Knowledge Discovery from Users Web-Page Navigation*

Cyrus Shahabi†, Amir M. Zarkesh‡, Jafar Adibi†, and Vishal Shah†

† Integrated Media Systems Center and
Computer Science Department
University of Southern California
Los Angeles, California 90089
[shahabi, adibi, vishalsh]@usc.edu

‡ Quad Design Technology
Camarillo, California 93010
azarkesh@qdt.com

Abstract

We propose to detect users navigation paths to the advantage of web-site owners. First, we explain the design and implementation of a profiler which captures client's selected links and pages order, accurate page viewing time and cache references, using a Java based remote agent. The information captured by the profiler is then utilized by a knowledge discovery technique to cluster users with similar interests. We introduce a novel path clustering method based on the similarity of the history of user navigation. This approach is capable of capturing the interests of the user which could persist through several subsequent hypertext link selections. Finally, we evaluate our path clustering technique via a simulation study on a sample WWW-site. We show that depending on the level of inserted noise, we can recover the correct clusters by %10-%27 of average error margin.

1. Introduction

The orthodox view on the web-pages assumes a uni-directional flow of information from the server to the client. In this view, the flow of information in the other direction is possible only if the client chooses to respond through e-mail or web-page forms. The previous studies on the design of the web-pages are also concentrated on how to make this uni-lateral flow more efficient to the client. In this paper, we discuss a broader view in which the servers also could continuously receive useful information from the clients.

Capturing the characteristics of the users of a business web site is an important task for their marketing department. The *navigation path* of the web-page users, if available to the server, carries valuable information about the users interests. In this paper, we propose a technique to detect the navigation path rather accurately up to the restrictions set by the current Internet security protocols. Our technique detects *link* hits and viewing time per page. This is a superset of information on the order of page hits. It basically encodes the entire user navigation path. Our profiling approach has crucial advantages over existing commercial and academic profilers. The main advantages include: 1) it requires no modification at the client site (e.g., hacking the browser source code), 2) it complies with the current state of the HTTP protocol and its packet structure (i.e., requires no modification and/or extension to the protocol), and 3) it does not require any manual modification at the server side, as the entire process is automated (i.e., the one-time process of modifying a large web-site to make it available for profiling takes less than an hour). Subsequently, a systematic approach, called *path-mining* [24], is explained. This approach is well suited to capture similarity among the orders of the accessed pages in the navigation paths. The merit of our approach in probing the similarity among users interests is shown in a detailed analysis of a test site in Sec. 5.

We start by describing the structure of users navigation profile and set the notation in Sec. 2. The structure of a sample site is also described there. In Sec. 3, we discuss the challenges involve in the construction of the user profile: accurate capturing of the web-page viewing time, detecting client cache hits, and obtaining hypertext links information. We describe the design of our profiler as a *remote agent* which can accurately construct the users profiles. In Sec. 4, we discuss a systematic method to cluster the users navigation paths. First a similarity measure on the feature space of the navigation paths is presented. This similarity measure is capable to detect similarity in the order of the navigation links. The behavior of this measure is shown in a detail analysis of some paths in the sample site. As the second step, many

*This research has been funded in part by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center with additional support from the Annenberg Center for Communication at the University of Southern California and the California Trade and Commerce Agency.

classical data mining approach can be used for the classification based on the given similarity metric. In Sec. 5, a big set of paths in the example web site is analyzed in details and the performance of the similarity measure is tested. Sec. 6 provides a conclusion and overview on the future works.

2. Structure of the Users Navigation Profile

The most detailed information we could gather from the Internet browsing of clients, is the list of links (s)he selected and the elapsed time between them. All other type of information, from the number of hits per page to the complete path profile and the time spent on a path, could be derived from the above data. In this section we formalize this structure and set the definitions. Moreover, the structure of a sample site is introduced. The sample site is used for the rest of this paper to illustrate the structure and efficiency of our approach.

The structure of an Internet server site could be abstracted as following. Let us denote the set of all hyper-text links exist in the WWW-pages of a site as $\mathcal{L} = \{l_1, \ldots l_N\}$. In the view of our applications, it is natural to consider a WWW-page as the set of all links in that page. Therefore, the set of all WWW-pages of a site, $\mathcal{P} = \{p_1, \ldots, p_n\}$, could be realized as a partition of $\mathcal{L}$ to the equivalent classes. Here links l_i and l_j are equivalent, *i.e.* $l_i \sim l_j$, if they are in the same page. Then $\mathcal{P}$ could be defined as

$$\mathcal{P} = \mathcal{L}/\sim . \tag{1}$$

On the other hand a graph structure could be recognized. Each link has a *starting* and an *ending* page. For some of the links starting points or the ending points could be on some page outside the site $\mathcal{P}$. Moreover it is possible to have links from a page to itself. A navigation path could be shown as a sequence of links. Formally, we are dealing with paths on a directed graph. The nodes of the graph are pages and the links are the WWW-hypertext links. We call this graph a site *connectivity graph*. Any page of a WWW-site could be potentially called from a page in some other site. Therefore the portion of navigation path which is inside the site could be started from any node. Similarly, any page could have links which points to the pages in other sites. In this paper, we are interested only in the part of clients navigation path inside the site. These paths are made only from the links in $\mathcal{L}$.

A sample site for a hypothetical entertainment center have been developed. The connectivity graph for the sample site is shown in Fig. 1. The names of the nodes are the title of the pages. The names of the links are the click-able hypertext, corresponding to those links. As it can be seen in Fig. 1, the connectivity graph could be quiet complicated. A subset of links in the sample site is shown in Fig. 2. If the site is well designed, navigation can be very informative. Only a few links to the outside sites are shown in Fig. 1.

