Web Usage Mining - Languages and Algorithms

John R. Punin, Mukkai S.Krishnamoorthy, Mohammed J.Zaki

Computer Science Department
Rensselaer Polytechnic Institute, Troy NY 12180, USA

Abstract: Web Usage Mining deals with the discovery of interesting information from user navigational patterns from web logs. While extracting simple information from web logs is easy, mining complex structural information is very challenging. Data cleaning and preparation constitute a very significant effort before mining can even be applied. We propose two new XML applications, XGMML and LOGML to help us in this task. XGMML is a graph description language and LOGML is a web-log report description language. We generate a web graph in XGMML format for a web site and generate web-log reports in LOGML format for a web site from web log files and the web graph. We show the simplicity with which mining algorithms can be specified and implemented efficiently using our two XML applications.

1 Introduction

Recently XML has gained wider acceptance in both commercial and research establishments. In this paper, we present two XML languages and a web data mining application which utilizes them to extract complex structural information. Extensible Graph Markup and Modeling Language (XGMML) is an XML application based on Graph Modeling Language (GML) which is used for graph description. XGMML uses tags to describe nodes and edges of a graph. The purpose of XGMML is to make possible the exchange of graphs between different authoring and browsing tools. The conversion of graphs written in GML to XGMML is straight forward. Using Extensible Stylesheet Language (XSL) with XGMML allows the translation of graphs to different formats.

Log Markup Language (LOGML) (Punin & Krishnamoorthy 2000) is an XML application designed to describe log reports of web servers. Web data mining is one of the current hot topics in computer science. Mining data that has been collected from web server logfiles, is not only useful for studying customer choices, but also helps to better organize web pages. This is accomplished by knowing which web pages are most frequently accessed by the web surfers. When mining the data from the log statistics, we use the web graph for annotating the log information. Further we produce summary reports, comprising of information such as client sites, types of browsers and the usage time statistics. We also gather the client activity in a web site as a subgraph of the web site graph. This subgraph can be used to get better understanding of general user activity in the web site. In LOGML, we create a new XML vocabulary to structurally express the contents of the logfile information.

Recently web data mining has been gaining a lot of attention because of its potential commercial benefits. For example, consider a web log database at a popular site, where an object is a web user and an attribute is a web page. The mined patterns could be the sets or sequences of most frequently accessed pages at that site. This kind of information can be used to restructure the web-site, or to dynamically insert relevant links in web pages based on user access patterns. Furthermore, click-stream mining can help E-commerce vendors to target potential online customers in a more effective way, at the same time enabling personalized service to the customers. Web mining is an umbrella term that refers to mainly two distinct tasks. One is web content mining (Cooley, Mobasher, & Srivastava 1997), which deals with problems of automatic information filtering and categorization, intelligent search agents, and personalize web agents. Web usage mining (Cooley, Mobasher, & Srivastava 1997) on the other hand relies on the structure of the site, and concerns itself with discovering interesting information from user navigational behavior as stored in web access logs. The focus of this paper is on web usage mining. While extracting simple information from web logs is easy, mining complex structural information is very challenging. Data cleaning and preparation constitute a very significant effort before mining can even be applied. The relevant data challenges include: elimination of irrelevant information such as image files and cgi scripts, user identification, user session formation, and incorporating temporal windows in the user modeling. After all this pre-processing, one is ready to mine the resulting database.

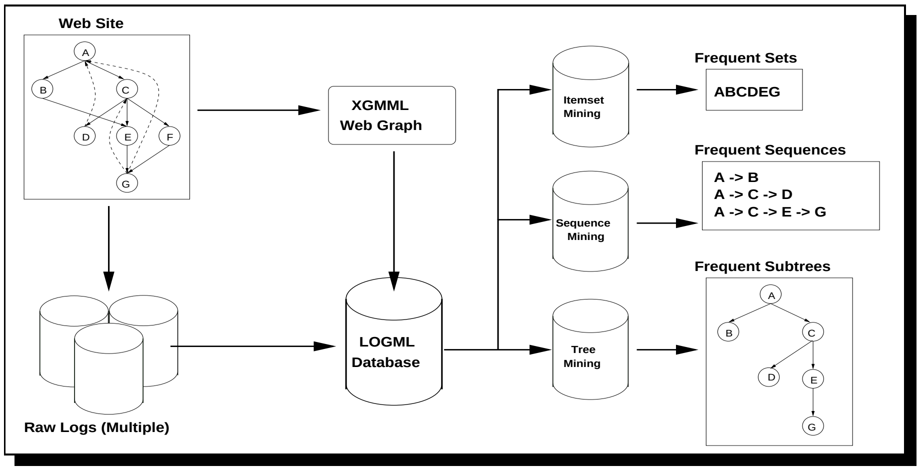


Figure 1: Web Usage Mining Architecture

The proposed LOGML and XGMML languages have been designed to facilitate this web mining process in addition to storing additional summary information extracted from web logs. Using the LOGML generated documents the pre-processing steps of mining are considerably simplified. We also propose a new mining paradigm, called Frequent Pattern Mining, to extract increasingly informative patterns from the LOGML database. Our approach and its application to real log databases are discussed further in Section 5. We provide an example to demonstrate the ease with which information about a web site can be generated using LOGML with style sheets (XSLT). Additional information about web characterization can also be extracted from the mined data.

