (a:2:111) and (c:1:1110) respectively. We can find that the node (c:1:1111) belongs to the left-tree of (c:1:111011). The reason is that the fourth bit of (c:1:111011) is 0, which means the node is extended from the node with position code
1110. The node with position code 1110 is the right sibling of node with 111 which is ancestor of node (c:1:1111). Thus, (c:1:111011) is the right-tree of (c:1:1111).

3.2 The PLWAP Algorithm

The PLWAP algorithm is much like the WAP-tree mining introduced in section 2.3. The motivation for PLWAP algorithm is to avoid the reconstruction of WAP-tree in section 2.3. The algorithm is based on the following property which is derived from the Apriori algorithm.

Property 2: If e is a frequent event in the set of suffixes of the sequence pattern F in the database WASD, then the sequence Fe is an access pattern of WASD.

For example, if we have a sequence pattern F = ac, and b is a frequent event within suffix sets of c, we can say that acb is an access pattern.

The general idea of our PLWAP algorithm is briefly described as follows:

1. The PLWAP tree data structure which is inherited from WAP-tree mining, is used to register access sequences and corresponding counts compactly, so that tedious support counting can be avoided. Also, the position code is provided with each node in a PLWAP tree to identify the position of the node in tree. The pre-order linkages between nodes with the same label are used for traversing prefixes with respect to the same suffix pattern efficiently. The PLWAP tree registers all information needed for the rest of the mining. Once a PLWAP tree is built, all the remaining mining processing is based on the PLWAP tree. The original access sequence database is not needed any more, because the size of a PLWAP tree is usually much smaller than that of the original access sequence database. As shown in section 3.2.1, the construction of PLWAP tree is quite efficient by simply scanning the access sequence database twice.

2. An efficient recursive algorithm is proposed to enumerate access patterns from the PLWAP 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 prefix sequence search. Instead of searching common suffix pattern as WAP-tree mining, PLWAP searches for the common prefix pattern \( F \) in WAP-tree, then, recursively extends the frequent pattern by adding the frequent event at the end of the pattern.

3. The essential structure of the PLWAP 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 \( \lambda \times |WASD| \) access sequences of \( WASD \), in which \( |WASD| \) denotes the number of access sequences in \( WASD \) and \( \lambda \) denotes the minimum support threshold. In the next scan, PLWAP builds a prefix tree data structure, which is inherited from WAP-tree mining, using frequent events, to register all count and position information for further mining. Then, PLWAP algorithm recursively mines the WAP-tree using a prefix conditional sequence search to find all web access patterns. An overview of the algorithm is given in Algorithm 1.

Algorithm 1 (PLWAP-mine: Pre-order Linked WAP-tree Mining)

**Input:** web access sequence database \( WASD \) and support threshold \( \lambda \) (0 < \( \lambda \) ≤ 1).

**Output:** the complete set of \( \lambda \)-pattern in \( WASD \)

**Method:**
1. Scan \( WASD \) once, find all frequent individual events.
2. Scan \( WASD \) again, construct a pre-order linked WAP-tree over the set of individual frequent events using algorithm 2 (presented in section 3.2.1).
3. Recursively mine the pre-order linked WAP-tree using common prefix pattern search (presented in section 3.2.2).

*Figure 3.2 Main PLWAP Algorithm*

The algorithm is divided into 3 main parts with 3 steps in algorithm 1 respectively. The main difference between WAP-tree mining and PLWAP is in step 3, where PLWAP only uses the original WAP-tree for mining, and the WAP-tree mining reconstructs the conditional WAP-trees. The key points of PLWAP algorithm are in steps 2 and 3. We
will focus on these two steps in following section.

3.2.1 Construction of Pre-order Linked WAP-tree

We follow the main idea from WAP-tree to construct the PLWAP tree [PHM+00]. There are several points to be considered when constructing the web access pattern tree.

1. Each node in a PLWAP tree registers three pieces of information: label, count and position code, denoted as label: count: position. The root of the tree is a special virtual node with an empty label, position code NULL and count 0. Every other node is labeled by an event in the event set E. The attribute count registers the number of occurrences of the corresponding prefix ended with that event in the Web access sequence database. The position code indicates the position of a node in the WAP tree. It is composed of a series of binary codes as explained in section 3.1. The position code of the root is set to NULL, and the position code of its leftmost child is set to "1".

2. The PLWAP tree is constructed as follows: Using the frequent individual event set obtained from step 1 of PLWAP algorithm, for each access sequence in the database, filter out any non-frequent events, and then insert the resulting frequent subsequence into PLWAP tree. The insertion of frequent subsequences is started from the root of PLWAP tree. Considering the first event, denoted as e, increment the count of the child node with label e by 1 if there exists one; otherwise create a child labeled by e and set the count to 1, and set its position code using the rule as follows: If the node is the leftmost child of its father, its position code is its father's position code attaching 1 at the end, otherwise, its position code is its closest left sibling's position code attaching 0 at the end. Then, recursively insert the rest of the frequent subsequence to the subtree rooted at that child labeled e.
Algorithm 2 (Constructing PLWAP Tree)

Input: A Web access sequence database WASD and the frequent 1-event L (Which is obtained by scanning WASD once and stored into the header linkage table).