Local caching at the client browser makes it possible for the client to jump back to some of the pages (s)he already visited. Each browser can have different caching strategy which gives ability to jump to all or part of the pages which are visited. Moreover, the possible jumps from each page dynamically depend on the history of the navigation of that specific client to reach to that page. Therefore the cache jumps are not shown in the connectivity graph. However, detection of the cache hits is important and a systematic method for this is presented in Sec. 3.

It is crucial to note that the sequence of pages in a path are not enough to accurately describe a client navigation path. This is due to the fact that two different links could have identical starting and ending pages. Knowing which one of these similar links is selected can be informative because of different context in which it is appeared. To illustrate, consider $News\overrightarrow{Evita}Evita$ and $News\overrightarrow{Argentina}Evita$ in our sample site (see Fig. 2). Choosing $\overrightarrow{Evita}$ can be due to the interest in Evita because of its box-office success. However, choosing $\overrightarrow{Argentina}$ can be due to the interest in the news about controversial screening of Evita in Argentina.

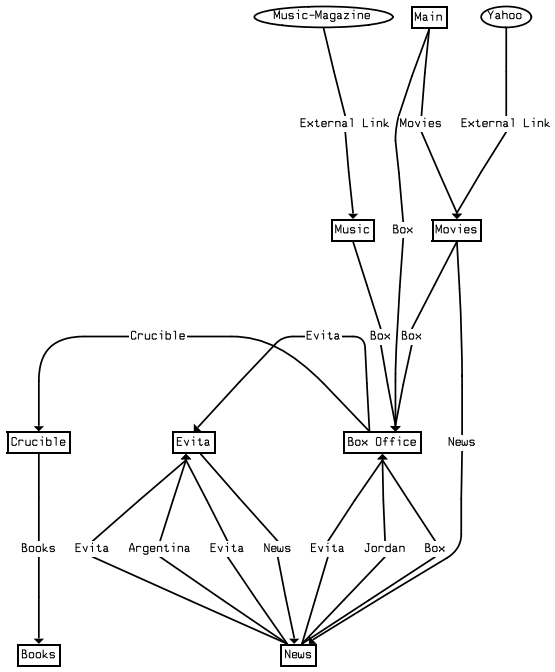

Figure 2. Simplified Graph

An important feature of the users navigation paths is the time they spend on different pages. We assign the time spent on each page by a user, to the link (s)he selected to leave that page. This should be the time that the user spent $viewing$ the page and it should exclude the time spent on receiving and loading the page (see Sec. 3). We assign the viewing time of the last page to the last page itself. If a user remains

on a page for a considerably long time before selecting the next link, we consider this as a sign that the client was not reading the page for the entire time. A time-out mechanism is explained in Sec. 3 that considers this page as the ending point of the client path. The next move of the client after the long pause is considered as the starting point of a new path.

We define the profile information of a user as the set of the links of his(er) navigation and the corresponding link times. We show the links sequence of the path traversed by the user u as $s_u = (l_{i_1}, \ldots, l_{i_n})$, the corresponding links time sequence as $T_u = (T_u(l_{i_1}), \ldots, T_u(l_{i_n}))$ and the user profile as pair of sequences, $(s_u; T_u)$.

Some examples of paths on the sample site (Fig. 2) are given in the following:

1. *Main* $\overrightarrow{Movies : 20sec}$ *Movies* $\overrightarrow{News : 15sec}$ *News* $\overrightarrow{Box : 43sec}$ *Box Office* $\overrightarrow{Evita : 52sec}$ *News* $\overrightarrow{Argentina : 31sec}$ *Evita: 44 sec*

2. *Music* $\overrightarrow{Box : 11sec}$ *Box Office* $\overrightarrow{Crucible : 12sec}$ *Crucible* $\overrightarrow{Books : 13sec}$ *Books:19 sec*

3. *Main* $\overrightarrow{Movies : 33sec}$ *Movies* $\overrightarrow{Box : 21sec}$ *Box Office* $\overrightarrow{Evita : 44sec}$ *News* $\overrightarrow{Box : 53sec}$ *Box Office* $\overrightarrow{Evita : 61sec}$ *Evita :31 sec*

4. *Main* $\overrightarrow{Movies : 19sec}$ *Movies* $\overrightarrow{News : 21sec}$ *News* $\overrightarrow{Box : 38sec}$ *Box Office* $\overrightarrow{Evita : 61sec}$ *News* $\overrightarrow{Evita : 24sec}$ *Evita* $\overrightarrow{News : 31sec}$ *News* $\overrightarrow{Argentina : 19sec}$ *Evita:39 sec*

5. *Movies* $\overrightarrow{Box : 32sec}$ *Box Office* $\overrightarrow{News : 17sec}$ *News* $\overrightarrow{Jordan : 64sec}$ *Box Office* $\overrightarrow{Evita : 19sec}$ *Evita:50*

6. *Main* $\overrightarrow{Box : 17sec}$ *Box Office* $\overrightarrow{Evita : 33sec}$ *News* $\overrightarrow{Box : 41sec}$ *Box Office* $\overrightarrow{Evita : 54sec}$ *Evita* $\overrightarrow{News : 56sec}$ *News: 47*

These examples are used in the following sections to explain different aspects of our analysis.

3. Design and Implementation of a Profiler

For the purpose of recording users' path profile, we designed and implemented a profiler. A simple profiler can be a counter which counts the number of accesses to a web-page. A sample profiler can be found in[13] which is written in Perl script and provides a comprehensive view of daily accesses for the web-site. Due to the specific requirements of our analyzer (see Sec. 4), we require to capture more information than those captured by previous profilers (e.g., [23]).

