LOGML: Log Markup Language for Web Usage Mining

John R. Punin, Mukkai S. Krishnamoorthy, Mohammed J. Zaki
Computer Science Department
Rensselaer Polytechnic Institute, Troy NY 12180
Email: {puninj,moorthy,zaki}@cs.rpi.edu

ABSTRACT

Web Usage Mining refers to the discovery of interesting information from user navigational behavior as stored in web access 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 using the web robot of the WWWPal system. We generate web-log reports in LOGML format for a web site from web log files and the web graph. We further illustrate the usefulness of LOGML in web usage mining; we show the simplicity with which mining algorithms (for extracting increasingly complex frequent patterns) can be specified and implemented efficiently using LOGML.

1. INTRODUCTION

Recently XML has gained wider acceptance in both commercial and research establishments. In this paper, we suggest 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 1.0 application based on Graph Modeling Language (GML; see *http://www.infosun.fmi.uni-passau.de/Graphlet/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 for graphs. 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. In Section 2, we present details of XGMML.

Log Markup Language (LOGML) is an XML 1.0 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. In section 2, we explain how the structure of a web site can be represented as a web graph using XGMML. 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. In section 3, we discuss LOGML in detail. Section 4 describes LOGML generator as an additional module for the WWWPal system [7].

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 [8], which deals with problems of automatic information filtering and categorization, intelligent search agents, and personalize web agents. Web usage mining [8] 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.

The proposed LOGML and XGMML languages have been designed to facilitate this web mining process in addition

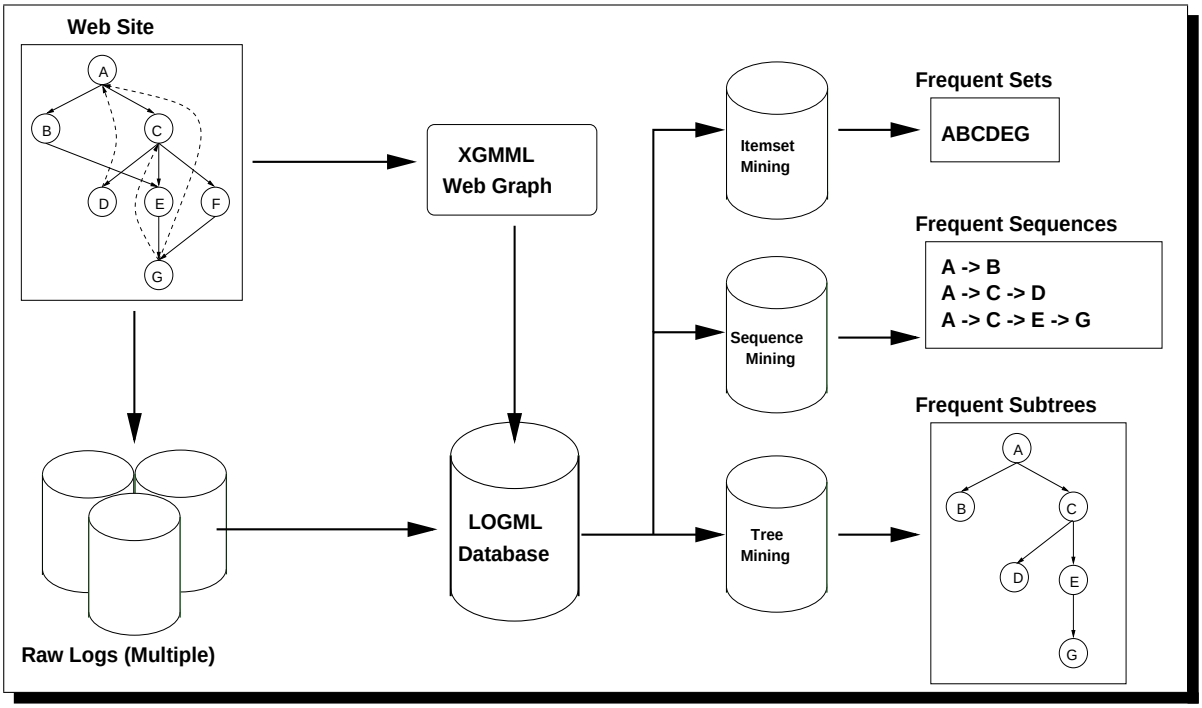

Figure 1: Web Usage Mining Architecture

to storing additional summary information extracted from web logs. Using the LOGML generated documents the preprocessing steps of mining are considerably simplified. We also propose a new mining paradigm, called Frequent Structure 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. [2] introduced the notion of a maximal forward chain of web pages and gave an algorithm to mine them. The WUM system [9] 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. [3] 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 [5; 4]. 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 (EXTENSIBLE GRAPH MARKUP AND MODELING LANGUAGE)

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 [7] 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. The following example is a graph with just one node.

```
<?xml version="1.0"?>
<!DOCTYPE graph PUBLIC "-//DTD graph description//EN"
          "http://www.cs.rpi.edu/~puninj/XGMML/xgmml.dtd">
<graph directed="1" id="2">
<node id="1" label="Node 1"/>
</graph>
```

XGMML well formed documents can be part of other XML documents using namespaces. The following example is a graph inside of an XHTML document :