Local Variable: NodeFound

Output: a WAP-tree T

Method:

1. Create a root node of T. The position code of the root is set to NULL, the count set to 0.

2. For each access sequence s in the sequence database WASD do
   
   (1) Extract frequent subsequence s' from s by removing all events appearing in s but not in L. Let s' = s1s2... sn, where s(i)(1 ≤ i ≤ n) are events in s'. Let current_node point to the leftmost child of root in tree T.

   (2) For i = 1 to n (length of sequence) do
       
       (a) Check whether there exists a child of current_node labeled si. Check whether there exists a child of current_node labeled si.

           If current_node is NULL, then create a new child node (si; 1) with the position code of current_node.father.positionCode attaching "1" at the end, make the current_node point to the newly created node.

           Else if current_node labeled si, then set NodeFound to true.

           Else let current_node point to current_node.sibling, and keep checking the whether current_node is labeled si, until there is no sibling or si is found.

       (b) If NodeFound

           Then increase the count of si by 1 and make current_node point to si.

           Else create a new child node (si; 1) with the position code of current_node.positionCode attaching "0" at the end, make the current_node point to the newly created node.

3. From root, do depth first search, add all node to the correspond linkage.

4. Return (T) with linkage header table L

Figure 3.3 Constructing PLWAP Tree Algorithm
3. Auxiliary node linkage structures are constructed to assist node traversal in a PLWAP tree as follows. All the nodes in the tree with the same label are linked by shared-label linkages into a queue, called an event-node queue. The event-node queue with label $e_i$ is called the $e_i$-queue. There is one linkage header table $L$ for a WAP-tree, and the header of each event-node queue is registered. The linkage registered into the queue is collected using the pre-order search. The pre-order traversal is stated as follows: Visit the root, visit its left child, and visit its right child. So, we can always check the root of the tree first, insert it into the queue with same label, then recursively mine its left child and right child. Using this method, the node has same label on the same subtree always linked closely. Thus, following the linkage, we can easily identify whether the nodes belong to same subtree or not.

**Example:** We use the same web access sequence database as used in section 2.3 and shown in figure 3.4. The minimum support threshold is also set to 75%, which is 3 out of 4 records in our example. The construction of PLWAP tree process can be described as follows:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Web Access Sequence</th>
<th>Frequent subsequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>abdac</td>
<td>Abac</td>
</tr>
<tr>
<td>200</td>
<td>eaebcac</td>
<td>Abcac</td>
</tr>
<tr>
<td>300</td>
<td>babfaec</td>
<td>Babac</td>
</tr>
<tr>
<td>400</td>
<td>afbacec</td>
<td>abacc</td>
</tr>
</tbody>
</table>

*Figure 3.4 Database of Web Access Sequences*

(1) Insert the sequence $abac$ into the initial tree as shown in figure 3.5(a). It creates a new node $(a:1:1)$ (labeled as a, count as 1, position code as 1) as the child of the root. Then, insert event $b$ into tree. Since there is no child for node $a$, a new node labeled $b$ with count 1 will be created as its left child. Since the node $b$ is first child of $a$, the position code is position code of $a$ ($l$) attaching 1 which is 11. The node $b$ will be presented as $(b:1:11)$. Following the same rule, $(a:1:111)$ and $(c:1:1111)$ are generated.

(2) Insert the sequence $abcac$ from root (figure 3.5(b)). Since the root has a child labeled $a$, thus $a$'s count is increased by 1. Now the node $(a:1:1)$ changed to be $(a:2:1)$. Similarly, we have $(b:2:11)$. The next event $c$ does not match the existing
node $a$. Thus, a new node is created. The new node is not the first child of its father, thus, its position code is position code of its nearest left sibling ($a$:1:111) with 0 attached. Then, the position code of new node is 1110. Finally, the new node can be represented as ($c$:1:1110). Next two events are newly created as ($a$:1:11101) and ($c$:1:111011) respectively.

(3) The process keeps running until there is no more access sequence in WASD. Figure 3.5(c) and 3.5(d) show the WAP-tree after sequences $babac$ and $abacc$ have been inserted into the tree.

(4) Then, we link the nodes with same label together. We start from root using preorder traversal mechanism to create the queue. First, we added ($a$:3:1) into the $a$-queue. Then visit its left child, ($b$:3:11). And ($b$:3:11) will be added to $b$-queue. After checking ($b$:3:11) left child, we found ($a$:2:111) is labeled as $a$, then add it to the $a$-queue. Then ($c$:2:1111) and ($c$:1:11111) will be inserted into the $c$-queue. Since there is no left child after ($c$:1:11111), we go backward, until we find the sibling of ($a$:2:111). Thus we insert ($c$:1:1110) into $c$-queue. The algorithm keeps running until the whole WAP-tree has been traversed.
Figure 3.5 Constructing PLWAP Tree Using Pre-order Traversal

The construction of the PLWAP algorithm is almost the same as the WAP-tree mining. In PLWAP, we introduce the position code, and store the auxiliary node linkage in pre-order traversal mode.

The reason why we store the auxiliary node linkage in the pre-order traversal mode is that the linkage stored in pre-order traversal mode can always put the nodes with same
label in same suffix tree close together. Thus, when we follow the $e_i$ linkage, we can always find the $e_i$ node in the most upper PLWAP tree first. Since the linkage is stored as pre-order mode, the $e_i$ nodes in its suffix tree are always in its following linkage, and it is impossible that the node in its suffix tree appears in linkage before itself. Thus, we can go through the linkage without going backward. Using this method, we can easily follow the linkage to find another $e_i$ node which is not in its suffix tree.