The overall architecture of our system is shown in Figure 1. The two inputs to our web mining system are 1) web site to be analyzed, and 2) raw log files spanning many days, months, or extended periods of time. The web site is used to populate a XGMML web graph with the help of a web crawler. The raw logs are processed by the LOGML generator and turned into a LOGML database. This processed database contains log information that can be used to mine various kinds of frequent pattern information such as itemsets, sequences and subtrees. The LOGML database and web graph information can also be used for web characterization, providing detailed statistics on top k pages, addresses, browsers, and so on.

It should be noted that association and sequence mining have also been applied to web usage mining in the past. Chen et al. (Chen, Park, & Yu 1996) introduced the notion of a maximal forward chain of web pages and gave an algorithm to mine them. The WUM system (Spiliopoulou & Faulstich 1998) applies sequence mining to analyze the navigational behavior of users in a web site. WUM also supports an integrated environment for log preparation, querying and visualization. Cooley et al. (Cooley, Mobasher, & Srivastava 1999) describe various data preparation schemes for facilitating web mining. Recent advances and more detailed survey on various aspects of web mining spanning content, structure and usage discovery can be found in (Masand & Spiliopoulou 2000; Kosala & Blockeel 2000). Our work differs in that our system uses new XML based languages to streamline the whole web mining process and allows multiple kinds of mining and characterization tasks to be performed with relative ease.

2 XGMML

A graph, G= (V,E), is a set of nodes V and a set of edges E. Each edge is either an ordered (directed graph) or unordered (undirected) pair of nodes. Graphs can be described as data objects whose elements are nodes and edges (which are themselves data objects). XML is an ideal way to represent graphs. Structure of the World Wide Web is a typical example of a graph where the web pages are "nodes," and the hyperlinks are "edges." One of the best ways to describe a web site structure is using a graph structure and hence XGMML documents are a good choice for containing the structural information of a web site. XGMML was created for use within the WWWPal System (Punin & Krishnamoorthy 1998) to visualize web sites as a graph. The web robot of W3C (webbot), a component of the WWWPal System, navigates through web sites and saves the graph information as an XGMML file. XGMML, as any other XML application, can be mixed with other markup languages to describe additional graph, node and/or edge information.

Structure of XGMML Documents: An XGMML document describes a graph structure. The root element is the `graph` element and it can contain `node`, `edge` and `att` elements. The `node` element describes a node of a graph and the `edge` element describes an edge of a graph. Additional information for graphs, nodes and edges can be attached using the `att` element. A `graph` element can be contained in an `att` element and this graph will be considered as subgraph of the main graph. The `graphics` element can be included in a `node` or `edge` element, and it describes the graphic representation either of a node or an edge.

Resource Description Framework (RDF) is one way to describe metadata about resources. XGMML includes metadata information for a graph, node and/or edge using the `att` tag. The example below is part of a graph describing a web site. The nodes represent web pages and the edges represent hyperlinks. The metadata of the web pages is included as attributes of a node. RDF and Dublin Core (DC) vocabularies have been used to describe the metadata of the nodes.

```
<?xml version="1.0"?>
<graph xmlns = "http://www.cs.rpi.edu/XGMML"
       xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
       xsi:schemaLocation="http://www.cs.rpi.edu/XGMML
       http://www.cs.rpi.edu/~puninj/XGMML/xgmml.xsd"
       directed="1" >
<node id="3" label="http://www.cs.rpi.edu/courses/" weight="5427">
<att>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.0/">
  <rdf:Description about="http://www.cs.rpi.edu/courses/"
    dc:title="Courses at Rensselaer Computer Science Department"
    dc:subject="M.S. requirements; Courses; People;
    Graduate Program; Computer  Algorithms; Programming in Java;
    Research; Course Selection
    Guide; Programming in Java; Models  of Computation"
    dc:date="2000-01-31"
    dc:type="Text"
    >
  </rdf:Description>
</rdf:RDF>
</att>
</node>
....
<edge source="1" target="3" weight="0" label="SRC IMG gfx/courses2.jpg" />
<edge source="7" target="3" weight="0" label="SRC IMG ../gfx/courses2.jpg" />
</graph>
```