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
      xmlns:xgmml="http://www.cs.rpi.edu/XGMML"
      xsi:schemaLocation="http://www.w3.org/1999/Style/Transform
              http://www.w3.org/1999/Style/Transform/xslt.xsd
              http://www.w3.org/1999/xhtml
              http://www.w3.org/1999/xhtml/xhtml.xsd
              http://www.cs.rpi.edu/XGMML
              http://www.cs.rpi.edu/~puninj/XGMML/xgmml.xsd"
      xml:lang="en">
<head>
<title>Graph Information</title>
</head>
<body>
<!-- XHTML Document here -->
     <xgmml:graph directed="1" graphic="1" Layout="points">
       <xgmml:node id="1" label="1" weight="0">
         <xgmml:graphics type="circle" x="250" y="90" />
       </xgmml:node>
       <xgmml:node id="2" label="2" weight="0">
         <xgmml:graphics type="circle" x="190" y="150" />
       </xgmml:node>
       <xgmml:edge source="1" target="2" weight="0" />
     </xgmml:graph>
<!-- XHTML Document here -->
</body>
</html>
```

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. Example 3 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="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="www@cs.rpi.edu; M.S. requirements; CSCI-1190
    Beginning C Programming for Engineers; Courses; People;
    Graduate Program; CSCI-4020 Computer  Algorithms; CSCI-
    2220-01  Programming in Java; Research; Course Selection
    Guide; CSCI-4961-01,  CSCI-6961-01 Advanced Robotics;
    Programming in Java; CSCI-2400 Models  of Computation"
    dc:date="2000-01-31"
    dc:type="Text"
    >
    <dc:format>
      <rdf:Bag
        rdf:_1="text/html"
        rdf:_2="5427 bytes"
      />
    </dc:format>
  </rdf:Description>
</rdf:RDF>
</att>
</node>
....
<edge src="1" target="3" weight="0" label="SRC IMG X.jpg" />
<edge src="7" target="3" weight="0" label="SRC IMG ../X.jpg" />
</graph>
```

Valid XGMML Documents: A valid XGMML document must be an well-formed XML document. A valid XGMML document additionally can be validated against an XGMML DTD or XGMML Schema. The XGMML Schema is based on the XML Schema Working Draft 22 September 2000. A valid XML document can have multiple schemas. The namespace for XGMML is: *www.cs.rpi.edu/XGMML* and the suffix for the XGMML elements is *xgmml:*. The examples above show two valid XML documents that can be validated using several XML schemas including XGMML Schema.

XGMML Elements and Attributes: The main elements of XGMML are: `graph`, `node`, `edge`, `att` and `graphics`. The `graph` element is the root element of an XGMML valid document. The `graph` element may not be unique in the XGMML document. Other graphs can be included as subgraphs of the main graph. All XGMML elements have global attributes that are `id, name` and `label`. The `id` attribute is an unique number to identify the XGMML element. The `name` is a string to identify the elements and the `label` is a string used as a text representation of the elements. The graph element has the `directed` attribute that is a boolean value to express whether the graph is directed or not.

Nodes and edges can reference XGMML documents. For example, a node may represent a graph that can be shown when the user points inside the node. This behavior is similar to hyperlinks in HTML documents. XGMML uses XLink framework to create hyperlinks either in nodes or edges. The XLink attributes: `type`, `role`, `title`, `show`, `actuate` and `href`, are added as attributes of the node and edge elements. All these attributes are taken directly from the XLink Working Draft.

The `node` element describes the properties of a node object. The node can be rendered as a graphic object and also can have additional meta information to be used for the application program. The only elements allowed inside the node are `graphics` and `att`. The graphic representation of the node is reported on the `graphics` element. For example, a graphical representation of a node can be a rectangle, a circle or a bitmap. The additional meta information is reported on the `att` element. For example, if a node is a representation of a web page, useful metadata is the title, date of creation and size of the web page.

The `edge` element describes the properties of an edge object. For each `edge` element two `node` elements have to be included in the `graph` element. An edge is between a source node and a target node. The application program must verify if the source node and target node are included in the XGMML document. The `weight` attribute is used to save the weight number for weighted graphs. The `edge` element as the `node` element can have a graphical representation and additional metadata information. The `graphics` element shows the graphical representation of an edge. For example, a graphical representation of an edge can be a line or an arc. An `att` element is used to attach additional meta information related to an edge. For example, if an edge is a representation of a hyperlink, useful metadata is the anchor string and the type of the hyperlink (Typed Links) [11].

An `att` element is used to hold meta information about the element that contains the `att` element. An `att` element can contain other `att` elements, say to represent structured metadata such as records, lists, etc. For example, the metadata of a person object A is name: John, ssn: 123456789 and e-mail: john@rpi.edu. To attach this metadata to a node of a graph using the att element, the following lines must be included in the node element:

```
<att type="list" name="person_description">
<att name="name" value="John"/>
<att name="ssn" value="123456789"/>
<att name="e-mail" value="john@rpi.edu"/>
</att>
```

The `graphics` element defines the graphical representation of a graph, a node or an edge. `Line`, `center` and `att` elements are the only elements that can be contained in a graphics element. `Line` element is defined between two `point` elements and it is used to represent edges. A `center` element is a special `point` element to represent the central point of the graphical representation of a node. The `att` element permits to add information to the graphical representation. All these elements are inherited from GML.