Many research studies [13, 23, 5] and some commercial products [2, 1] have looked at capturing users' web access patterns and store them in log files for different purposes. Our profiler is distinguishable from all those studies due to its following main characteristics. Our profiler is executed at the client site instead of the server site. While this results in more accurate information gathered from the client (e.g., client cache hits, precise page viewing time), its implementation is very challenging. We achieved this by implementing a Java applet which its details are sketched in Sec. 3.1.2. Note that in [5], they also gathered accurate information about the client access. They achieved that by modifying the web-browser. Our profiler, however, requires no modification at the client site (including the browser), and it does not rely on user cooperation. In addition, no modification needs to be done in the current state of HTTP protocol and/or its packet structure. It is only required to modify the web-pages at the server site. This modification can be done automatically by some kind of a parser. That is, we define a new phase between the time that a web-page is constructed and the time it is made available through Internet. During this intermediate phase every page is parsed and modified automatically. First, a call to a Java applet is added to every page (see Sec. 3.1.2 for more details). Second, all the hypertext links are modified so that by clicking on them, more information will be transfered to the server site (see Sec. 3.3). A demo of our profiler is available on $http://www.usc.edu/dept/imsc/profiler.html$. For the rest of this section, we describe the design of our profiler.

For each user, our profiler should capture the ordered list of selected links and the viewing time of each page. The information content of our profile is a superset of statistical information such as frequency of access to a page and a link. To generate profile sequence accurately, the profiler needs to overcome three interesting challenges: 1) accurate recording of the time spent by client viewing a page, 2) detecting a page access at the server site, should the client observes a hit in its local cache, and 3) detecting the links traversed by the clients. For the rest of this section we describe these challenges and our proposed solutions.

3.1 Viewing Time

The time spent by a user viewing a page is a very important piece of information that can be employed to measure user's interest in the page. However, accurate recording of the viewing time is not trivial. To illustrate, consider Fig. 3. This figure demonstrates a client interactions with the server from the time the user requests a page (time t_0) until the time the server receives the user's subsequent request for another page (time t_5). We are interested in the duration $t_4 - t_3$ that is termed *viewing time.* Note that the difference between t_1

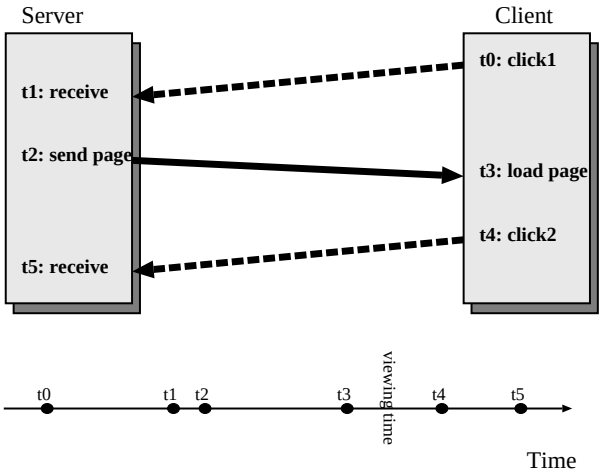

Figure 3. Client/Server Interactions

and t_2 is in the order of a fraction of milli-second and can be ignored (i.e., $t_1 \approx t_2$). Currently, different web servers such as NCSA [12] and Apache [14] provide us with the flexibility to obtain t_1 and t_5 (actually t_2 and t_6) as the times the request was served. They store this information in Common [19] or Extended Log File [21] format. However, for the purpose of our analysis we need times t_0 and t_4. In addition, we need time t_3 to compute viewing time accurately.

Another argument to question the accuracy of viewing time information is that the user might load a page and then leave. Hence, even $t_4 - t_3$ is not the real time spent on viewing the page. One method to compensate is to compute a threshold based on the page size and reading speed of humans. If the viewing time exceeded the threshold, then the time information for that page must be invalidated and the threshold can be used as the viewing time. An alternative is to use smart cameras [18] to determine if the user is really viewing the page. This of course has privacy issues which might only be acceptable for certain applications (such as educational applications).

To illustrate the problem of using inaccurate viewing time, consider the following example. Suppose a clustering algorithm clusters users based on the time they spend viewing a page (as in [23]) and the profiler assumes $t_5 - t_1$ as the viewing time. The clustering algorithm will end up classify users based on the network traffic. For example, at 9:00am suddenly all the users show interest on page i while the reality is that the network was congested and it took longer to load the next page from page i for most of the users.

We propose two techniques to compute viewing time. The first one is based on a server-site profiler and has the disadvantage of using approximation. The second approach is a client-site profiler which is more accurate but it is possible for a user to disable it due to tight security considerations.

3.1.1 Server Site Profiler

If the profiler is running at the server site, it can trivially obtain t_1, t_2 and t_5. The server can then execute a *ping* command and measure its round-trip time (say δ). Subsequently, $(t_1 - t_0)$ (or $(t_5 - t_4)$) can be approximated as $\frac{\delta}{2}$. Using ping is appropriate because it incorporates the network congestion at the time of request. In other words δ is a function of network congestion. Finally, $(t_3 - t_2)$ can be approximated as $PageSize * \frac{\delta}{2}$, where $PageSize$ is the size of the page in number of TCP/IP packets. Having the above durations, the computation of $t_4 - t_3$ is straightforward.

Note that this method is relying heavily on approximation because some metrics such as the server load has not been considered. Consequently, the estimated viewing time might not be accurate. In the following section, we describe a more accurate method to capture the viewing time.