By following the auxiliary $e_i$ node linkage, we store the first frequent event $e_i$ into a variable $S$. Then, we keep checking the node in $S$ with the nodes along the linkage. If the node is its descendant, we will ignore it. Otherwise, we know that we have found another suffix tree of $e_i$. Then, we replace the node in $S$ with the new $e_i$ node which is the root node in another suffix tree. And we keep checking the nodes in linkage behind it. Since we use the pre-order mode to store the linkage, after we store the node in $S$, that means we have found another suffix subtree of $e_i$, and we have no chance to meet a node that belongs to the previous suffix subtree of $e_i$.

From the PLWAP tree, we find the following properties.

**Property 3:** The count of the frequent node $e_i$ in PLWAP tree is the sum of counts of the suffix subtree for $e_i$.

From this property, we know that, the count of any node in $e_i$ suffix subtree is less or equal to the count of $e_i$. Suppose we have a node in the $e_i$ suffix subtree also labeled $e_i$. Which count of $e_i$ will be added to total count? In our solution, we use the prefix sequence to find the frequent sequence. That means, we find the frequent sequence from the beginning. Thus, first $e_i$'s count contributes the count to the total count of $e_i$. The count of any $e_i$ in its suffix subtree will be ignored. Suppose we find a node labeled $e_i$ is an event in a pattern. We still have a chance to find another node labeled $e_i$ in the first $e_i$ suffix subtree. For example, in figure 3.5(e), node $a:2:111$ is the descendant of node $a:3:1$. When we find the frequent 1-sequence, the count of $a:3:1$ will be added to the total. The count of $a:2:111$ will be ignored. Since the node $a:2:111$ belongs to the suffix tree of node $a:3:1$, the count of $a:2:111$ can be contribute to the second event of 2-sequence $aa$.

Access sequences with the same prefix share some upper part of the path from the root. Statistically, considering the factor of prefix sharing, the size of PLWAP tree is
much smaller than the size of access sequence database.

**Property 4:** For any access sequence in an access sequence database \( W ASD \), there exists a unique path in the PLWAP tree starting from the root such that all labels of nodes in the path in order are exactly the same as the events in the sequence.

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

### 3.2.2 Mining Web Access Patterns

The PLWAP tree structure constructed by algorithm 2 provides a way to obtain an interesting sequence. The mining process is based on the PLWAP tree, and no longer needs to read the information from the original sequence database \( W ASD \). Unlike WAP-tree mining, which searches access patterns with the same suffix, the PLWAP is going to find the access sequences with the same prefix.

**Property 5:** For any frequent event \( e_i \), all the frequent subsequences containing \( e_i \) can be visited by following the \( e_i \)-sequence, starting from the record for \( e_i \) in the linkage header table of a WAP-tree.

Property 5 provides a way to access all the pattern information related to the frequent event \( e_i \) by following all the branches in PLWAP tree linked by the \( e_i \)-queue only once. As introduced in section 3.2.1, the frequent nodes with same event \( e_i \) in PLWAP tree are pre-order linked. For example, in figure 3.5(e), the linkage \( a \)-queue is linked as \( a:3:1 \rightarrow a:2:111 \rightarrow a:1:1101 \rightarrow a:1:101 \rightarrow a:1:10111 \). When we try to check if the event \( a \) is frequent or not in any subsequence, we only need to visit the \( a \)-queue to check if all nodes in this queue meet the requirement or not.

The PLWAP tree \( T \), the frequent 1-sequence, and threshold \( \lambda \) can be obtained either
from previous step or be user defined. These variables will be used for every round of the algorithm and thus will be considered to be global variables. The other variables we need to know in order to process the algorithm are the suffix subtree roots set, and the frequent sequences gained from previous round. When the algorithm 3 was called the first time, the suffix subtree roots set is the root of the whole PLWAP tree. While the algorithm keeps running in a particular round, the algorithm is called by the previous round, which found that $e_i$ is a frequent event in the sequence. Thus, it has the suffix subtree roots set, which is a group of suffix subtrees of $e_i$ node in PLWAP tree. Also, the frequent sequence $F$ is the sequence, which is ended with $e_i$ in previous round.

**Algorithm 3 (Mining PLWAP Tree)**

**Input:** Global Variable: PLWAP tree $T$, linkage header table $L$, threshold $\lambda$

Called In Variable: suffix tree roots set $R$, Frequent n-sequence $F$

(R include root, and $F$ is empty when the algorithm is called first time)

Local Variable: $S$ stores the information of node whether it is the ancestor of the following nodes in the queue.

$C$ stores the total number of event $e_i$ in different suffix tree

**Output:** Frequent (n+1)-sequence $F'$

**Method:**
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. $e_i$ suffix tree
   (a) Save first event in $e_i$-queue to $S$
   (b) Following the $e_i$-queue

   If event $e_i$ is the descendant of any event in R, and is not the descendant of $S$
   Insert it into suffix-tree-header set $R'$,
   Add occurrence of $e_i$ into $C$.
   Replace the $S$ with $e_i$
   (c) If $C$ is greater than $\lambda$,
   Attach the $e_i$ after $F$ to form $F'$ and output $F'$
   Call algorithm 3, Passing the $R'$ and $F'$.