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; example below shows part of a LOGML file.

```
<?xml version="1.0"?>
<logml xmlns="http://www.cs.rpi.edu/LOGML"
       xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
       xsi:schemaLocation="http://www.cs.rpi.edu/LOGML
       http://www.cs.rpi.edu/~puninj/LOGML/logml.xsd"
        start_date="12/Oct/2000:05:00:05"
        end_date="12/Oct/2000:16:00:01">
<graph xmlns="http://www.cs.rpi.edu/XGMML"
       xmlns:lml="http://www.cs.rpi.edu/LOGML"
       xsi:schemaLocation="http://www.cs.rpi.edu/XGMML
       http://www.cs.rpi.edu/~puninj/XGMML/xgmml.xsd
       http://www.cs.rpi.edu/LOGML
       http://www.cs.rpi.edu/~puninj/LOGML/logml.xsd"
       directed="1">
<node id="234" label="http://www.cs.rpi.edu/~puninj/JAVA/projects/lfarrw.gif"
 lml:hits="1" weight="1">
<att name="title" value="No title"/>
<att name="mime" value="image/gif"/>
<att name="size" value="1291"/>
<att name="date" value="Sun Jun 11 02:14:28 2000"/>
<att name="code" value="200"/>
</node>
....
<edge source="191" target="234" label="SRC IMG lfarrw.gif" lml:hits="1" weight="1">
<att value="image"/>
</edge>
....
<edge source="550" target="561" lml:hits="1" weight="1" lml:indp="1"/>
....
</graph>
<hosts count="35">
<host name="vamos.inria.fr" access_count="43" bytes="487397" html_pages="43"/>
<host name="kbl-ternzn1200.zeelandnet.nl" access_count="13" bytes="46354"
html_pages="1"/>
....
</hosts>
<domains count="9">
<domain name="unknown" access_count="25" bytes="388608" html_pages="16"/>
<domain name="com" access_count="21" bytes="229979" html_pages="19"/>
....
</domains>
<directories count="30">
<directory name="http://www.cs.rpi.edu/~puninj/XGMML" access_count="21"
total_count="49" bytes="1116521"/>
....
</directories>
<userAgents count="23">
<userAgent name="Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)"
access_count="27" bytes="670815" html_pages="9"/>
....
</userAgents>
<hostReferers count="14">
<hostReferer name="No Referer" access_count="66" bytes="945527"/>
<hostReferer name="http://www.cs.rpi.edu" access_count="41" bytes="701097"/>
....
</hostReferers>
<referers count="11">
<referer name="No referer" access_count="66" bytes="945527"/>
access_count="1" bytes="35272" target="8"/>
<referer name="http://informant.dartmouth.edu/" access_count="1" bytes="1112"
 target="2"/>
....
</referers>
<keywords count="10" search_count="9">
<keyword name="java" count="3"/>
<keyword name="xhtml" count="2"/>
....
</keywords>
<summary
 requests="132" sessions="6" bytes="1796173"
```

```
html_pages="56" nhtml_pages="17" inline_objects="10" hyperlink_html="7"
hyperlink_nhtml="16"
html_entry_pages="55" nhtml_entry_pages="4" unique_sites="35" unique_host_referers="8"
unique_se_referers="6"
unique_external_url_referers="7" unique_internal_url_referers="4" unique_user_agents="23"
requests_hour="12.00" requests_day="288.03" kbytes_day="159.48" kbytes_hour="3827.46"
searches="9" unique_keywords="10">
<httpCode code="200" name="200 - OK " count="118" bytes="1793393" html_pages="83"/>
<httpCode code="404" name="404 - Not Found" count="5" bytes="1722" html_pages="5"/>
<httpMethod name="GET" count="131" bytes="1796173" html_pages="95"/>
<httpMethod name="HEAD" count="1" bytes="0" html_pages="1"/>
<httpCode name="HTTP/1.0" count="97" bytes="1399288" html_pages="83"/>
<httpCode name="HTTP/1.1" count="35" bytes="396885" html_pages="13"/>
<dateStat>
<monthStat month="10" hits="132" bytes="1796173" html_requests="96"/>
<dayStat day="12" hits="132" bytes="1796173" html_requests="96"/>
<hourStat hour="5" hits="12" bytes="15622" html_requests="12"/>
<hourStat hour="6" hits="15" bytes="103280" html_requests="14"/>
</dateStat>
</summary>
....
<userSessions count="2" max_edges="100" min_edges="2">
<userSession name="proxy.artech.com.uy" ureferer="No referer"
entry_page="http://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>
<userSession name="207.234.33.12"
ureferer="http://search.excite.com/search.gw?search=XHTML"
entry_page="http://www.cs.rpi.edu/~puninj/TALK/head.html"
start_time="12/Oct/2000:14:05:10" access_count="3">
<path count="2">
<uedge source="2" target="7" utime="12/Oct/2000:14:05:24"/>
<uedge source="2" target="8" utime="12/Oct/2000:14:06:14"/>
</path>
</userSession>
</userSessions>
</logml>
```

LOGML Valid Documents: A LOGML valid document is a well-formed XML document that can be validated against a LOGML DTD or LOGML Schema. The namespace for LOGML is *http://www.cs.rpi.edu/LOGML* and the suffix for LOGML elements is *lml:*.

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 are:

• `hosts, host` - This host list is composed by a container `hosts` element whose attribute is the `count` of the `host` element inside of the `hosts` element. The `host` element is an empty element and contains information about the visiting site such as hostname, IP and number of bytes transferred by the site.

• `domains, domain` - The `domains` element is a list of all domains visiting the web server. The domain is the suffix of the domain name of the sites. For example: *edu* is the domain of the site *www.cs.rpi.edu.*

• `directories, directory` - The directories list contains the top directories of the web site that have most requested web pages. The directory is the prefix of the URI of the web page. For example: The directory of the web page: *http://www.rpi.edu/dept/urp/find.html* is *http://www.rpi.edu /dept/urp*

• `userAgents, userAgent` - The list of user agents contains all user agents (browsers and/or spiders) that have made requests to the web server. The LOGML reader can refine this list to compute the top platforms and top web browsers since the User Agent name contains information about the platform and name of web browser.