3.1.2 Remote Agent

If the profiler can be executed at the client site, then the measurement of viewing time can be done much more accurately. In this section we propose a method to run a *remote agent* at the client site without violating the security of the user. Our remote agent is implemented in Java [8] language. We developed a Java applet which is loaded into the client machine only once when the first page of our server has been accessed. Subsequently, every time a new HTML page (say page A) is loaded at the client display, the applet will send the system time as ($T_{load}(A)$) to the server. Similarly, once the page is unloaded, again the system time will be reported to the server as ($T_{exit}(A)$). Trivially, by deducting $T_{load}(A)$ from $T_{exit}(A)$, the server can compute the exact viewing time for page A.

The only disadvantage of this method is that the loading time of the applet is considered as part of the viewing time of the first page. After that, the applet becomes resident in the client's cache and no more loading time will be encountered. The loading time is unpredictable because it depends on various parameters such as connection speed, network congestion, number of classes being downloaded, speed of the just-in-time compiler etc. and varies from one platform/browser to the other. To resolve this problem, we investigated an alternative solution based on JavaScript [3] language which is a compact, object-based scripting language for developing client/server Internet applications. In comparison with Java applet, this implementation consumes very little or no time in interpretation as the JavaScript is a lightweight language. However, we abandoned the JavaScript implementation because the only time that the code can capture and report T_{exit}, was when the user clicks on a hypertext link. Instead, if the user clicks on hot-key buttons on the browser (e.g., $back$, $forward$ or go buttons in Netscape) for navigation, the JavaScript code cannot capture and report T_{exit} to

the server. Hence, we select the Java applet implementation and ignore the loading time of the applet for the first page.

To capture T_{load} and T_{exit} by the Java applet, each HTML page should be modified to incorporate a call to the applet. The following are sample statements that should be added (automatically) to the beginning of every HTML page ("index.html" page in this example):

```
<APPLET CODEBASE="/java"
 CODE="ViewTime" WIDTH=1 HEIGHT=1>
 <PARAM NAME="PAGE_NAME"
  VALUE="http://imsc.usc.edu/index.html">
</APPLET>
```

Incorporating the above statements at the beginning of the page results in invocation of the applet "ViewTime" immediately when the page is loaded. Subsequently, the applet stops execution when the page is unloaded.

T_{load} and T_{exit} are captured by the applet using the *currentTimeMillis()* method from *java.lang.System*. We implemented our own class to send T_{load} and T_{exit} to the server using *java.net.Socket* class. The submissions of T_{load} and T_{exit} to the server are incorporated into the *start* and *stop* methods of *java.applet.Applet* class, respectively. The *start* method is called each time the applet is revisited in the web-page. The *stop* method is called when the web-page that contains this applet has been replaced by another page and also just before the applet is to be destroyed. In addition to T_{load} and T_{exit}, the applet also sends the URL of the loaded page. This information is utilized later to detect client cache hits (see Sec. 3.2).

At the server site, a Perl script is running that will gather all the information sent by the client (e.g., T_{load}, T_{exit}) and record it as the user profile. This information will be analyzed by our path clustering techniques as described in Sec. 4. The major drawback of the remote agent is that some users might disable Java from their browser due to security concerns.

3.2 Client Cache

Typically, web browsers employ some techniques to cache the pages that has recently been accessed. If the user decides to return to an already accessed page, the server will not be notified and hence a server site profiler cannot record this information. There are many alternative methods to detect cache hits.

One method is to assign a short expiration time to HTML pages, enforcing the browser to retrieve every page from the server (and hence notifying the server). This can be done using *Expires* entity-header field in the HTTP protocol. This allows the web server to select a date after which the information may no longer be valid. One can set the date to value of zero (0) or an invalid date format that results in immediate expiration of a page after its retrieval. Alternatively the user can also set the browser cache size to zero. However, this requires the user cooperation and cannot be enforced. The obvious disadvantage of both of the above methods is the performance degradation resulted from observing many cache misses.

An alternative method to capture references to cached pages is by doing some kind of detective work. This method is based on the fact that the links of a path should be consistent, i.e., the ending page of each link should be the same as the starting page of the next link. Violation of this rule in a user profile is the sign of local cache usage. This can be best shown by an example. Assume the following access pattern for a user: p_i, p_j, p_i, and p_k. However, due to some caching mechanism, the server is only notified of the following pattern: p_i, p_j, and p_k. This is because the second access to page p_i observed a hit at the client's cache. The server can analyze the access pattern and assuming there is no link from p_j to p_k (which is a likely true assumption; otherwise, the user could have directly used that link to load p_k) detect that the user has loaded p_i first and then p_k. We developed a heuristic to generate as accurately as possible the real access pattern from reported access patterns. In our heuristic method, we use the Referrer request-header field in HTTP protocol [15]. However, due to lack of space and since our alternative method to capture cache hits (described in the following paragraphs) is more reliable and accurate, we do not elaborate more on this heuristic method. Note that another serious shortcoming of our heuristic method is that it might not be able to detect the link sequences. To illustrate, in the previous example, if there are two different links from p_j to p_i, the server cannot find out which link was selected. This is because no report about this access was sent to the server. The situation becomes worse if the client uses hot-key buttons on the browser (e.g., $back$ button in Netscape).

In contrast to all the above methods, our remote agent design can capture cache hits simply and accurately. Recall from Sec. 3.1.2, that for every HTML page, either loaded from cache or sent by the server, the "ViewTime" applet submit T_{load} and T_{exit} to the server. Hence, if the server receives T_{load} for a page that has not been requested (say page A), it can interpret it as a cache hit for page A. Note that this technique is independent of how the cached page was referenced, i.e., by using hot-key buttons or by directly clicking on hypertext links.