*Figure 3.6 Mining PLWAP Tree Algorithm*
The algorithm starts by finding the frequent sequence with the frequent 1-sequence in $L$. For every frequent event in $L$ and the suffix trees of current conditional PLWAP tree being mined, following the linkage of this event, find the first occurrence of the frequent event in every suffix tree of current conditional PLWAP tree being mined, add the support count of all first occurrence of all suffix trees. If the count is greater than the minimum support threshold, then this event is added (concatenated) to the last list of frequent sequences, $F$. The suffix trees of these first occurrences events in the previous mined conditional PLWAP trees are now being used for mining the next event.

Please note, the conditional PLWAP tree which is obtained during the mining process does not actually exist beside the original PLWAP tree. To obtain this conditional PLWAP tree, we only need to remember the roots of suffix trees. In our solution, we use a roots set to store these roots for next round mining. For example, in figure 3.7(b), we can store \{b:3:11, b:1:1011\} as the roots set of conditional PLWAP tree of two nodes, a:3:1 and a:1:101. When we want to mine the events in the suffix tree of a:3:1 and a:1:101, we only need to check the nodes which are descendants of the roots set \{b:3:11, b:1:1011\} using the method of position codes described in section 3.1.

Having the idea described above, we now can mine the PLWAP tree by recursively finding the frequent events in the conditional PLWAP tree. For example, in figure 3.7(a), the PLWAP tree for the example database of figure 3.4 is given with pre-order header linkage. In order to mine this PLWAP tree, we start following the header linkage of the first frequent event 'a'. Since the two suffix trees of PLWAP tree rooted at a:3:1 and b:1:10 respectively have the first occurrences of 'a' node with total support of 4 from a:3:1 and a:1:101. Because the minimum support is 3, then we have 'a' as a frequent event to be added to the last list of frequent sequence (Ø). Next, we continue to mine all frequent events in the suffix trees of a:3:1 and a:1:101, which are rooted at b:3:11 and b:1:1011 respectively (shown in figure 3.7(b) with un-shadowed nodes). From figure 3.7(b), we keep finding the first occurrence of 'a' for each suffix tree. Then, we find a:2:111, a:1:1101 and a:1:10111 to give $a$ as a frequent event. Thus, $a$ is added to the last list of frequent sequence 'a', to form the new frequent sequence 'aa'. We continue to mine the conditional PLWAP tree in figure 3.7(c). The suffix trees of these $a$ nodes,
which are rooted at c:2:1111, c:1:11011 and c:1:101111, give another c frequent to obtain the sequence 'aac'. The last suffix tree (in figure 3.7(d)) rooted at c:1:11111 is no longer frequent. And this terminates this leg of recursive search. Backtracking in the order of the previous conditional PLWAP tree mined, we search for other frequent events. Since no more frequent events are found in the conditional PLWAP tree in figure 3.7(c), we backtrack to figure 3.7(b) and we find that b:3:11, b:1:1011 yield frequent event for b to give the next frequent sequence as ab (in figure 3.8(a)).
Figure 3.7 Mining PLWAP Tree - Find Frequent Sequence Starting with aa
Figure 3.8 Mining PLWAP Tree- Find Frequent Sequence Starting with ab or ac
The algorithm keeps running. The whole process of finding the frequent sequences can be seen in figure 3.7 through 3.9. From figure 3.7, we can find all frequent sequence \{aa, aac\} starting with aa. Then, since the roots set is empty, we backtrack to find the frequent sequences starting with ab in figure 3.8(a) to 3.8(d) and find the frequent sequence set \{ab, aba, abac, abc\}. From figure 3.8(e), we can find the frequent sequence \{ac\}. Since all frequent sequences starting with a have been found, we are going to find the frequent sequence starting with b or c in figure 3.9. Finally, combining the frequent sequence found in figure 3.7 through 3.9, we have the frequent sequence set \{a, aa, aac, ab, aba, abac, abc, ac, b, ba, bac, bc, c\}. After checking with the final set of frequent sequences, we find it is the same the result extracted using WAP-tree mining.
Figure 3.9 Mining PLWAP Tree - Find Frequent Sequence Starting with b or c
3.3 Algorithm Analysis

After reviewing the algorithm introduced in the previous section, we find PLWAP algorithm has several advantages. First of all, it does not need to construct the WAP tree recursively, all mining process is handled with only the original WAP-tree. The only costly part is comparison of the position code when checking the relationship between nodes. Secondly, the PLWAP inherits the advantages of WAP-tree mining, the whole process is based on the WAP-tree which is compact. The WAP-tree is more likely to fit in memory, thus, reducing the I/O cost.

Comparing PLWAP with WAP-tree mining considering the CPU time cost and memory cost view. From the time cost view, the PLWAP algorithm is much more efficient than WAP-tree mining. In WAP-tree mining, when the sequence extends one event from the previous round, it should re-construct the WAP-tree again. Also, please note that, when the algorithm tries to construct the WAP-tree based on the sequence obtained in previous round, it also needs to be compared to combine, where it should deduce the same sequence from same prefix subsequence. The time cost on this part is the most costly part in the algorithm. On the other hand, in order to avoid repeatedly checking the whole \( e_i \) linkage in mining process, PLWAP involves a pre-order traversal for whole WAP-tree in algorithm 2. After this traversal, the \( e_i \) in the same suffix tree will be stored closely in the linkage. We compare the time cost of two algorithms using the example demonstrated in previous section.