3 LOGML (Log Markup Language)

Log reports are the compressed version of logfiles. Web masters in general save web server logs in several files. Usually each logfile contains a single day of information. Due to disk space limitation, old log data gets deleted to make room for new log information. Generally, web masters generate HTML reports of the logfiles and do not have problems keeping them for a long period of time as the HTML reports are an insignificant size. If a web master likes to generate reports for a large period of time, he has to combine several HTML reports to produce a final report. LOGML is conceived to make this task easier. Web masters can generate LOGML reports of logfiles and combine them on a regular basis without much effort. LOGML files can be combined with XSLT to produce HTML reports. LOGML offers the flexibility to combine them with other XML applications, to produce graphics of the statistics of the reports. LOGML can also be combined with RDF to provide some metadata information about the web server that is being analyzed. LOGML is based on XGMML. LOGML document can be seen as a snapshot of the web site as the user visits web pages and traverses hyperlinks. It also provides a succinct way to save the user sessions. In the W3C Working Draft "Web Characterization Terminology & Definitions Sheet", the user session is defined as "a delimited set of user clicks across one or more Web servers".

Structure of LOGML Documents: A typical LOGML document has three sections under the root element `logml` element. The first section is a graph that describes the log graph of the visits of the users to web pages and hyperlinks. This section uses XGMML to describe the graph and its root element is the `graph` element. The second section is the additional information of log reports such as top visiting hosts, top user agents, and top keywords. The third section is the report of the user sessions. Each user session is a subgraph of the log graph. The subgraphs are reported as a list of edges that refer to the nodes of the log graph. Each edge of the user sessions also has a timestamp for when the edge was traversed. This timestamp helps to compute the total time of the user session. LOGML files are large files.

LOGML Elements and Attributes: The root element of a LOGML document is the `logml` element. The rest of the elements are classified with respect to the three sections of the LOGML document. The first section is the report of the log graph and we use the XGMML elements to describe this graph. The second section report the general statistics of the web server such as top pages, top referer URLs, top visiting user agents, etc. And, the last section reports the user sessions.

The following global attributes are used by most of the LOGML elements: `id` - unique number to identify the elements of LOGML document. `name` - string to identify the elements of LOGML document. `label` - text representation of the LOGML element `access_count` - number of times the web server has been accessed. For example, the number of times of a specific user agent accessed the web server. `total_count` - total number of times that an element is found in a logfile. For example, the total count of a keyword. `bytes` - number of bytes downloaded. `html_pages` - number of HTML pages requested from the web server. For example, the number of html pages requested by a specific site.

The XGMML elements that we use to describe the log graph are `graph`, `node`, `edge` and `att`. We add the `hits` attribute to the `node` and `edge` elements to report the number of visits to the node (web page) and the number of traversals of the edge (hyperlink). The `att` element is used to report metadata information of the web page such as mime type and size of the file. The elements of the second section include hostname, IP, domains, directories, user agents, referers, host referers, keywords, http code, method and version, summary information like the the total number of requests, user sessions or bytes transferred, and statistics by date of requests.

The third section of the LOGML document reports the user sessions and the LOGML elements are:

• `userSessions, userSession` - The `userSessions` element is the container element for the set of the user sessions. Each user session is described using the `userSession`, `path` and `uedge` elements where a `path` is the collection of hyperlinks that the user has traversed during the session.
• `path` - The `path` element contains all hyperlinks that the user has traversed during the user session.
• `uedge` - The `uedge` element reports a hyperlink that has been traversed during the user session. The `source` and the `target` attributes are reference to nodes of the Log Graph in the first section and the `utime` attribute is the timestamp where the user traversed this hyperlink. Example below is the report of one user session in a LOGML document:

```
<userSession name="proxy.artech.com.uy" ureferer="No referer"
entry_page="www.cs.rpi.edu/~puninj/XGMML/" start_time="12/Oct/2000:12:50:11"
access_count="4">
<path count="3">
<uedge source="3" target="10" utime="12/Oct/2000:12:50:12"/>
<uedge source="3" target="21" utime="12/Oct/2000:12:51:41"/>
<uedge source="21" target="22" utime="12/Oct/2000:12:52:02"/>
</path>
</userSession>
```

LOGML Generator: We have written a simple LOGML Generator as part of our WWWPal System. The LOGML Generator reads a common or extended log file and generates a LOGML file. The LOGML Generator also can read the webgraph (XGMML file) of the web site being analyzed and combine the information of the web pages and hyperlinks with the log information.

The information that we extract from the common log files include host name or IP, date of the request, relative URI of the requested page, HTTP version, HTTP status code, HTTP method and the number of bytes transferred to the web client. The extended log files additionally contain the absolute URI of the referer web page and a string that describes the User Agent (web browser or web crawler) that has made the request. This information is saved in a data structure to generate the corresponding LOGML document. The LOGML Generator also can output HTML reports making this module a powerful tool for web administrators.