• `referers, referer` - The `referers` list contains two lists: The list of the top external referers and the list of the top internal referers. The external referers are the referers whose host are different than the web server. The host of the internal referers are the same as the web server.

• `hostReferers, hostReferer` - The host referers list contains the top host referers of the web pages of the web server. This list combines the number of accesses of the referers with the same host.

• `keywords, keyword` - Keywords are the searching words found in the URI referers of the search engines. Several `keywords` lists can be reported. Each `keywords` list is associated with a particular search engine.

• `summary` - The `summary` element contains a brief overview of the essential information of the web server. This information is very important for web masters to know the efficiency of the web server. The `summary` attributes are: `requests` - the total number of requests. `sessions` - the total number of user sessions. `bytes` - the total number of bytes transferred. `html_pages` - the total number of unique html pages. `nhtml_pages` - the total number of unique non html pages. `inline_objects` - the total number of unique inline objects. Inline objects are the objects inside of a html page such as images `hyperlinks_html` - the total number of unique hyperlinks to html pages. `hyperlinks_nhtml` - the total number of unique hyperlinks to non html pages. `html_entry_pages` - the total number of unique html pages that are entry pages to the web site of the web server. `nhtml_entry_pages` - the total number of unique non html pages that are entry pages to the web site of the web server. `unique_sites` - the total number of unique visiting sites. `unique_host_referers` - the total number of the unique host. referers to the web pages of the web server. `unique_se_referers` - the total number of the unique search engines that access the web server. `unique_external_url_referers` - the total number of unique external URI referers to the web pages of the web server. `unique_internal_url_referers` - the total number of unique internal URI referers to the web pages of the web server. `unique_user_agents` - the total number of the unique user agents that access the web pages of the web server. `requests_hour` - the number of requests per hour. `requests_day` - the number of requests per day.

`kbytes_hour` - the number of kilobytes transferred per hour. `kbytes_day` - the number of kilobytes transferred per day. `searches` - the total number of searching requests. `uniquekeywords` - the total number of unique keywords in the searching requests.
• `httpCode` - The `httpCode` element gives the summary of the HTTP status code of the requests.
• `httpMethod` - The `httpMethod` element gives the summary of the HTTP methods used by the web clients to communicate with the web server.
• `httpVersion` - The `httpVersion` element gives the summary of the HTTP version used by the web clients to communicate with the web server.
• `dateStat, monthStat, dayStat` and `hourStat`. - The date elements give the summary of the statistics by date of the 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="http://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>
```

4. 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 [3; 6; 12]. A simple algorithm uses the IP or host name of the web client to identify a user. SpeedTracer System [12] 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 [3]. The LOGML Generator algorithm, to find user sessions, is very similar to the algorithm used by SpeedTracer System.

The LOGML Generator has a module called the User Manager. The User Manager is invoked for each web log line that is processed. It received the following information: current date, URL of the requested page, URL of the referer, IP of the user and user agent. The User Manager has access to the container of the user sessions and the web graph of the web site of the web logs so the User Manager can add user sessions and get metadata information from the web graph such as title, size and mime type of the web page.

These are the following steps that the User Manager takes to create and finish user sessions :
• Check if any of the current user sessions has finished. A user session is considered finished when the lapse time between the last request time of the user session and the current request time is greater than a time window. This time window is a parameter of the LOGML generator and from experience we set the value to be 30 minutes. The User Manager marks the finished user sessions so they can be reported in the LOGML document.
• Check if the user agent is a spider. A Spider is being recognized by the name of the user agent or by the excessive number of requests to the web site. Spider sessions are not considered user sessions so the User Manager skips the requests of the spiders.
• Check if the current requested page is an inline object. User sessions are reported as a set of hyperlinks between HTML pages so inline object are not reported in the user session. We can expand the user sessions' inline objects using the log graph of the first section of the LOGML document. The User Manager skips the inline object requests.
• Search for a user session in the table of user sessions. A user session is identified by IP or domain name, and the name of the user agent. If a user session is not found, a new user session is created and stored in the table of user sessions.
• Verify if the referer of the requested page is an external or internal URL of the web site being analyzed. If the referer is external, it means that the requested page is an entry page and a new possible user session has started. The User Manager checks if the current user session has more than two requests and it considers the user session. If the current user session has just one request, the user session is discarded.
• Add the new hyperlink (edge) to the graph of the user session. Each edge is saved with the date (timestamp) where the user has traversed this hyperlink. This timestamp is used for Web usage mining purposes.

Once that the LOGML generator reads all the web log lines, only those finished user sessions are reported in the LOGML document. This is the general algorithm that the User Manager uses to create, add and finish the user sessions.

