An Online Recommender System

R. Baraglia[1], F. Silvestri[1,2]

[1] Istituto ISTI, Consiglio Nazionale delle Ricerche (CNR), Pisa, Italy
[2] Dipartimento di Informatica, Università di Pisa, Italy

Abstract One of the most important class of Data Mining applications is the so called "*Web Mining Systems*" which analyzes and extracts important and non-trivial knowledge from Web related data. Typical applications of Web Mining are represented by the personalization or recommender systems. These systems are aimed to extract knowledge from the analysis of historical information of a web server in order to improve the web site expressiveness in terms of readability and content availability. Typically, these systems are made up of two parts. One, which is usually executed off-line, analyzes the server access logs in order to find a suitable categorization, and another, which is usually executed online, classifies the active requests, according to the previous off-line analysis. In this paper we propose *SUGGEST 2.0* a recommender system which differently from previously proposed WUM systems does not make use of an off-line component. Moreover, in the last part of the paper, we analyze the quality of the generated suggestions and the performance of our solution. To this purpose we also introduce a new quality metric which try to estimate the effectiveness of a recommender system as the capacity to anticipate users' requests that will be issued farther in the future[1].

1 Introduction

One of the most important class of Data Mining (DM) application is represented by Web Mining. The main concern of Web Mining is to discover information "hidden" into Web-related data. More specifically, Web Usage Mining (WUM) is the process of extracting knowledge from Web users access data (also called *clickstreams*), by exploiting Data Mining technologies.

A typical application of WUM is represented by the so called *recommender* systems. The main goal of this kind of system is to improve a Web site usability.

A recommender system is usually structured as a two staged process [3], [10], [4], and [1]. The first stage analyzes the historical data recorded in a Web server log and builds a knowledge base that will be used in the next step for the generation of personalized content.

[1] This work was funded by the Italian Ministry of Education, University and Research (MIUR) as part of the National Project Legge 449/97, 1999, settore Società dell'Informazione: Technologies and Services for Enhanced Contents Delivery (2002-2004).

The first module, usually executed *off-line* with respect to the Web server normal operations, proceeds by analyzing the historical data. The functions which are carried out during the first stage are: *Preprocessing*, such as data cleaning, and session identification; and *Pattern Discovery* using DM techniques, like association rules, sequential patterns, clustering or classification with the goal of building the system's knowledge base.

The second component, which is usually executed *on-line* with respect to the normal Web server operations, makes use of this extracted knowledge to enrich the content of a Web site in several forms: links to pages, advertisements, hints regarding products or service retained to be interesting for the current user.

In the past, several WUM projects have been proposed to foresee users' preference and their navigation behavior, as well as many recent results improved separately the quality of the personalization or the user profiling phase [6], [5], [2].

Analog [10] is one of the first WUM systems proposed. It is structured according to two main components. Past users activity, which is recorded in server log files, is processed off-line to form clusters of user sessions. Then the online component builds active user sessions which are then classified into one of the clusters found by the off-line component. The classification allows to identify pages related to the ones in the active session and to return the requested page with a list of suggestions. The geometrical approach used for clustering is affected by several limitations, related to scalability and to the effectiveness of the results found. Nevertheless, the architectural solution introduced was maintained in several other more recent projects.

In [3] B. Mobasher *et al.* present WebPersonalizer a system which provides dynamic recommendations as a list of hypertext links to users. The analysis is based on anonymous usage data combined with the structure formed by the hyperlinks of the site. In order to obtain aggregate usage profiles, DM techniques such as clustering, association rules, and sequential pattern discovery are used in the preprocessing phase. Using such techniques, server logs are converted in *session clusters* (i.e. sequence of visited page) and *pageview clusters* (i.e. set of page with common usage characteristics). The online phase considers the active user session in order to find matches among the users' activities and the discovered usage profiles. The matching entries are then used to compute a set of recommendations which will be inserted into the last requested page as a list of hypertext links. Recently the same author proposed [5] a hybrid personalization model that can dynamically chose between either non-sequential models (like association rules or clustering) or models where the notion of time ordering is maintained (like sequences). The decision of which model to use is based on an analysis of the site connectivity.