Several algorithms have been developed to find the user sessions in the log files (Cooley, Mobasher, & Srivastava 1999; Wu, Yu, & Ballman 1997). A simple algorithm uses the IP or host name of the web client to identify a user. SpeedTracer System (Wu, Yu, & Ballman 1997) also checks the User Agent and date of the request to find the user session. Straight ways to find user session requires “cookies” or remote user identification (Cooley, Mobasher, & Srivastava 1999). The LOGML Generator algorithm, to find user sessions, is very similar to the algorithm used by SpeedTracer System.

4 Using LOGML for Web Data Mining

In this section, we propose solving a wide class of mining problems that arise in web data mining, using a novel, generic framework, which we term Frequent Pattern Mining (FPM). FPM not only encompasses important data mining techniques like discovering associations and frequent sequences, but at the same time generalizes the problem to include more complex patterns like tree mining and graph mining. These patterns arise in complex domains like the web. Association mining, and frequent subsequence mining are some of the specific instances of

FPM that have been studied in the past (Agrawal *et al.* 1996; Zaki 2000; Srikant & Agrawal 1996; Zaki 2001b). In general, however, we can discover increasingly complex structures from the same database. Such complex patterns include frequent subtrees, frequent DAGs and frequent directed or undirected subgraphs. Mining such general patterns was also discussed in (Schmidt-Thieme & Gaul 2001). As one increases the complexity of the structures to be discovered, one extracts more informative patterns.

The same underlying LOGML document that stores the web graph, as well as the user sessions, which are subgraphs of the web graph, can be used to extract increasingly complex and more informative patterns. Given a LOGML document extracted from the database of web access logs at a popular site, one can perform several mining tasks. The simplest is to ignore all link information from the user sessions, and to mine only the frequent sets of pages accessed by users. The next step can be to form for each user the sequence of links they followed, and to mine the most frequent user access paths. It is also possible to look at only the forward accesses of a user, and to mine the most frequently accessed subtrees at that site. Generalizing even further, a web site can be modeled as a directed graph, since in addition to the forward hyperlinks, it can have back references, creating cycles. Given a database of user accesses one can discover the frequently occurring subgraphs.

In the rest of this section, we first formulate the FPM problem. We show how LOGML facilitates the creation of a database suitable for web mining. We illustrate this with actual examples from RPI logs (from one day). Using the same example we also describe several increasingly complex mining tasks that can be performed.

4.1 Frequent Pattern Mining: Problem Formulation

FPM is a novel, generic framework for mining various kinds of frequent patterns. Consider a database $\mathcal{D}$ of a collection of structures, built out of a set of primitive *items* $\mathcal{I}$. A structure represents some relationship among items or sets of items. For a given structure G, let $S \preceq G$ denote the fact that S is a substructure of G. If $S \preceq G$ we also say that G *contains* S. The collection of all possible structures composed of the set of items $\mathcal{I}$ forms a partially ordered set under the substructure relation $\preceq$. A structure formed from k items is called a *k-structure*. A structure is called *maximal* if it is not a substructure of any other in a collection of structures. We define the *support* of a structure G in a database $\mathcal{D}$ to be the number of structures in $\mathcal{D}$ that contain G. Alternately, if there is only one very large structure in the database, the support is the number of times G occurs as a substructure within it. We say that a structure is *frequent* if its support is more than a user-specified *minimum support (min_sup)* value. The set of frequent k-structures is denoted as $\mathcal{F}_k$.

A *structural rule* is an expression $X \Rightarrow Y$, where X and Y are structures. The *support* of the rule in the database of structures is the joint probability of X and Y, and the *confidence* is the conditional probability that a structure contains Y, given that it contains X. A rule is *strong* if its confidence is more than a user-specified *minimum confidence (min_conf)*.

The frequent pattern mining task is to generate all structural rules in the database, which have a support greater than *min_sup* and have confidence greater than *min_conf*. This task can be broken into two main steps: 1) *Find all frequent structures having minimum support and other constraints.* This step is the most computational and I/O intensive step, since the search space for enumeration of all frequent substructures is exponential in the worst case. The minimum support criterion is very successful in reducing the search space. In addition other constraints can be induced, such as finding maximal, closed or correlated substructures. 2) *Generate all strong structural rules having minimum confidence.* Rule generation is also exponential in the size of the longest substructure. However, this time we do not have to access the database; we only need the set of frequent structures.