```
int user_session_manager(WebStat ws, Date d, URL page,
       URL referer, IP ip, UserAgent ua, WebGraph g)
{
   User u;
   check_finished_users(ws,d);
   if(is_spider(ua)) return IS_SPIDER;
   if(is_inline_object(g,page)) return IS_INLINE_OBJECT;
   u = find_user(ws,ip,ua); // find user in the users table
   if(!u) // if user was not found, create a new user session
      u = new_user_session(ws,d,ip,ua);
   if(is_external(referer)) {
      finish_user_session(ws,u);
      u = new_user_session(ws,d,ip,ua);
   }
   add_hyperlink(u,page,referer,d);
   return 0;
}
```

We use the Graph Visualizer of WWWPal System to display the log graph of the LOGML document or any of the user sessions that has been identified in the log files. Figure 2 shows part of the log graph of the Rensselaer News web site (*http://www.rpi.edu/web/News/*). The numbers on the edges are the times that a user has traversed that edge (hyperlink). The number in the nodes are the times that a user has requested the corresponding web page. For visualization purposes just the main nodes of the log graph have been displayed.

5. 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 Structure Mining (FSM). FSM 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 FSM that have been studied in the past [1; 13; 10; 15]. 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. 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 (with full information about their traversals, including forward and backward links) one can discover the frequently occurring subgraphs.

In the rest of this section, we first formulate the FSM 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.

5.1 Frequent Structure Mining

FSM 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 structure 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.

5.2 Database Creation from LOGML

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 FSM tasks. For structure mining from web logs, we mainly make use of two sections of the LOGML document. As described above, the

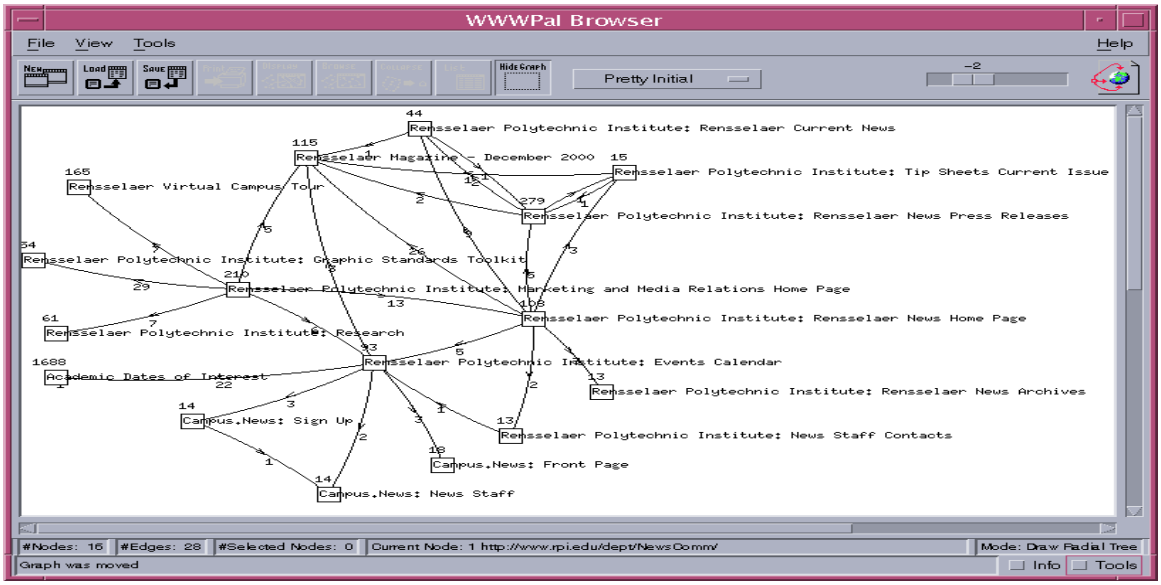

Figure 2: Log graph of RPI News Website

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/
4 http://www.cs.rpi.edu/guide/machines/
6 http://www.cs.rpi.edu/courses/
8 http://www.cs.rpi.edu/current-events/
10 http://www.cs.rpi.edu/grad/
12 http://www.cs.rpi.edu/People/
14 http://www.cs.rpi.edu/research/
16 http://www.cs.rpi.edu/undergrad/
31 http://www.cs.rpi.edu/guide/
...
```

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
```

For frequent tree mining, we can easily extract the forward edges from the user session (avoiding cycles or multiple parents) to obtain the subtree corresponding to each user. For our example above this yields the following record (note: the tree is encoded as a string, using a depth-first traversal of the nodes; a -1 indicates a back edge).