In [9] SpeedTracer, a usage mining and analysis tool, is described. Its goal is to understand the surfing behavior of users. Also in this case the analysis is done by exploring the server log entries. The main characteristic of SpeedTracer is that it does not require cookies or user registration for session identification. In fact, it uses five kind of information: IP, Timestamp, URL of the requested page, Referral, and Agent to identify user sessions. The application uses innovative inference algorithms to reconstruct user traversal paths and identify user sessions. Advanced mining algorithms uncover users' movement through a Web site. The final result is a collection of valuable browsing patterns which help webmasters to better understand user behavior. SpeedTracer generates three types of statistics: user-based, path-based and group-based. User-based statistics pinpoint reference counts by user and durations of access. Path-based statistics identify frequent traversal paths in Web presentations. Group-based statistics provide information on groups of Web site pages most frequently visited.

PageGather [7] is a personalization system concerned with the creation of index pages containing topic-focused links. The site organization and presentation is enhanced by using automatically extracted users' access patterns. The system takes as input a Web server log and a conceptual description of each page at a site and it uses clustering to discover pages that are visited together. To perform this task a *co-occurrence* matrix M is built where each element M_{ij} is defined as the conditional probability that page i is visited, given that page j has been visited in the same session. A threshold for M_{ij} allows to prune those entries regarding pages having a low probability of being accessed together. The directed acyclic graph associated with M is then partitioned finding the graph's cliques. Finally, cliques are merged to originate the clusters. This solution introduced the hypotheses that users behave coherently during their navigation, i.e. pages within the same session are in general conceptually related. This assumption is called *visit coherence*. The generated static index pages are kept in a separate "Suggestion Section" of the site.

As we have observed in the introduction, in most of the previous works a two-tiers architecture is generally used. The main limitation of this approach is the loosely coupled integration of the WUM system with the Web server ordinary activity. Indeed, the use of two components leads to the disadvantage to have an "asynchronous cooperation" between the components themselves. The off-line component has to be periodically performed to have up-to-date data patterns, but how frequently it is a problem that has to be solved on a case-specific basis. To overcome this problems, in this work we propose *SUGGEST 2.0*: an incremental personalization procedure, tightly coupled with the Web server ordinary functionalities. Furthermore, we propose a novel quality metric which can be used, in general, to estimate the effectiveness of a recommender system. The metric is focused on measuring the capacity of a recommender system to anticipate users' requests that will be made farther in the future.

In this work we focus our attention on the architecture of WUM systems, and not on the algorithms used for the personalization process. As an extension of a previous work [1] we propose a WUM system whose main contribution with respect to other solutions is to be composed of only one component that is able to update incrementally and automatically the knowledge obtained from historical data and to generate online personalized content. Moreover, as stated above, we used a novel effectiveness measure based on the assumption that a good recommender system should suggest pages to users in order to reduce the average length of a session.

It should be noted that the integration of the off-line and online component functionalities in a single component, poses other problems in terms of overall system performance, and response time. The system, in fact, must be able to generate personalization without too much overhead and, the knowledge mined by the single component has to be comparable, if not better, with that mined by the two separate (online/off-line) components.

2 Our system

The main goal of a recommender system is to provide users with useful suggestions about pages they may find of their interest.

The first version of the system, *SUGGEST 1.0*, was organized as a two-tiers system composed by an off-line module which carried out the first stage and an online classification module which carried out the second stage.

In the new version, hereinafter referred to as *SUGGEST 2.0*, we merged the two stages into a single module performing exactly the same operations but in a complete online fashion. *SUGGEST 2.0* is implemented as a module of the Apache [8] Web server. This to allow an easy deployment on potentially any kind of Web site currently up and running, without any modification of the site itself.