4.2 Database Creation: LOGML to Web Mining

We designed the LOGML language to facilitate web mining. The LOGML document created from web logs has all the information we need to perform various FPM tasks. For structure mining from web logs, we mainly make use of two sections of the LOGML document. As described above, the first section contains the web graph; i.e., the actual structure of the web site in consideration. We use the web graph to obtain the page URLs and their node identifiers. For example, the example below shows a snippet of the (node id, URL) pairs (out of a total of 56623 nodes) we extracted from the web graph of the RPI computer science department:

```
1 http://www.cs.rpi.edu/
6 http://www.cs.rpi.edu/courses/
8 http://www.cs.rpi.edu/current-events/
12 http://www.cs.rpi.edu/People/
14 http://www.cs.rpi.edu/research/
...
```

For enabling web mining we make use of the third section of the LOGML document that stores the user sessions organized as subgraphs of the web graph. We have complete history of the user clicks including the time at which a page is requested. Each user session has a session id (the IP or host name), a path count (the number of source and destination node pairs) and the time when a link is traversed. We simply extract the relevant information depending on the mining task at hand. For example if our goal is to discover frequent sets of pages accessed, we ignore all link information and note down the unique source or destination nodes in a user session. For example, let a user session have the following information as part of a LOGML document:

```
<userSession name="ppp0-69.ank2.isbank.net.tr" ...>
<path count="6">
<uedge source="5938" target="16470"
utime="24/Oct/2000:07:53:46"/>
<uedge source="16470" target="24754"
utime="24/Oct/2000:07:56:13"/>
<uedge source="16470" target="24755"
utime="24/Oct/2000:07:56:36"/>
<uedge source="24755" target="47387"
utime="24/Oct/2000:07:57:14"/>
<uedge source="24755" target="47397"
utime="24/Oct/2000:07:57:28"/>
<uedge source="16470" target="24756"
utime="24/Oct/2000:07:58:30"/>
```

We can then extract the set of nodes accessed by this user:

```
#format: user name, number of nodes accessed, node list
ppp0-69.ank2.isbank.net.tr 7 5938 16470 24754 24755 47387 47397 24756
```

After extracting this information from all the user sessions we obtain a database that is ready to be used for frequent set mining, as we shall see below. On the other hand if our task is to perform sequence mining, we look for the longest forward links, and generate a new sequence each time a back edge is traversed. Using a simple stack-based implementation all maximal forward node sequences can be found. For the example user session above this would yield:

```
#format: user name, sequence id, node position, node accessed
ppp0-69.ank2.isbank.net.tr 1 1 5938
ppp0-69.ank2.isbank.net.tr 1 2 16470
ppp0-69.ank2.isbank.net.tr 1 3 24754
ppp0-69.ank2.isbank.net.tr 2 1 5938
ppp0-69.ank2.isbank.net.tr 2 2 16470
ppp0-69.ank2.isbank.net.tr 2 3 24755
ppp0-69.ank2.isbank.net.tr 2 4 47387
ppp0-69.ank2.isbank.net.tr 3 1 5938
ppp0-69.ank2.isbank.net.tr 3 2 16470
ppp0-69.ank2.isbank.net.tr 3 3 24755
ppp0-69.ank2.isbank.net.tr 3 4 47397
ppp0-69.ank2.isbank.net.tr 4 1 5938
ppp0-69.ank2.isbank.net.tr 4 2 16470
ppp0-69.ank2.isbank.net.tr 4 3 24756
```

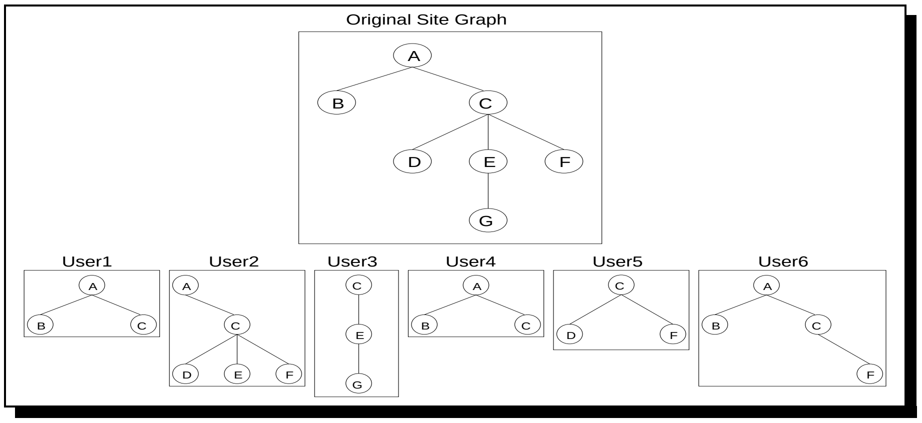


Figure 2: LOGML Document: Web Site Graph and User Sessions

For more complex mining task like tree or graph mining, once again the appropriate information can be directly produced from the LOGML user sessions.