```
#format: user name, number of nodes, node list in tree
ppp0-69.ank2.isbank.net.tr 7 5938 16470 24754 -1 24755 47387
           -1 47397 -1 -1 24756 -1 -1
```

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

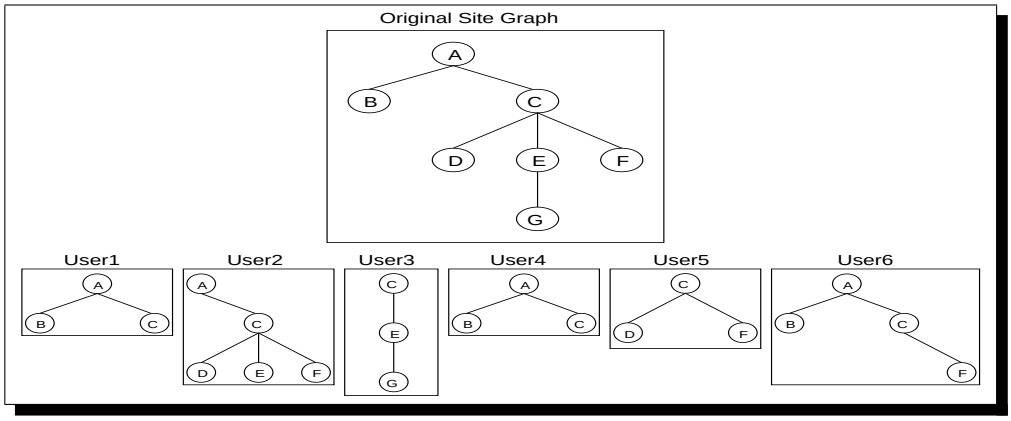

Figure 3: LOGML Document: Web Site Graph and User Sessions

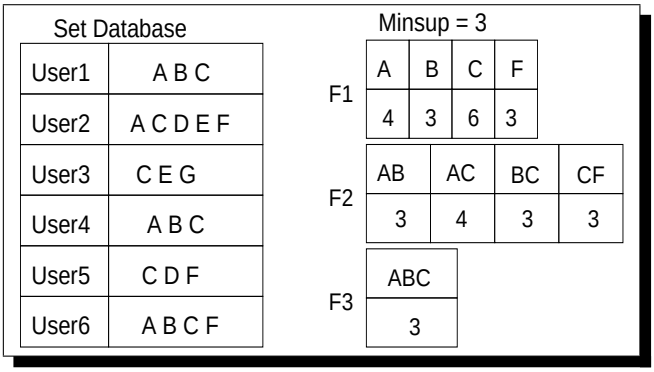

Figure 4: Frequent Set Mining

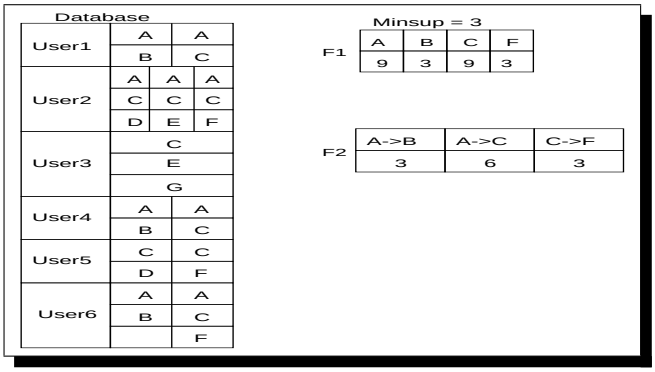

Figure 5: Frequent Sequence Mining

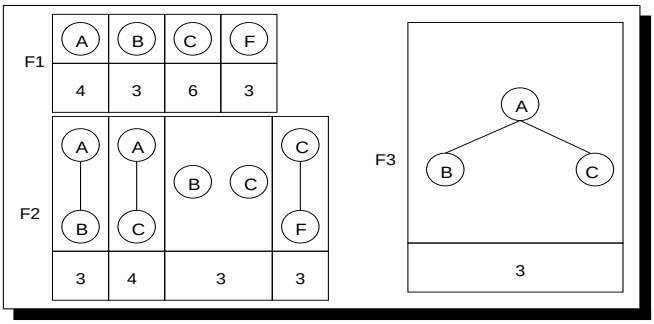

Figure 6: Frequent Tree Mining

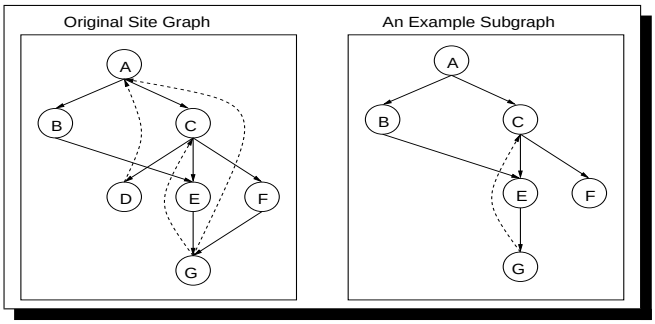

Figure 7: A General User Graph

We will illustrate various instances of the FSM paradigm in web mining using the example in Figure 3, 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, to the frequently traversed paths, to the frequent subtrees, and so on.

5.3 Web Data Mining

Frequent Sets: This is the well known association rule mining problem[1; 13]. 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 4. For each user (in Figure 3) 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 4 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 [16] 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:

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

Frequent Sequences: The problem of mining sequences [10; 15] 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 $(\alpha_1 \rightarrow \alpha_2 \rightarrow \cdots \rightarrow \alpha_q)$, where α_i is an event; the symbol $\rightarrow$ denotes a "happens-after" relationship. We say α is a *subsequence* (not necessarily consecutive) of another sequence β, denoted as $\alpha \preceq \beta$, if α is completely contained within β.

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 5, 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 $A \rightarrow B$, $A \rightarrow C$, $C \rightarrow F$ are the maximal frequent sequences.

We applied the SPADE sequence mining algorithm [15] to an actual LOGML document from the RPI web site. From the 200 user sessions, we obtain 8208 maximal forward sequences, with an average sequence size of 2.8. It took us 0.12s to do the mining with minimum support set to 0.1% (or a frequency of at least 8). 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
```

Frequent Trees: We denote an ordered, labeled, and rooted 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 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 3 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 [14] is to find all subtrees that appear in at least *min_sup* trees. For example, for the user access subtrees shown in Figure 3, we mine the frequent subtrees shown in Figure 6. 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 [14] to the same RPI LOGML file used above. From the 200 user sessions, we obtain 1009 subtrees (a single user session can lead to multiple trees if there are multiple roots in the user graph), with an average record length of 84.3 (including the back edges, -1). 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:

```
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
```

Other Generalizations: It is instructive to compare the patterns returned by the above three tasks from a common web logs database. We started by ignoring all link information to obtain frequent sets of pages. We then found the frequent paths, and finally the frequently traversed subtrees. These tasks were arranged according to increasing order of complexity (and thus increasing execution time), but at the same time in increasing order of information conveyed to the user. For example, in frequent set mining, we only know that the pages A, B, and C were frequently accessed. Sequence mining gives us partial sequence information about the order in which pages are traversed, e.g., $A \rightarrow B$. But in tree mining, we obtain full knowledge about the relationships between the three pages; e.g. A is the root with two children B and C. Not only can one mine such patterns, but it is relatively easy in our framework based on the LOGML document information to apply constraints on the patterns as well. For example, a web site analyst might want to know only those patterns that occur within a short time window, or those that occur after long gaps between accesses, etc. All this information can directly be extracted from the edge times in the user sessions.