3.3 Traversed Links

One of the major differences between this approach (see Sec. 4) and previous studies is that we consider the order of page accesses. Our profiler has even made one more step forward and captures the links the client has selected to ac-

cess pages. To illustrate, in Fig. 2, the user can select either $\overrightarrow{Argentina}$ or $\overrightarrow{Evita}$ hypertext links to navigate from the *News* web-page to the *Evita* web-page. In order for our path clustering algorithm to differentiate between these two users, our profiler should be able to capture the link information. This becomes even more challenging when the links have both identical names and target pages, but appear in different contexts. For example, in Fig. 2, there are two links with the identical name of $\overrightarrow{Evita}$, but in two different contexts (i.e. a "song" context and an "Oscar" context).

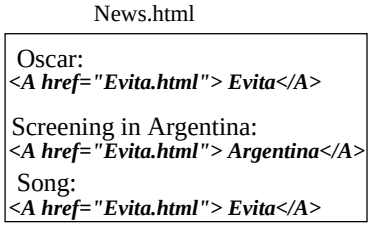

Figure 4. The original $News$ web-page

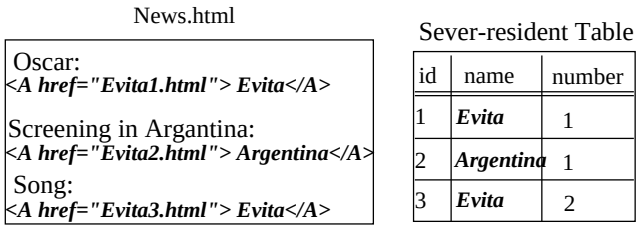

Figure 5. The modified $News$ web-page

Currently, only the URL of the page requested by the client is passed to the server. This information is not sufficient to distinguish between links pointing to the same page (either with or without identical names). Briefly, our profiler extends the URL address of the pages with a *link identifier*. The link identifier is an index to a server resident table whose rows contain link names and numbers.

As mentioned before, we add a new phase between the time that a page is constructed and the time that it is made available through Internet. During this intermediate phase each page is parsed and modified automatically. For example, Fig. 4 shows the structure of the $News$ web-page prior to applying the intermediate phase. Subsequently, Fig. 5 depicts the result of applying the intermediate phase on the $News$ web-page. Hence, the three references to *Evita.html* are augmented with three different link identifiers. In addition, a table is generated that is indexed by link identifiers. Each row of the table contains a link identifier, its corresponding reference name and the number of occurance of the reference name in the page. Now the page is ready to become available on the Internet. That is, if a user clicks on "Argentina", the target page, *Evita2.html*, will be passed to the server. The server decomposes *Evita2.html* into the target page address, *Evita.html*, and the link identifier 2. From the link identifier, the server retrieves the 2nd row of the index table and realizes that the link "Argentina" was selected. Subsequently, it sends the target page *Evita.html* to the client and record the following access pattern for the user: *News.html* $\overrightarrow{Argentina}$ *Evita.html*. Note that without the above steps, the server could have only recorded *News.html Evita.html* access pattern. Similarly, a user interested in Oscar nominations who selects the first occurance of "Evita" to access $Evita.html$ can be distinguished from the other user interested in music who selects the second occurance of "Evita". The server will record *News.html* $\overrightarrow{Evita * 1}$ *Evita.html* as the access pattern of the first user and *News.html* $\overrightarrow{Evita * 2}$ *Evita.html* as the access pattern of the second one.

Since the above technique requires no modification at the client site, it can be employed by both server or client site profilers. We have incorporated this technique into our remote agent design (see Sec. 3.1.2).

4. Knowledge Discovery from Users Profile

The purpose of knowledge discovery from users profile, is to find clusters of similar interests among the users. If the site is well designed, there will be strong correlation among the similarity of the navigation paths and similarity among the users interest. Therefore, clustering of the former could be used to cluster the latter. In the following section we show this correlation using many examples from our sample site. We call WWW-sites with high correlation between users interests and their navigation path a *Server Informative WWW-site*. In [25] some design guidelines to construct Server Informative web-sites are explained.

The basic question is: what do we mean by calling two paths *similar*? Similarity among the navigation paths should be based on some of their features. The definition of the similarity is application dependent. Here we provide an overview on a powerful path clustering method called *path-mining*[24]. This approach is suitable for knowledge discovery in databases with partial ordering in their data. In this method, first a general *path feature space* is characterized. Then a *similarity* measure among the paths over the *feature space* is introduced. Finally this similarity measure is used in the clustering purposes. For more in-depth analysis of the path-mining approach and its other applications consult [24]. Here we cover different aspect of path-mining in the context of the WWW-site navigation analysis.

4.1 Path Feature Space

Defining a similarity measure among paths is not straightforward. This is due to the fact that we need to measure the distance between paths with different length and/or different starting pages. Moreover, paths are defined on a directed graph that can potentially have many cycles.

We consider a space of path features which is rich enough to capture important features of a path but yet has a simple underline structure. Consider a path s consisting of n links. We call s a n-*hop* path. Let's define $\mathcal{S}_m$ as the set of all possible m-hop sub-paths of s. Therefore, $\mathcal{S}_m(s)$ has $n-m+1$ elements. As a convention a page is considered as a 0-hop path. The set of m-hop sub-paths contains all possible orders of m connected links in the path. Note that a cyclic path includes some of its sub-paths more than once. The union of all $\mathcal{S}_m(s)$'s for all $0 \leq m \leq n$, which is shown by $\mathcal{S}(s)$, is called the *feature space* of path s. Note that path s itself belongs to $\mathcal{S}(s)$.