<table>
<thead>
<tr>
<th></th>
<th>WAP-tree</th>
<th>PLWAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan Database</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Traversal WAP-tree for linkage</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Construct WAP tree</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Save and Scan intermediate database</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Calculating frequent event in conditional pattern</td>
<td>10</td>
<td>/</td>
</tr>
<tr>
<td>Construct intermediate WAP tree</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Check the nodes whether belong to the suffix tree of roots set</td>
<td>/</td>
<td>yes</td>
</tr>
</tbody>
</table>

*Figure 3.10 CPU Time Comparison between PLWAP and WAP-tree Mining*
In general, the main difference between these two algorithms is that the WAP-tree mining algorithm saves conditional patterns to disk and then reads these patterns from disk to construct the intermediate WAP tree, while the PLWAP needs to compare the position code of nodes along the linkage queue with that of nodes in the roots set. Suppose the number of sequence patterns that can be extracted from the WAP-tree is \( s = |\text{sequence}| \). The time of constructing the intermediate WAP-tree is \( O(s) \).

From the memory view point, we find that the WAP-tree uses less memory than PLWAP at the beginning of the algorithm. But while WAP-tree needs to reconstruct WAP-tree, it needs more space. We can estimate how much space is needed in the WAP-tree mining. If the space exceeds the memory, the time used on I/O will increase a lot. The reason why PLWAP uses more space at the beginning of the algorithm is that PLWAP needs to store position codes for every event node in WAP-tree. The position code is based on the binary tree transformation of WAP-tree. Its length is the height of the binary tree. And the height of binary tree is height plus width of WAP-tree. The height of WAP-tree is the length of longest frequent subsequence in web access sequence database. As we introduced in chapter 1, the web access sequence is a sequence of page-views which is produced by the user visiting the web pages. Normally, its length is shorter. But the width of the WAP-tree is extremely big. For every node in the WAP-tree, we need 1 byte to store the label, 1 byte to store the count, 2 bytes store to the links, which indicate its child and its sibling. In the PLWAP algorithm, the extra bytes used to store position code vary. It depends on the average of the binary tree height. We can use the example shown in the previous chapter to estimate the memory requirement for PLWAP and WAP-tree mining.
<table>
<thead>
<tr>
<th>Store original WAP-tree (13 nodes)</th>
<th>WAP-tree</th>
<th>PLWAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>Count</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>Position code</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>Links</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>linkage</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>Store intermediate WAP-tree (in deepest recursive round)</td>
<td>52</td>
<td>0</td>
</tr>
<tr>
<td>Total (bytes)</td>
<td>104</td>
<td>65</td>
</tr>
</tbody>
</table>

*Figure 3.11 Memory Comparison between PLWAP and WAP-tree Mining*

We can find that the deepest the recursive round the WAP-tree algorithm calls, the more space it needs. In the worst case, it may need several times the space of the original WAP-tree. When the original WAP-tree is large and the sequence we try to find is long, WAP-tree may face a problem to store all information into the main memory. Some I/O work may need to exchange between the main memory and virtual memory. For PLWAP mining, at the beginning of mining process, the extra space we need is to store the position code, which can be stored as bit string. During the mining process, the PLWAP may need some space to store the roots set. But the PLWAP only needs to store the pointer of nodes in the roots set. It takes a small amount of space to store this information.
Chapter 4

Experimental Evaluation and Performance Analysis

In this chapter we compare the performance of PLWAP with the GSP, and WAP-tree mining algorithms. The PLWAP algorithm is implemented as described in chapter 3. All experiments are performed on 400MHz Celeron PC machine with 64 megabytes memory. The operating system is Windows 98. All algorithms are written in C++ language and running under Inprise C++ Builder environment.

4.1 Dataset

Our synthetic datasets are generated using the publicly available synthetic data generation program of the IBM Quest data mining project at http://www.almaden.ibm.com/cs/quest/, which has been used in most sequential pattern mining studies [SA96, MCP98, PHM+00]. The datasets consist of sequences of events, where each event represents a web page for web accessing.

The parameters shown below are used to generate the data sets.

|D|: Number of sequences in database
|C|: Average length of the sequences
|S|: Average length of maximal potentially frequent sequence
|N|: number of events

For example, C10.S5.N2000.D60k means that |C| = 10, |S| = 5, |N|= 2000, and |D| = 60k. It represents a group of data that the average length of the sequences is 10, the average length of maximal potentially frequent sequence is 5, the events in the database are 2000, and the total number of sequences in database is 60k. The datasets with different parameters test different aspects of the algorithms. Basically, if the number of these four parameters becomes larger, the execution time becomes longer.
4.2 Experiment 1: Execution Time for Different Support

This experiment uses fixed size database and different minimum support to compare the performance of PLWAP, WAP and GSP. The datasets is described as C10.S5.N2000.D60k. We test algorithms with the minimum support from 0.8% to 10% against the 60k database.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Runtime (in seconds) at different support(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.8</td>
</tr>
<tr>
<td>GSP</td>
<td>—</td>
</tr>
<tr>
<td>WAP</td>
<td>2080</td>
</tr>
<tr>
<td>PLWAP</td>
<td>492</td>
</tr>
</tbody>
</table>

*Figure 4.1 Execution Time (seconds) with Different Minimum Support*

![Graph showing execution time trend with different minimum support](image)

*Figure 4.2 Execution Time Trend with Different Minimum Support*

From figure 4.1 and 4.2, we can find that the execution time of every algorithm...
decreases while the minimum support increases. The idea is obvious, when the minimum support increases, the number of candidate sequence decreases. Thus, the algorithms need less time to find the frequent sequences.