There are many other generalizations that are possible. For example, we can generalize the tree mining problem to directed acyclic graphs, and more generally to directed and undirected graphs. Continuing the web mining example, a general web site can be modeled as a directed graph, since in addition to the forward hyperlinks, it can have back references, creating cycles. Figure 7 shows an example web graph. Given a database of user accesses (with full information about their traversal, including both forward and backward links) one might be able to discover the frequently occurring subgraphs, such as the one shown.

Experiments: We ran detailed experiments on logs files collected over 1 month at the RPI computer science department. Experiments were run on a 450Mhz Pentium II processor with 256MB memory, running Linux 6.0. The logs touched a total of 27343 web pages within our department's web site. After processing the LOGML database with 34838 user graphs, we had as many transactions for association mining, with 8.24 items per transaction on average. For the sequence database, from the same LOGML file, we generated 165276 sequences, where the average sequence (or page

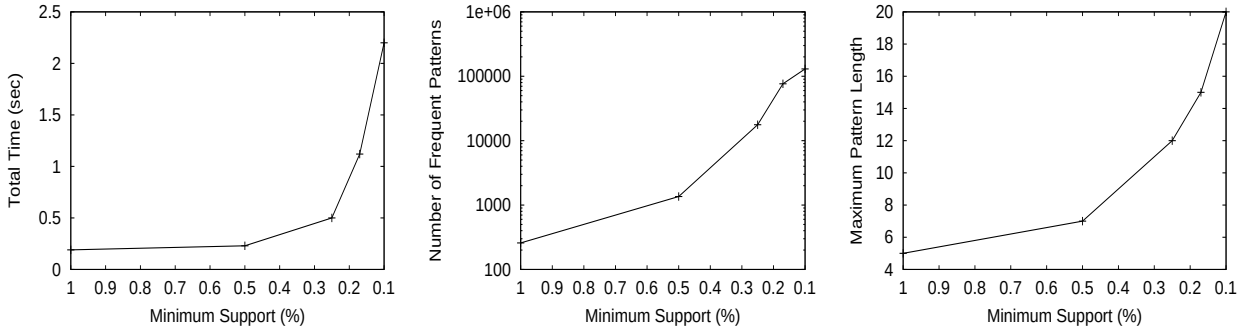

Figure 8: Association Mining: a) Total Time, b) Number of Patterns, and c) Max. Pattern Length

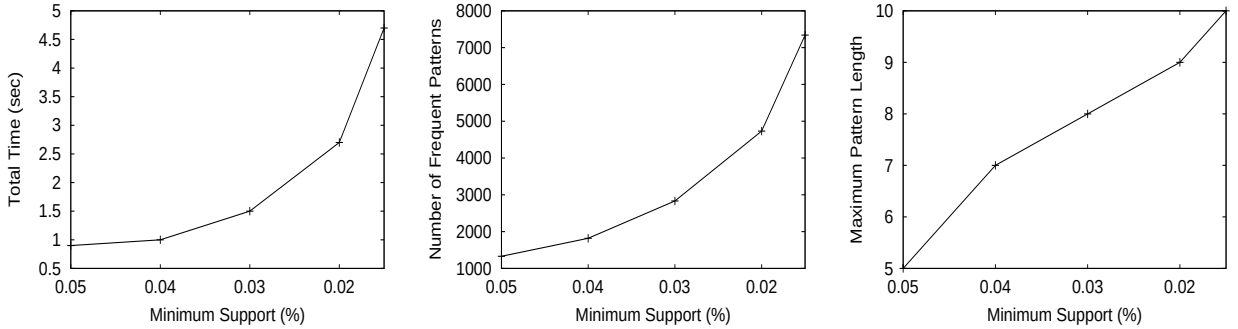

Figure 9: Sequence Mining: a) Total Time, b) Number of Patterns, and c) Max. Pattern Length

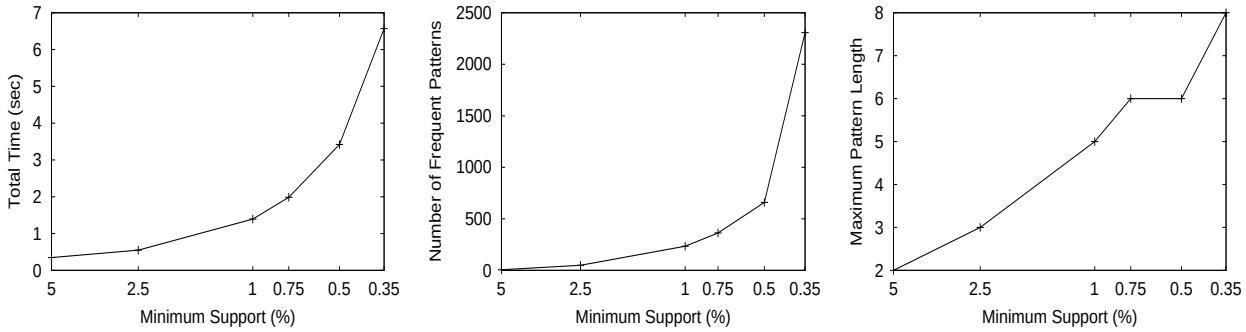

Figure 10: Tree Mining: a) Total Time, b) Number of Patterns, and c) Max. Pattern Length

reference) length was 2.6. Finally for tree mining, we obtained 59691 trees (stored in string format) from the 34838 user graphs, with an average of 20.1 items per tree (including -1's). Note that we can have more trees than the number of user graphs, since a user can have multiple entry points into a web site, and each such entry point (i.e., a page with no parent or with an external parent) serves as the root of a new subtree for the same user.