We will illustrate various instances of the FPM paradigm in web mining using the example in Figure 2, which pictorially depicts the original web graph of a particular web site. There are 7 pages, forming the set of primitive items $\mathcal{I} = \{A, B, C, D, E, F, G\}$ connected with hyperlinks. Now the LOGML document already stores in a systematic manner the user sessions, each of them being a subgraph of the web graph. The figure shows the pages visited by 6 users. We will see below how this user browsing information can be used for mining different kinds of increasingly complex substructures, starting with the frequently accessed pages and frequently traversed paths, to the frequent subtrees, etc.

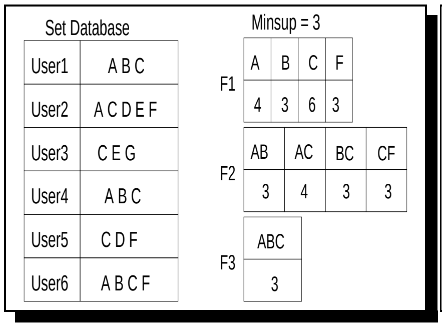
4.3 Web Data Mining

Frequent Sets: This is the well known association rule mining problem(Agrawal *et al.* 1996; Zaki 2000). Here the database $\mathcal{D}$ is a collection of *transactions*, which are simply subsets of primitive items $\mathcal{I}$. Each structure in the database is a transaction, and $\preceq$ denotes the subset relation. The mining task, then, is to discover all frequent subsets in $\mathcal{D}$. These subsets are called *itemsets* in association mining literature.

Consider the example web logs database shown in Figure 3. For each user (in Figure 2) we only record the pages accessed by them, ignoring the path information. The mining task is to find all frequently accessed sets of pages. Figure 3 shows all the frequent k-itemsets $\mathcal{F}_k$ that are contained in at least three user transactions; i.e., $min\_sup = 3$. ABC, AF and CF, are the maximal frequent itemsets.

We applied the Charm association mining algorithm (Zaki & Hsiao 2002) to a real LOGML document from the RPI web site (one day's logs). There were 200 user sessions with an average of 56 distinct nodes in each session. It took us 0.03s to do the mining with 10% minimum support. An example frequent set found is shown below:

```
FREQUENCY = 22 , NODE IDS =  25854 5938 25649 25650 25310 16511
        http://www.cs.rpi.edu/ sibel/poetry/poems/nazim_hikmet/turkce.html
        http://www.cs.rpi.edu/ sibel/poetry/sair_listesi.html
        http://www.cs.rpi.edu/ sibel/poetry/frames/nazim_hikmet_1.html
        http://www.cs.rpi.edu/ sibel/poetry/frames/nazim_hikmet_2.html
        http://www.cs.rpi.edu/ sibel/poetry/links.html
        http://www.cs.rpi.edu/ sibel/poetry/nazim_hikmet.html
```



Set Database

User1	A B C
User2	A C D E F
User3	C E G
User4	A B C
User5	C D F
User6	A B C F

Minsup = 3

F1

A	B	C	F
4	3	6	3

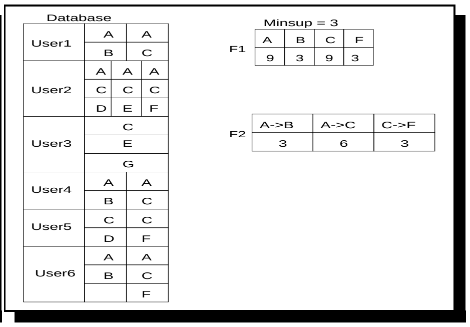
F2

AB	AC	BC	CF
3	4	3	3

F3

ABC
3

Figure 3: Set Mining



Database

User1	A	A	
	B	C	
User2	A	A	A
	C	C	C
	D	E	F
User3	C		
	E		
	G		
User4	A	A	
	B	C	
User5	C	C	
	D	F	
User6	A	A	
	B	C	
		F	

Minsup = 3

F1

A	B	C	F
9	3	9	3

F2

A->B	A->C	C->F
3	6	3

Figure 4: Sequence Mining

Frequent Sequences: The problem of mining sequences (Srikant & Agrawal 1996; Zaki 2001b) can be stated as follows: An *event* is simply an itemset made up of the items $\mathcal{I}$. A *sequence* is an ordered list of events. A sequence α is denoted as $(\alpha_1 \to \alpha_2 \to \cdots \to \alpha_q)$, where α_i is an event; the symbol $\to$ denotes a "happens-after" relationship. We say α is a *subsequence* of another sequence β, denoted as $\alpha \preceq \beta$, if there exists a one-to-one order-preserving function f that maps events in α to events in β, such that, 1) $\alpha_i \subseteq f(\alpha_i)$, and 2) if $\alpha_i < \alpha_j$ then $f(\alpha_i) < f(\alpha_j)$.