Since *SUGGEST 2.0* is required to work completely online with respect to the stream of requests, we had to completely re-engineer the previous system. In addition we developed a novel online graph partitioning algorithm which we used to incrementally adjust the clustering structure used as the system's knowledge base.

Schematically, *SUGGEST 2.0* works as follows: once a new request arrives at the server, the underlying knowledge base is updated, and a list of suggestions is appended to the requested page. Collapsing the two tiers into a single module pushed aside the asynchronous cooperation problem, i.e. the need to estimate the update frequency of the knowledge base.

In Algorithm 1 the steps carried out by *SUGGEST 2.0* are presented. At each step *SUGGEST 2.0* identifies the URL u requested and the session to which the user belongs. By using the session id it retrieves the identifier of the URL v from which the user is coming. According to the current session characteristics it updates the knowledge base and generates the suggestions to be presented to the user. All this steps are based on a graph-theoretic model which represents the aggregate information about navigational sessions.

To extract information about navigational patterns, our algorithm models the usage information as a complete graph $G = (V, E)$. The set V of vertices contains the identifiers of the different pages hosted on the Web server. The set of edges E is weighted using the following relation:

$$W_{ij} = N_{ij}/max\{N_i, N_j\} \tag{1}$$

where N_{ij} is the number of sessions containing both pages i and j, N_i and N_j are the number of sessions containing only page i or page j, respectively. The rationale of Equation 1 is to try to discriminate internal pages from the so called *index pages*. Index pages are those which, generally, do not contain useful contents and are used only as a starting point for a browsing session. For this reason dividing N_{ij} by the maximum occurrence of the two pages reduces the relative importance of links involving index pages. Index pages are very likely to be visited with any other page and nevertheless are of little interest as potential suggestions. The data structure we used to store the weights is an adjacency matrix M where each entry M_{ij} contains the value W_{ij} computed according to formula 1.

```
Internal state:

  – The matrix $M$ representing the current adjacency matrix for the site;
  – The list $L$ of clusters;
  – The list $A$ of active sessions;
  – The list $PageWindows$ indexed by session identifiers.

Input: The URL $u$ of the requested page.
Output: A list $S$ of suggestions containing URLs considered important with respect
to the detected user session.

$page\_id_u$ = Identify_Page($u$); // Retrieves the id of the URL $u$ by accessing a trie built on top of all of the existing URLs.
$session\_id$ = Identify_Session(); // Using cookies.
$page\_id_v$ = Last_Page($session\_id$); // Returns the last page visited during the current session.
$PW$ = Page_Windows[$session\_id$];
if (!Exists($page\_id_u$, $page\_id_v$, $PW$)) then
  // Exists returns true if and only if the couple ($u$,$v$) is already present in $PageWindows[session\_id]$.
  $M[page\_id_u, page\_id_v]$++;
  if (($W_{uv} > minfreq$ ) $\wedge$ ($L[u] \neq L[v]$)) then
    MergeCluster(L[$u$], L[$v$]); // Merges the two clusters containing $u$ and $v$.
  end if
end if
if (!Exists($page\_id_u$, $PW$)) then
  $M[page\_id_u, page\_id_u]$++;
  $New\_L$ = Cluster($M$, $L$, $page\_id_u$);
  $L = New\_L$;
end if
push($u$, $PW$);
$S$ =Create_Suggestions($PW$, $L$, $u$);
return($S$);
```

Algorithm 1: The operations performed by *SUGGEST 2.0*.

Before computing the weights W, it is necessary to identify user sessions. In *SUGGEST 1.0* we were only able to guess the actual user's session by applying an heuristic based on the IP address and time-stamp. The main drawbacks of this approach are due to the presence of users behind proxies of NATs[2]. In this case, in fact, those users appear as a single one coming from the NAT (or gateway) machine. In *SUGGEST 2.0* users' sessions are identified by means of cookies stored on the client side. Cookies contain the *keys* to identify the clients' sessions. Once a key has been retrieved by our module, it is used to access a table which contains the corresponding session id. The computational cost of such operation is thus relatively small. In fact, a hash table can be used to store and retrieve the keys allowing this phase to be carried out in time $O(1)$.