To illustrate, consider the first and the third path in the example of Sec. 2. The feature space which is embedding these two paths is the union set of all sub-paths of these two paths. This set of features is listed in the following (only up to : 2-hops)

- 0-hops: *Main, Movies, News, Box Office, Evita*
- 1-hops: *Main* $\overrightarrow{Movies}$ *Movies, Movies* $\overrightarrow{News}$ *News, News* $\overrightarrow{Box}$ *Box Office, Box Office* $\overrightarrow{Evita}$ *News, News* $\overrightarrow{Argentina}$ *Evita, Box Office* $\overrightarrow{Evita}$ *Evita*
- 2-hops: *Main* $\overrightarrow{Movies}$ *Movies* $\overrightarrow{News}$ *News, Movies* $\overrightarrow{News}$ *News* $\overrightarrow{Box}$ *Box Office, News* $\overrightarrow{Box}$ *Box Office* $\overrightarrow{News}$ *News, Box Office* $\overrightarrow{News}$ *News* $\overrightarrow{Argentina}$ *Evita, Main* $\overrightarrow{Movies}$ *Movies* $\overrightarrow{Box}$ *Box Office, Box Office* $\overrightarrow{News}$ *News* $\overrightarrow{Box}$ *Box Office, News* $\overrightarrow{Box}$ *Box office* $\overrightarrow{Evita}$ *Evita*

Considering all possible m-hops up to 5-hops, the feature space of these two paths includes 30 sub-paths.

The time spent over a sub-path is simply defined by the sum of the times spent on the links of that sub-path. As we mentioned in Sec. 2, the link time is defined by the viewing time on the end page of that link. The total viewing for the user u on the sub-path s is denoted as $T_u(s)$.

4.2 Path Angles and Path Clustering

A natural *angle* among the paths can be constructed by using an *inner product* over the feature space[24]. The angle among the navigation paths s_1 and s_2 over sub-paths with length m is given by

$$\cos(\theta_{m;s_1,s_2}) = \frac{< s_1, s_2 >_m}{(< s_1, s_1 >_m)^{\frac{1}{2}} (< s_2, s_2 >_m)^{\frac{1}{2}}} \quad (2)$$

where the *inner product* over sub-paths with length m for the paths s_1 and s_2 is defined by

$$< (s_1; T_{u_1}), (s_2; T_{u_2}) >_m = \sum_{k=0}^{m} \sum_{s \in \mathcal{S}_k(s_1) \cap \mathcal{S}_k(s_2)} T_{u_1}(s) \times T_{u_2}(s). \quad (3)$$

Based on the above definitions, the paths with no common page are *perpendicular*. Also each path has zero angle with itself. This angle can be employed as a natural *similarity* measure. Note that above definition works for any non-negative integers n_1, n_2 and m, where n_1 and n_2 are the lengths of s_1 and s_2, respectively. For $m \geq \min(n_1, n_2)$ all the inner-products are equal and we drop the m index.

Here we investigate the sample paths given in the Sec. 2 to check if the path angle measure predicts reasonable results. A simple calculation based on Eq. 3 produces the path angle among every pair of the six paths given in that example. The result is shown in the following path angle matrix.

$$\cos(\theta_{s_1,s_2}) = \begin{pmatrix} 1 & 0.10 & 0.250 & 0.319 & 0.031 & 0.190 \\ 0.010 & 1 & 0.019 & 0.010 & 0.010 & 0.004 \\ 0.250 & 0.019 & 1 & 0.010 & 0.120 & 0.306 \\ 0.319 & 0.010 & 0.010 & 1 & 0.072 & 0.105 \\ 0.031 & 0.010 & 0.120 & 0.072 & 1 & 0.107 \\ 0.190 & 0.004 & 0.306 & 0.105 & 0.107 & 1 \end{pmatrix} \quad (4)$$

Up to here we showed how to calculate the similarity between each pairs of the paths. Hereafter, using the path angle as a measure for similarity, we can apply a handful of algorithms in the classical theory of data mining[9, 6, 10]. In the next section we use K-means algorithm to classify a large number of paths on our sample site. For the purpose of illustration, henceforth a simple threshold method to classify our six example paths are employed.

Knowing the number of desired classes we could determine a threshold angle θ_{th} to split the classes. Paths with angels less than the threshold angle are considered in the same class. A membership matrix could be constructed by changing elements of the similarity matrix to ones or zeros depending if they are larger or smaller than the threshold. For a threshold angle in the following range

$$0.250 < \cos(\theta_{th}) < 0.306 \quad (5)$$

the membership matrix for our example is

$$M_{s_1,s_2} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \quad (6)$$

In this matrix, a 1 in row i and column j suggests similarity between paths i and j. We could transform this membership matrix to a block diagonal matrix with simple swapping of the rows and columns (see Tab.1). From Tab.1, we can clas-

Path ID	1	4	3	6	2	5
1	1	1	0	0	0	0
4	1	1	0	0	0	0
3	0	0	1	1	0	0
6	0	0	1	1	0	0
2	0	0	0	0	1	0
5	0	0	0	0	0	1

Table 1. Results after clustering

sify our sample paths to the following four classes: path 1 and 4 are in the first class, paths 3 and 6 are in the second class. Paths 2 and 5 each make an isolated class.

The power of path-mining algorithm is in probing the order of the links. For example, consider paths 1 and 3. Although, these two paths visit the same pages, path-mining did not cluster them in the same group. This is because they have visited different links and/or different links order. Other methods in which similarity function is only based on visiting similar pages[23], are not able to address this issue. Notice the distinction among path 1 and 3 is mapped to a distinction between the interests of the two users. User with path 1 may be interested in news about Evita while user with path 3 is interested in Evita because of the box office records.

Two paths are considered perpendicular if they have no link or page in common. For example the angle between path 2 and 6 is close to 90 since they have only one page in common. Examine paths 4 and 1 to see how path-mining algorithm handles path with cycles. These examples indicate that clustering based on the path angles follows an intuitive clustering. Fast algorithms in finding the path angles are important when the graph size is very large. For a detailed analysis consult[24].