The structure database $\mathcal{D}$ consists of a collection of sequences, and $\preceq$ denotes the subsequence relation. The mining goal is to discover all frequent subsequences. For example, consider the sequence database shown in Figure 4, by storing all paths from the starting page to a leaf (note that there are other ways of constructing user access paths; this is just one example). With minimum support of 3 we find that $A \to B$, $A \to C$, $C \to F$ are the maximal frequent sequences.

We applied the SPADE sequence mining algorithm (Zaki 2001b) to an actual LOGML document from the RPI web site. From the 200 user sessions, we obtain 8208 maximal forward sequences. It took us 0.12s to do the mining with minimum support set to 0.1%. An example frequent sequence found is shown below:

```
Let Path=http://www.cs.rpi.edu/∼sibel/poetry
FREQUENCY = 21, NODE IDS =  37668 -> 5944 -> 25649 -> 31409
  Path/ -> Path/translation.html ->
  Path/frames/nazim_hikmet_1.html -> Path/poems/nazim_hikmet/english.html
```

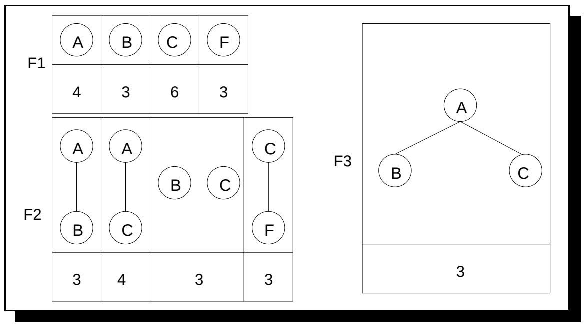


Figure 5: Frequent Tree Mining

Frequent Trees: We denote an ordered, labeled tree as $T = (V_t, E_t)$, where V_t is the vertex set, and E_t are the edges or branches. We say that a tree $S = (V_s, E_s)$ is a subtree of T, denoted as $S \preceq T$, if and only if $V_s \subseteq V_t$, and for all edges $e = (v_1, v_2) \in E_s$, v_1 is an ancestor or descendant of v_2 in T. Note that this definition is different from the usual definition of a subtree. In our case, we require that for any branch that appears in S, the two vertices must be on the same path from a root to some leaf. For example, in Figure 2 the tree S, with $V = \{C, G\}$ and $E = \{CG\}$ is a subtree of the site graph.

Given a database $\mathcal{D}$ of trees (i.e., a forest) on the vertex set $\mathcal{I}$, the frequent tree mining problem (Zaki 2001a) is to find all subtrees that appear in at least *min_sup* trees. For example, for the user access subtrees shown in Figure 2, we mine the frequent subtrees shown in Figure 5. There are two maximal frequent subtrees, $(V = \{C, F\}, E = \{CF\})$ and $(V = \{A, B, C\}, E = \{AB, AC\})$ for *min_sup* $= 3$.

We applied the TreeMinerV algorithm (Zaki 2001a) to the same RPI LOGML file used above. From the 200 user sessions, we obtain 1009 subtrees. It took us 0.37s to do the mining with minimum support set to 5% (or a frequency of at least 50). An example frequent subtree found is shown below (-1 denotes a back edge):

```
Let Path=http://www.cs.rpi.edu/~sibel/poetry
Let Poet = Path/poems/orhan_veli
FREQUENCY = 65, NODE IDS = 16499 31397 37807 -1 37836 -1 -1 25309
             Path/orhan_veli.html
              /                \
             /                  \
   Poet/turkce.html       Path/frames/orhan_veli_2.html
    /           \
   /             \
Poet/golgem.html  Poet/gunes.html
```

Source	Raw Logs		LOGML		LOGML Breakdown		
	Regular	Gzip	Regular	Gzip	Webgraph	Sessions	Other
RPI1(14Jun)	52.4MB	5.5MB	19.9MB	2.1MB	88.3%	8.3%	3.4%
RPI2(15Jun)	52.4MB	5.5MB	19.4MB	2.1MB	88.2%	8.4%	3.4%
CS1 (28Jun)	10.5MB	1.1MB	4.6MB	0.5MB	74.6%	16.7%	8.7%
CS2 (29Jun)	10.3MB	1.1MB	5.3MB	0.6MB	75.8%	16.6%	7.6%

Table 1: Size of Raw Log Files versus LOGML Files (Size is in Bytes)

Size of LOGML Documents: Since raw log files can be large, there is a concern that the LOGML files will be large as well. Table 1 shows the observed size of raw log files compared to the LOGML documents (with and without compression), the number of requests and user sessions, and the breakdown of LOGML files for the CS department (*www.cs.rpi.edu*) and RPI web site (*www.rpi.edu*). For example, for RPI1 (logs from 14th June, 2001) there were about 275,000 request for different nodes comprising 6,000 user sessions. The LOGML file is more than 2.5 times *smaller* than the raw log file. Same trends are observed for other sources.