As in the original formulation, *SUGGEST 2.0* finds groups of strongly correlated pages by partitioning the graph according to its connected component. The algorithm performed by this phase of *SUGGEST 2.0* is shown in Algorithm 2. *SUGGEST 2.0* actually uses a modified version of the well known incremental connected components algorithm. We start from u a Depth First Search (DFS) on the graph induced by M and we search for the connected component reachable from u. Once the component is found, we check if there are any nodes not considered in the visit. If so, a previously connected component has been split and it needs to be identified. We simply apply again the DFS, starting from one of the nodes not visited. In the worst case (when all the URLs are in the same cluster) the cost of this algorithm will be quadratic in the number of pages of the site (to be more precise it will be linear in the number of edges of the complete graph G). Actually, to reduce the contributions of poorly represented link, the incremental computation of the connected components is driven by two threshold parameters. Aim of these thresholds is to limit the number of edges to visit by:

1. filtering those W_{ij} below a constant value. We called this value $minfreq$. Links (i.e. elements M_{ij} of M) whose values are less than $minfreq$ are poorly correlated and thus not considered by the connected components algorithm;
2. considering only components of size greater than a fixed number of nodes, namely $minclustersize$. All the components having less than $minclustersize$ nodes are discarded because considered not sufficiently significant.

In general, the incremental connected component problem can be solved using an algorithm working in $O(|V| + |E|A)$ time, where $A = \alpha(|E|, |V|)$ is the inverse of the Ackermann's function[3]. This is the case in which we have the entire graph and we would incrementally compute the connected component by adding one edge at a time. Our case is slightly different. In fact, we do not deal only with edge addition but also with edge deletion operations. Moreover, depending on the value chosen for *minfreq*, the number of clusters and their size will vary, inducing a variation in the number of edges considered in the clusters restructuring phase.

[2] Network Address Translators.

[3] Since Ackermann's function grows extremely fast, its inverse is a very slowly growing function.

```
Input:

  – The matrix $M$.
  – The clustering structure $L$: $L[i] = c \Leftrightarrow$ the page identifier $i$ is assigned to the
    cluster $c$.
  – The page identifier $u$.

Output: An updated clustering structure.

ret_val = $L$; clust=$L[u]$;
$C = \{n \in [1..|L|] \mid L[n] = \text{clust}\}$;
$h$ = pop($C$);
ret_val[$h$] = $h$;
clust = $h$;
$F \neq \emptyset$;
while $h \neq NULL$ do
  for all ($i \in C$ s.t. $W_{hi} > minfreq$) do
    remove($C$,$i$); // Remove the node $i$ from the set $C$.
    push($F$,$i$);
    retval[$i$]=clust;
  end for
  if $F \neq \emptyset$ then
    pop($F$,$h$);
  else
    if ($C \neq \emptyset$) then
      pop($C$, $h$);
      clust = $h$;
    else
      $h = NULL$;
    end if
  end if
end while
return(ret_val);
```

Algorithm 2: The clustering phase: Cluster(M, L, $page\_id_u$).

After the clustering step, we have to construct the actual suggestions list. This is built in a straightforward manner by finding the cluster which have the largest intersection with the *Page Window* correspondent to the current session (see Algorithm 3). The final suggestions are composed by the most relevant pages in the cluster, according to the order determined by the clustering phase. The cost of this algorithm is proportional to the *Page Window* size and thus is constant ($O(1)$).