Among these three algorithms, the GSP algorithm is the most time-consuming algorithm. It is hard to generate and test the candidate sequences when the number of frequent 1-sequence becomes large. Suppose the frequent 1-sequence is 9, and the candidate 2-sequences will be 81, the testing work against the 60k database is huge. As shown in figure 4.2, the execution time of GSP increases sharply while the minimum support decreases. The GSP algorithm faced a problem when the minimum support is set to the 4%. It took too much time.

As introduced in previous chapter, the WAP-tree and PLWAP algorithms are both based on a prefix tree. The PLWAP algorithm always uses less runtime than the WAP mining. WAP tree mining needs I/O process, which is much more costly process than the other processes in the algorithm. While the minimum support threshold decreases, the number of events, which meet the minimum support, increases. This means the WAP-tree becomes larger and longer, and the algorithm needs much more I/O work during the process of WAP-tree mining algorithm. As the minimum support decreases, the execution time difference between WAP-tree and PLWAP increases.

4.3 Experiment 2: Execution Time for Databases with Different Size

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Runtime (in seconds) at different data size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20k</td>
</tr>
<tr>
<td>WAP</td>
<td>158</td>
</tr>
<tr>
<td>PLWAP</td>
<td>54</td>
</tr>
</tbody>
</table>

*Figure 4.3 Execution Time (seconds) with Different Data Size*

*Figure 4.4 Execution Time Trend with Different Data Size*

When the data size increases, the cost for scanning the original database twice and building original WAP-tree increases. These costs occur in both algorithms. The reason why execution time for WAP-tree increases more than that of PLWAP algorithm is similar to the idea behind experiment 1. While the data size increases, the WAP-tree becomes large, it needs more I/O work to store the intermediate patterns and construct intermediate WAP-trees.
4.4 Experiment 3: Execution Time for Different Length Sequences

In experiment 3, we check the efficiency of WAP and PLWAP for the sequences with different lengths. Three datasets are used to do the comparison: C10.S5.N1500.D10k, C20.S8.N1500.D10k and C30.S10.N1500.D10k. The minimum support is set to 1%.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Runtime (in seconds) with different length sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10(5)</td>
</tr>
<tr>
<td>WAP</td>
<td>402</td>
</tr>
<tr>
<td>PLWAP</td>
<td>217</td>
</tr>
</tbody>
</table>

*Figure 4.5 Execution Time (seconds) with Different Length Sequences*

![Execution Time Trend with Different Length Sequences](image)

*Figure 4.6 Execution Time Trend with Different Length Sequences*

From the result shown in figure 4.5 and 4.6, we can clearly find PLWAP algorithm is much more efficient than WAP algorithm. When the sequence becomes longer, the
WAP algorithm needs more time to construct the intermediate WAP-tree.

4.5 Correctness of Algorithm Implementations

To show that the implementations of three algorithms are correct, a small sample database, which is used in chapter 3, is used to test the correctness of implementations. As shown in figure 4.7, the database includes 4 sequences with a total of 26 events in it. The minimum support of the database is set to 0.75 or 75%.

<table>
<thead>
<tr>
<th>SID</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>10, 20, 40, 10, 30</td>
</tr>
<tr>
<td>200</td>
<td>50, 10, 50, 20, 30, 10, 30</td>
</tr>
<tr>
<td>300</td>
<td>20, 10, 20, 60, 10, 50, 30</td>
</tr>
<tr>
<td>400</td>
<td>10, 60, 20, 10, 30, 60, 30</td>
</tr>
</tbody>
</table>

*Figure 4.7 Sample Datasets Testing the Correctness*

After running three algorithms against the database in figure 4.7, the results of frequent sequence are shown in figure 4.8. By comparing with results produced from the theoretic work in chapter 3, the results of these three algorithms are the same. The only difference between these results is the order of sequences produced from the algorithms. Having these results, we can say the experiments conducted in previous 3 sections are based on correct algorithm implementations.
<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Frequent Sequence Results</th>
</tr>
</thead>
</table>
| **GSP**   | {{10}, {20}, {30},
|          | {10,10}, {10,20}, {10,30}, {20,10}, {20,30},
|          | {10,10,30}, {10,20,10}, {10,20,30}, {20,10,30},
|          | {10,20,10,30}}          |
| **WAP**   | {{10,20,30}, {20,30}, {10,20,10,30}, {20,10,30}, {10,10,30}, {10,30}, {30},
|          | {10,20}, {20},
|          | {10,20,10}, {20,10}, {10,10}, {10}} |
| **PLWAP** | {{10}, {10,10}, {10,10,30}, {10,20}, {10,20,10}, {10,20,10,30}, {10,20,30},
|          | {10,30},
|          | {20}, {20,10}, {20,10,30}, {20,30},
|          | {30}} |

Figure 4.8 Results Generating from Three Algorithms to Show the Correctness
Chapter 5

Conclusions and Future Work

5.1 Conclusions

This thesis presents a new algorithm (PLWAP) for efficiently mining sequential patterns from the web log. The PLWAP algorithm adapts the WAP-tree structure to store the frequent patterns. In this thesis, we present to find common prefix patterns instead of suffix patterns in WAP-tree mining. In order to avoid recursively re-constructing WAP-tree, the pre-order linkage and position codes are proposed. While the pre-order linkage provides a way to traverse the event queue without going backwards, the position code is used to identify the position of nodes in WAP-tree. After using these two methods, we find the frequent event in the suffix tree without traversing the whole WAP-tree. Thus, it avoids constructing WAP-tree recursively. The experiments show that mining web log using PLWAP algorithm is more efficient than WAP-tree and GSP algorithm, especially when the frequent sequence becomes longer and the original database becomes larger.