Figure 8 shows the total time taken for mining association rules for various minimum support values. It also shows the total number of frequent sets found and the maximum pattern length. For example, at 0.1% minimum support, we found more than 130,000 patterns in just 2.2 seconds; the maximum set length was 20!

Figure 9 shows the total time taken, total number of patterns found, and maximum sequence length, for mining frequent sequences for various minimum support values. For example, at 0.015% minimum support, we found more than 7,000 patterns in just 4.7 seconds; the maximum sequence length was 10.

Finally Figure 10 shows the total time taken, number of patterns, and maximum subtree length, for mining frequent subtrees for various minimum support values. For example, at 0.35% minimum support, we found about 2,300 trees in 6.5 seconds; the maximum subtree had 8 nodes.

These results lead to several interesting observations that support the mining of complex patterns from web logs. For example, if one performs only itemset mining, one discovers many long patterns. Sequence mining takes longer time but the patterns are likely to be more useful, since they contain path information. One has to lower the support compared to set mining, since the same set of nodes can lead to many maximal forward sequences. However, tree mining, tough it takes more time than sequence mining, produces relatively fewer patterns which are even more informative. As in sequence mining, an itemset can contain contain several subtrees, since there is exactly one itemset per user session, but there could be several subtrees from the same session. Furthermore, one frequent subtree many correspond to several maximal forward sequences (as many as the number of leafs in the tree).

Source	Raw Logs		LOGML		#Requests	#Sessions	LOGML Breakdown		
	Regular	Compressed	Regular	Compressed			Webgraph	UserSessions	Other
RPI1(14Jun01)	52,428,686	5,544,951	19,850,273	2,103,298	275726	5891	88.3%	8.3%	3.4%
RPI2(15Jun01)	52,428,742	5,457,778	19,485,594	2,063,934	275061	5436	88.2%	8.4%	3.4%
CS1 (28Jun01)	10,506,256	1,065,771	4,633,113	520,290	51950	2153	74.6%	16.7%	8.7%
CS2 (29Jun01)	10,323,505	1,089,098	5,269,929	580,146	49378	2063	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. The same trends are observed for the other sources.

The potential benefits of LOGML for web usage mining 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.

6. 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. LOGML is not only a preprocessor for our data mining applications, but also useful for web characterization and report generation.

Future work includes mining user graphs (structural information of web usages), as well as visualization of mined data using WWWPal system [7]. 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.

The LOGML 1.0 and XGMML 1.0 draft specifications, with their respective DTDs, Schemas and other details are available online at:

http://www.cs.rpi.edu/∼puninj/LOGML/
http://www.cs.rpi.edu/∼puninj/XGMML/

7. REFERENCES

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, AAAI Press, 1996.

[2] M. Chen, J. Park, and P. Yu. Data mining for path traversal patterns in a web environment. In *International Conference on Distributed Computing Systems*, 1996.

[3] R. Cooley, B. Mobasher, and J. Srivastava. Data preparation for mining world wide web browsing pattern. *Knowledge and Information Systems*, 1(1), 1999.

[4] R. Kosala and H. Blockeel. Web mining research: A survey. *SIGKDD Explorations*, 2(1), June 2000.

[5] B. Masand and M. Spiliopoulou, editors. *Advances in Web Usage Mining and User Profiling*. LNAI 1836. Springer Verlag, July 2000.

[6] P. Pirolli, J. Pitkow, and R. Rao. Silk from a Sow's Ear: Extracting Usable Structure from the Web. In *Conference on Human Factors in Computing Systems*, Apr. 1996.

[7] J. Punin and M. Krishnamoorthy. WWWPal System - A System for Analysis and Synthesis of Web Pages. In *WebNet 98 Conference*, Nov. 1998.

[8] R. Cooley, B. Mobasher, and J. Srivastava. Web Mining: Information and Pattern Discovery on the World Wide Web. In *8th IEEE Intl. Conf. on Tools with AI*, 1997.

[9] M. Spiliopoulou and L. Faulstich. WUM: A Tool for Web Utilization Analysis. In *EDBT Workshop WebDB'98, LNCS 1590*. Springer Verlag, Mar. 1998.

[10] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *5th Intl. Conf. Extending Database Technology*, Mar. 1996.

[11] M. Thüring, J. Hannemann, and J. Haake. Hypermedia and cognition: Designing for comprehension. *Communications of the ACM*, 38(8):57–66, Aug. 1995.

[12] K. Wu, P. Yu, A. Ballman. Speed Tracer: Web usage mining and analysis tool. *Internet Computing*, 37(1):89, 1997.

[13] M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372-390, May-June 2000.

[14] M. J. Zaki. Efficiently mining trees in a forest. Tech. Report 01-7, CS Dept., Rensselaer Polytechnic Institute, July 2001.

[15] M. J. Zaki. SPADE: Efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42(1), Jan 2001.

[16] M. J. Zaki and C.-J. Hsiao. CHARM: An efficient algorithm for closed association rule mining. Technical Report 99-10, CS Dept., Rensselaer Polytechnic Institute, October 1999.