5. Performance Evaluation

To evaluate the merit of our profiler and path-mining algorithm, we conducted a two step experiment on our sample web-site shown in Fig. 1. The parameters of this site are reported in Table 2. First we chose ten users to surf on the sample web-site. Using our profiler the links and their corresponding viewing time were captured. The profiler output was checked against a direct measurement of the users activities. The profiler recorded the viewing time and link access accurately.

Subsequently, we generated from 30 to 150 paths around the ten *nucleus* paths. The mean length of the generated

Number of pages	34
Number of links	136
Avg number of links per page	4
Avg similar target-different links per page	2
Avg similar target-identical links per page	2
Total number of links to outside web pages	11

Table 2. Sample web-site parameters

ID	# of paths in original cluster	Result BF=5%	Error %	Result BF=10%	Error %
1	100	113	13	131	31
2	115	87	24	92	20
3	43	48	11	57	32
4	56	54	3	44	21
5	93	86	8	86	14
6	140	151	7	119	15
7	30	35	17	15	50
8	84	96	14	123	46
9	129	139	7	110	15
10	150	131	7	163	9

Table 3. K-Means clustering Result

paths was 8 *links*. To generate a path s_k around the nucleus path s_n, we employed Markov Chain model as follows. Suppose i, j is a link in s_n and we want to generate the next link after i for s_k. Subsequently, we go to j with 95% probability as opposed to any other node in the graph. We call this a path with branching factor(BF) or noise level $\%5$. Therefore, the probability of generating an 8-hops identical to the nucleus path is about 66%. The other 34% of generated paths are in a range of similar to entirely different from the nucleus path. The generated paths can be different in links, time, and length of a given path.

We report the result of the clustering on paths generated with branching factor (BF) 5% and 10%. A total of 940 paths were generated and fed into our path-mining algorithm. Subsequently, we applied K-Means clustering algorithm [6, 9] on the path angles computed from our path similarity algorithm. The result for K-Means algorithm with $K = 10$ (i.e., 10 clusters) is presented in Table 3. Our experiment shows that the average error margin in recovering the original clusters are $\%10$ and $\%27$ for the branching factors $\%5$ and $\%10$, respectively. Note that due to the impact of inserted noise, some of the generated paths might legitimately belong to some class different than which they have been originated.

6. Conclusions and Future Research Directions

Conventionally, web users access have been captured by profilers to the advantage of the user in order to achieve a smoother navigation or understand the user behavior [10, 17]. In this study, we propose to capture user navigation path to the advantage of web site owner. Capturing this information requires a more elaborated user profiler which has been designed and implemented as part of this study. Traditional profilers only count the frequency of access to a page as well as some data regarding the clients system[13]. Other studies in this area focussed on how to differentiate users who are logging in with the same IP address using *session identifier*[20, 16] in conjunction with identifier timeout mechanism (as mentioned in [23]), which has also been incorporated by our profiler. Some studies mentioned the importance of the page viewing time [23] while no feasible implementation to accurately collect this piece of information was proposed. Our profiler, in addition to the above, captures clients: 1) access page order, 2) link access, 3) cache reference, and 4) accurate page viewing time. We also proposed a design and implementation for a remote agent. The tradeoff between user privacy and servers' requirement to capture user information is currently under investigation by alternative standardization committees [20, 22]. In this study, we assumed the current status of HTTP standard and Java security system.

Next, we implemented the *path-mining* algorithm[24] to cluster the navigation paths detected by the our profiler. This algorithm finds a scalar number as the similarity among the paths. These similarity numbers could be fed to standard data-mining algorithms [7] to cluster the users interests. The advantage of this approach over previous attempts [23] is the utilization of the links orders in addition to the users page access viewing times.

There are many interesting applications that can benefit from the knowledge extracted by our method [11]. Dynamic link generation and pre-fetching the pages have already been mentioned in [23, 4]. Our clustering method provides a better knowledge base for these applications due to path order considerations. Another interesting application is to map the user navigation path data to the answers of a specific questionnaire. Having this done, the marketing division of a business could *implicitly* get the answers to some of its marketing questions just from users navigation on their web-site. We need a systematic web-site design methodology to create new web-pages, or modify existing web-pages, such that different users' navigation patterns could be better mapped to the answers to a set of specific questions. We have already developed some preliminary design rules and our experimental results have been promising and will appear in [25]. This direction involves the use and test of our package with more sophisticated knowledge discovery methods, like *associated rules*.

References

[1] Interse market focus 3 by interse corporation. http://www.interse.com.

[2] Netcount service from pricewaterhouse llp. http://www.netcount.com.

[3] See. http://home.netscape.com/eng/mozilla/Gold/handbook /javascript/index.html.

[4] A. Bestavros. Using speculation to reduce server load and service time on the www. In *Proceedings of CIKM'95: The* 4^{th} *ACM International Conference on Information and Knowledge Management*, Baltimore, Maryland, November 1995.

[5] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of www client-based traces. Technical Report TR-95-010, Boston University, CS Dept, Boston, MA 02215, April 1995.

[6] B. Everitt. *Cluster Analysis*. H-E-B Ltd., 1974.

[7] U. M. Fayyad, G. Piatetsky-Shapiro, G. Smyth, and P. Uthurusamy. *Advances in Knowledge discovery and Data Mining*. AAAI/MIT Press, 1996.

[8] M. A. Hamilton. Java and the shift to net-centric computing. *IEEE Computer*, 29(8):31–39, 1996.

[9] J. Hartigan. *Clustering Algorithms*. New York: John Wiley & Sons Inc., 1975.

[10] H. Lieberman. An agent that assist web browsing. In *Proceding of 14th Int. Joint Confernce on Artificial Intelligence*, pages 924–929, Mnotreal, Canada, 1995.