5.2 Future Work

For sequential patterns mining, the following aspects can be considered as future work.

- As the synthetic datasets used in our experiment, it is no need to transform the original web log to the sequential database. In web usage mining application, the procedure for transforming the web log to sequential database is one of time-consuming parts. Thus, it needs a lot of consideration when doing web usage mining with real web logs.
- It may need some modification when applying PLWAP algorithm to sequential pattern mining in large databases other than web log. For mining the sequential pattern in transaction database, we need to consider of the concurrence of events.

- When we do web usage mining, in most of time, it may relate to the content of web pages. It will be great improvement to consider the web page content when doing web usage mining.
Reference


70


[MPT00] F. Masseglia, P. Poncelet and M. Teisseire, "Incremental Mining of Sequential Patterns in Large Databases", Actes des 16iem Journes Bases de Donnes Avances (BDA'00), Blois, France, October 2000.


[PHM+00] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. "Mining Access Patterns Efficiently from Web Logs", Proc. 2000 Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD'00), pages 396-407, Kyoto, Japan, April 2000.


Appendix: Correctness Testing Results

1. PLWAP algorithm

G:\thesis\PLWAP> plwap >> p.result

Now start the program......

Scan the database first time: 1
Finish scanning the database: 1
Begin to build tree 1
End of building tree and begin to build linkage...
End of building linkage...

Print the WAP tree
event:-1 occurrence = 4. the son is 10
event:10 occurrence = 3. the son is 20.... its parent is -1
event:20 occurrence = 3. the son is 10.... its parent is 10
event:10 occurrence = 2. the son is 30.... its parent is 20
event:30 occurrence = 2. the son is 30.... its parent is 10
event:10 the sibling is 30; the occurrence are 1
event:30 occurrence = 1. the son is 10.... its parent is 20
event:10 occurrence = 1. the son is 30.... its parent is 30
event:10 the sibling is 20; the occurrence are 1
event:20 occurrence = 1. the son is 10.... its parent is -1
event:10 occurrence = 1. the son is 20.... its parent is 20
event:20 occurrence = 1. the son is 10.... its parent is 10
event:10 occurrence = 1. the son is 30.... its parent is 20

74
End of printing WAP tree and begin printing linkage
10--->10(3,1,2147483648)--->10(2,3,3758096384)--->10(1,5,3892314112)-->
>10(1,3,2684354560)--->10(1,5,3087007744)

20--->20(3,2,3221225472)--->20(1,2,2147483648)--->20(1,4,2952790016)

30--->30(2,4,4026531840)--->30(1,5,4160749568)--->30(1,4,3758096384)-->
>30(1,6,3959422976)--->30(1,6,3154116608)

Begin the mining process

Entering the mining process 1 times
10

Entering the mining process 2 times
10 10

Entering the mining process 3 times
10 10 30
10 20

Entering the mining process 4 times
10 20 10

Entering the mining process 5 times
10 20 10 30
10 20 30
10 30
20
Entering the mining process 6 times
20 10

Entering the mining process 7 times
20 10 30
20 30
30

begin time: 1023713478
deadline : 1023713478
The execution time is: 0

End the program
2. WAP algorithm

G:\thesis\WAP> wap >> w.result

Now start the program......

Scan the Original database....
Finish scanning the database and Begin to build the tree ...
End of building tree and linkage...
Print the WAP tree
event:-1 occurrence = -1. the son is 10
event:10 occurrence = 3. the son is 20.... its parent is -1
event:20 occurrence = 3. the son is 10.... its parent is 10
event:10 occurrence = 2. the son is 30.... its parent is 20
event:30 occurrence = 2. the son is 30.... its parent is 10
event:10 the sibling is 30; the occurrence are 1
event:30 occurrence = 1. the son is 10.... its parent is 20
event:10 occurrence = 1. the son is 30.... its parent is 30
event:10 the sibling is 20; the occurrence are 1
event:20 occurrence = 1. the son is 10.... its parent is -1
event:10 occurrence = 1. the son is 20.... its parent is 20
event:20 occurrence = 1. the son is 10.... its parent is 10
event:10 occurrence = 1. the son is 30.... its parent is 20

End of printing WAP tree and begin printing linkage

10-->10(3)-->10(2)--10(1)--10(1)--10(1)
20-->20(3)--20(1)--20(1)
30-->30(2)--30(1)--30(1)--30(1)--30(1)
Begin the mining process: 0

Scan the intermediate database: 1
Finish scanning the database and Begin to build the tree ...
End of building tree and linkage...
Print the WAP tree
  event:-1 occurrence = -1. the son is 10
  event:10 occurrence = 3. the son is 20.... its parent is -1
  event:20 occurrence = 3. the son is 10.... its parent is 10
  event:10 the sibling is 20; the occurrence are 1
  event:20 occurrence = 1. the son is 10.... its parent is -1
  event:10 occurrence = 1. the son is 20.... its parent is 20
  event:20 occurrence = 1. the son is 10.... its parent is 10