The benefits of LOGML become prominent when we consider the breakdown of the LOGML files. For the RPI site we find that about 88% of the LOGML file is used to store the webgraph, while the user sessions occupy only 8% (the other elements to store statistics, etc. use up 3.4% space). For the CS department site, we find that the webgraph takes about 75% space, while the user sessions occupy 17%. In general, the webgraph is not likely to change much from one day to the next, and even if it does, one can always store a master webgraph spanning several days or months separately. Then on a per day basis we need only store the user sessions (and the other LOGML sections if desired). For example for the RPI site this would require us to store 174,573 bytes per day, while for the CS site is comes to 86,888 bytes per day for storing only the user sessions (with compression). Thus, not only does LOGML facilitate web usage mining, it also can drastically reduce the amount of daily information that needs to be stored at each site.

Conclusion: In this paper, we defined two new XML languages, XGMML and LOGML, and a web usage mining application. XGMML is a graph file description format, and an ideal candidate to describe the structure of web sites. Furthermore XGMML is a container for meta-data information. LOGML, on the other hand, is an extension of XGMML to collect web usage. Future work includes mining user graphs (structural information of web usages), as well as visualization of mined data using WWWPal system (Punin & Krishnamoorthy 1998). To perform web content mining, we need keyword information and content for each of the nodes. Obtaining this information will involve analyzing each of the web pages and collecting relevant keywords. Work is under way to accomplish this task.

References

AGRAWAL, R.; MANNILA, H.; SRIKANT, R.; TOIVONEN, H.; and VERKAMO, A. I. (1996): Fast discovery of association rules. In Fayyad, U., and et al. (Eds.), *Advances in Knowledge Discovery and Data Mining*, 307–328. AAAI Press, Menlo Park, CA.

CHEN, M.; PARK, J.; and YU, P. (1996): Data mining for path traversal patterns in a web environment. In *International Conference on Distributed Computing Systems*.

COOLEY, R.; MOBASHER, B.; and SRIVASTAVA, J. (1997): Web mining: Information and pattern discovery on the world wide web. In *8th IEEE Intl. Conf. on Tools with AI*.

COOLEY, R.; MOBASHER, B.; and SRIVASTAVA, J. (1999): Data preparation for mining world wide web browsing pattern. *Knowledge and Information Systems* 1(1).

KOSALA, R., and BLOCKEEL, H. (2000): Web mining research: A survey. *SIGKDD Explorations* 2(1).

MASAND, B., and SPILIOPOULOU, M. (Eds.). (2000): *Advances in Web Usage Mining and User Profiling: Proceedings of the WEBKDD'99 Workshop.* Number 1836 in LNAI. Springer Verlag.

PUNIN, J., and KRISHNAMOORTHY, M. (1998): WWWPal System - A System for Analysis and Synthesis of Web Pages. In *Proceedings of the WebNet 98 Conference*.

PUNIN, J., and KRISHNAMOORTHY, M. (2000): Log Markup Language Specification. `http:// www.cs.rpi.edu/ ~puninj/ LOGML/ draft-logml.html`.

SCHMIDT-THIEME, L., and GAUL, W. (2001): Frequent substructures in web usage data - a unified approach. In *Web Mining Workhsop (with 1st SIAM Int'l Conf. on Data Mining.*

SPILIOPOULOU, M., and FAULSTICH, L. (1998): WUM: A Tool for Web Utilization Analysis. In *EDBT Workshop WebDB'98, LNCS 1590.* Springer Verlag.

SRIKANT, R., and AGRAWAL, R. (1996): Mining sequential patterns: Generalizations and performance improvements. In *5th Intl. Conf. Extending Database Technology.*

WU, K.; YU, P.; and BALLMAN, A. (1997): Speed Tracer: A Web usage mining and analysis tool. *Internet Computing* 37(1):89.

ZAKI, M. J., and HSIAO, C.-J. (2002): CHARM: An efficient algorithm for closed itemset mining. In *2nd SIAM International Conference on Data Mining.*

ZAKI, M. J. (2000): Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering* 12(3):372-390.

ZAKI, M. J. (2001a): Efficiently mining trees in a forest. Technical Report 01-7, Computer Science Dept., Rensselaer Polytechnic Institute.

ZAKI, M. J. (2001b): SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal* 42(1/2):31–60.