[11] G. Piatetsky-Shapiro, R. Braachman, T. Khabaza, W. Kloesgen, and E. Simoudis. An overview of issues in developing industrial data mining and knowledge discovery applications. In *Proceeding of The Second Int. Confernce on Knowledge Discovery and Data Mining*, pages 89–95, 1996.

[12] See. http://hoohoo.ncsa.uiuc.edu.

[13] See. http://netpressence.com/accesswatch/.

[14] See. http://www.apache.org.

[15] See. http://www.ics.uci.edu/pub/ietf/http/rfc1945. In *Request For Comments 1945*.

[16] See. http://www.pathfinder.com.

[17] See. http://www..public.iastate.edu/ cyberstacks/aristotle.html.

[18] See. http://www.usc.edu/dept/imsc/smcam.html.

[19] See. http://www.w3.org/pub/WWW/Daemon/User/Config /Logging.html#common_logfile_format.

[20] See. http://www.w3.org/pub/WWW/TR.

[21] See. http://www.w3.org/pub/WWW/TR/WD-logfile.html.

[22] See. http://www.ai.mit.edu/projects/iiip/conferences/survey96 /cfp.html. In *Workshop on Internet Survey,Methodology and Web Demographics*, Cambridge, MA, January 29-30 1996.

[23] T. W.Yan, M. Jacobsen, H. Garcia-Molina, and U. Dayal. From user acess patterns to dynamic hypertext linking. In *Proceedings of the* 5^{th} *International World-Wide Web Conference*, Paris, France, May 1996.

[24] A. Zarkesh and J. Adibi. Pathmining: Knowledge discovery in patialy ordered databases. Submmitted to KDD-97.

[25] A. Zarkesh, J. Adibi, C. Shahabi, and V. Shah. Discovery of the answers to hidden questionnaires based on users web-site navigation. In preparation.

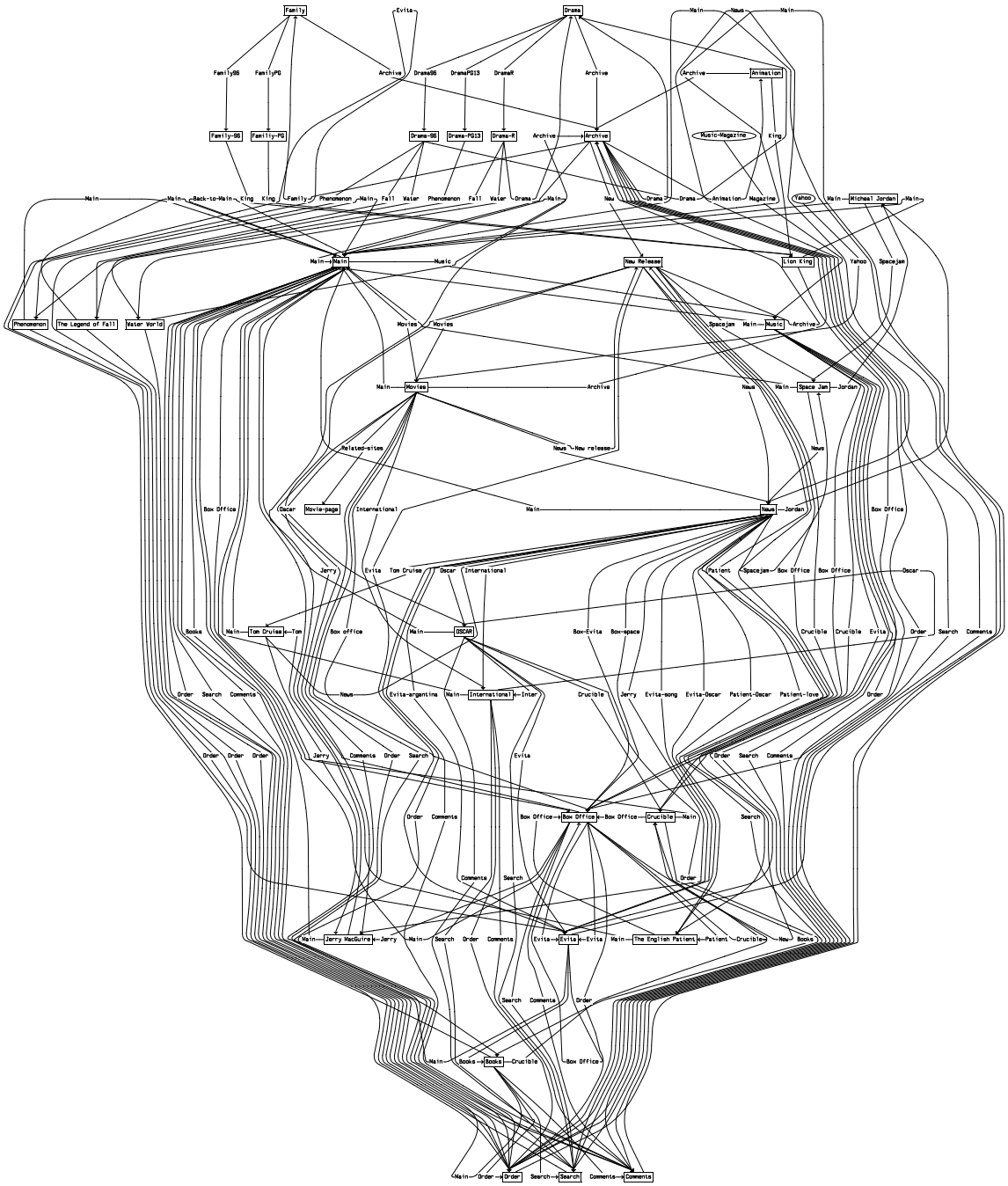

Figure 1. Connectivity Graph for the Sample Site