3 Suggest Evaluation

To measure the effectiveness and the efficiency of our system we performed several tests. The first measure is the most difficult to define because it implies to

```
Input:
  – The $PageWindow$ related to the current session.
  – The clustering structure $L$: $L[i] = c \Leftrightarrow$ the page identifier $i$ is assigned to the
    cluster $c$.
  – The page identifier $u$.

Output: A list of URL identifiers.

clust= 0; max_rank= 0; ret_val= $\emptyset$;
for ($i = 0; i < PageWindow; i{+}{+}$) do
  rank[$i$] = $|\{n \in PageWindow \mid L[n] = L[PageWindow[i]]\}| + 1$;
  // If the number of nodes shared by the PageWindow and the cluster is maximum and if the size of
  the cluster containing $PageWindow[i]$ is larger than $minclustersize$.
  if ( (rank[$i$] > max_rank)
  $\wedge(|\{n \in L \mid L[n] = L[PageWindow[i]]\}| > minclustersize)$ ) then
    max_rank = rank[$i$];
    clust=L[$PageWindow[i]$];
  end if
end for
$C = \{n \in L \mid L[n] = \text{clust}\}$;
// Return only the pages which have not been visited before.
for all ($c \in C$) do
  for ($i = 0; i < NUMSUGGESTIONS; i{+}{+}$) do
    if (($c \in PageWindow$) $\vee$ ($W_{cu} < W_{u,\text{ret\_val}[i]}$)) then
      break;
    else
      shift(ret_val, $i$);
      ret_val[i] = c;
    end if
  end for
end for
```

Algorithm 3: The suggestions building phase: Create_Suggestions($PageWindow$, L, $page\_id_u$).

be able to measure how much a suggestion has been useful for the user. In [1] we introduced a formula that allows to quantify the effectiveness - i.e. the quality - of the suggestions. Such measure was based on the intersection of real sessions with the corresponding set of suggestions. For every session S_i composed by n_i pages there is a set of suggestions R_i, generated by the module in response to the requests in S_i. The intersection between S_i and R_i is:

$$\omega_i^{old} = \frac{|\ \{p \in S_i |\ p \in R_i\}\ |}{n_i} \tag{2}$$

The main drawback with this measure was that we were not able to capture the potential impact of the suggestions on the user navigational session. For example, if a page that the user would have visited at the end of the session is instead suggested at the beginning of the session, the suggestion in this case could help the user finding a shorter way to what s/he is looking for. Therefore we extend expression 2 taking into account the distance of the suggestions generated with the actual pages visited during the session.

For every user session S_i, we split the session into two halves. The first half S_i^1 is used to generate a set of suggestions R_i^1, the second half is used to measure the intersection with the suggestions. For every page p_k that belongs to the intersection $S_i^2 \cap R_i^1$ and appears in position k within S_i^2, we add a weight $f(k)$. We choose f so that more importance is given to pages actually visited at the end of the session.

A different form for f could have been chosen. For example to have the save coverage measure as used in [5] it is sufficient to take $f(k) = 1$, or any constant value. It is also possible to increase the importance of the pages non linearly by taking $f(k) = k^2$.

In conclusion, for the whole session log, the measure of the quality of the suggestions is given by

$$\Omega = \sum_{i=1}^{N_S} \frac{\sum_{k=1}^{n_i/2} [\![p_k \in \{S_i^2 \cap R_i^1\}]\!] \frac{f(k)}{F}}{N_S} \tag{3}$$

where N_S is the number of sessions and $[\![expr]\!]$ is the truth function equal to 1 if *expr* evaluates to true, 0 otherwise. F is simply a normalization factor on the weights, i.e. $F = \sum_{j=1}^{n_i/2} f(j)$.

We choose to take $f(k) = k$ assuming, in this way, that the weights assigned to pages into the session increase linearly when the position occupied by a page into the session increases.

Another important feature of this measure is that it can also evaluate the ability of *SUGGEST 2.0* in finding a good way to reduce the computational load on the Web server side. In fact, $1 - \omega$ is a measure of the number of additional accesses to pages (maybe unimportant) that a user would have done without the support given by *SUGGEST 2.0*.

Starting from real life access log files, we generated requests to an Apache server running *SUGGEST 2.0* and recorded the suggestions generated for every navigation session contained in the log files. We considered three real life access log files, publicly available: NASA, USASK and BERKLEY containing the requests made to three different Web servers. We report in Table 1 the main features of such datasets.