End of printing WAP tree and begin printing linkage

10--->10(3)--->10(3)--->10(1)--->10(1)
20--->20(3)--->20(1)--->20(1)

Begin the mining process: 1

Scan the intermediate database: 2
Finish scanning the database and Begin to build the tree ...
End of building tree and linkage...
Print the WAP tree
  event:-1 occurrence = -1. the son is 10

End of printing WAP tree and begin printing linkage
10→10(4)

Begin the mining process: 2

Here is the result......
20 30

Scan the intermediate database: 3
Finish scanning the database and Begin to build the tree ...
End of building tree and linkage...
Print the WAP tree
event:-1 occurrence = -1. the son is 10
event:10 occurrence = 3. the son is 20.... its parent is -1
event:10 the sibling is 20; the occurrence are 1
event:20 occurrence = 1. the son is 10.... its parent is -1
event:10 occurrence = 1. the son is 20.... its parent is 20

End of printing WAP tree and begin printing linkage

10→10(3)→10(1)
20→20(3)→20(1)→20(1)

Begin the mining process: 3

Scan the intermediate database: 4
Finish scanning the database and Begin to build the tree ...
End of building tree and linkage...
Print the WAP tree
event: 1 occurrence = -1. the son is 10

End of printing WAP tree and begin printing linkage

10 --> 10(4)

Begin the mining process: 4

Here is the result......
20 10 30

Scan the intermediate database: 5
Finish scanning the database and Begin to build the tree ...
End of building tree and linkage...
Print the WAP tree

End of printing WAP tree and begin printing linkage

Begin the mining process: 5

Here is the result......
10 10 30

Here is the result......
10 30

Here is the result......
30

80
Scan the intermediate database: 6
Finish scanning the database and Begin to build the tree ...
End of building tree and linkage...
Print the WAP tree
   event:-1 occurrence = -1. the son is 10

End of printing WAP tree and begin printing linkage

10-->10(4)

Begin the mining process:6

Here is the result......

20

Scan the intermediate database: 7
Finish scanning the database and Begin to build the tree ...
End of building tree and linkage...
Print the WAP tree
   event:-1 occurrence = -1. the son is 10
   event:10 occurrence = 3. the son is 20.... its parent is -1
   event:10 the sibling is 20; the occurrence are 1
   event:20 occurrence = 1. the son is 10.... its parent is -1
   event:10 occurrence = 1. the son is 20.... its parent is 20

End of printing WAP tree and begin printing linkage

10-->10(3)-->10(1)
20-->20(3)-->20(1)-->20(1)
Begin the mining process: 7

Scan the intermediate database: 8
Finish scanning the database and Begin to build the tree ...
End of building tree and linkage...
Print the WAP tree
event:-1 occurrence = -1. the son is 10

End of printing WAP tree and begin printing linkage

10-->10(4)

Begin the mining process: 8

Here is the result......
20 10

Scan the intermediate database: 9
Finish scanning the database and Begin to build the tree ...
End of building tree and linkage...
Print the WAP tree

End of printing WAP tree and begin printing linkage

Begin the mining process: 9
Here is the result......
10 10

Here is the result......
10

Here is the result......

begin time: 1023714962
der time : 1023714962
The execution time is: 0

End the program
3. GSP algorithm

G:\thesis\GSP> gsp >> g.result

Now start the program......

Scan the original database once to find the 1-sequence
Found 1-sequence.....
10
20
30

Generating 2-sequence.....
Testing 2-sequence....
Finish testing 2-sequence and This is result...
10 10
10 20
10 30
20 10
20 30

Generating 3-sequence.....
Testing 3-sequence....
Finish testing 3-sequence and This is result...
10 10 30
10 20 10
10 20 30
20 10 30

84
Generating 4-sequence.....
Testing 4-sequence...
Finish testing 4-sequence. and This is result...
10 20 10 30

Generating 5-sequence.....
Testing 5-sequence...
Finish testing 5-sequence. and This is result...
begin time: 1023716187
end time : 1023716187
The execution time is:
0

End the program
Now start the program.....

Scan the original database once to find the 1-sequence
Found 1-sequence .....  
10  
20  
30  

Generating 2-sequence.....
Testing 2-sequence...
Finish testing 2-sequence. and This is result...
10 10  
10 20  
10 30  
20 10  
20 30
Generating 3-sequence.....
Testing 3-sequence...
Finish testing 3-sequence. and This is result...
10 10 30
10 20 10
10 20 30
20 10 30

Generating 4-sequence.....
Testing 4-sequence...
Finish testing 4-sequence. and This is result...
10 20 10 30

Generating 5-sequence.....
Testing 5-sequence...
Finish testing 5-sequence. and This is result...

begin time: 1023716230
end time : 1023716230
The execution time is:
0

End the program
VITA AUCTORIS

NAME  Yi Lu
YEAR OF BIRTH  1971
PLACE OF BIRTH  Hangzhou, P.R.China
EDUCATION  M.Sc., Computer Science
           University of Windsor
           Windsor, Ontario
           Canada
           2000-2002

           B.Sc., Computer Science
           Zhejiang University
           Hangzhou
           P.R.China
           1989-1993