Dataset	**Time window**	N_s
NASA	27 days	19K
USASK	180 days	10K
BERK	22 days	22K

Table1. Access log files used to measure the quality of suggestions.

For each dataset we measured Ω. The results obtained are reported in Fig. 1. In all curves, varying the *minfreq* parameter, we measure Ω for the suggestions generated by *SUGGEST 2.0*. We also plotted the curve relative to the suggestions generated by a random suggestion generator. As it was expected, the random generator performs poorly and the intersection between a random suggestion and a real session is almost null. On the other hand, suggestions generated by *SUGGEST 2.0* show a higher quality Ω, which, in all dataset, reaches a maximum for *minfreq*=0.2.

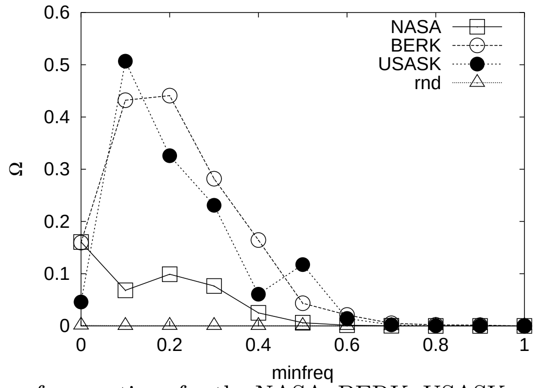

Figure1. Coverage of suggestions for the NASA, BERK, USASK access log files, varying *minfreq*. We also plotted the result of a random suggestions generator.

For low values of the *minfreq* parameter, good values are obtained for the quality of the suggestions, yet a wide margin is left for further improvements.

In order to measure the impact of the *SUGGEST 2.0* module on the overall performance of the HTTP server, we plotted the throughput (see Fig. 2), i.e. number of HTPP requests served per time unit, and the total time needed to serve one single request (see Fig. 3).

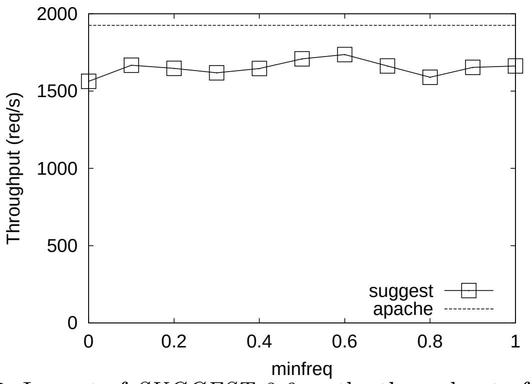

Figure2. Impact of *SUGGEST 2.0* on the throughput of Apache

As we can see from the Figures 2 and 3, the impact of *SUGGEST 2.0* on the Apache Web server is relatively limited, both in terms of throughout and in terms of the time need to serve a single requests. This limited impact on

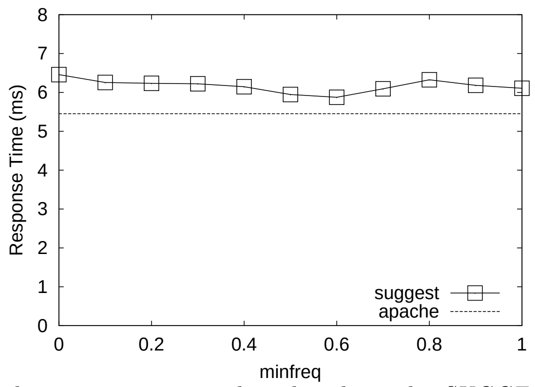

Figure3. Apache response time with and without the *SUGGEST 2.0* module

performance makes *SUGGEST 2.0* suitable to be adopted in real life production servers. It must be noticed, however, that in its current version *SUGGEST 2.0* allocates a buffer of memory whose dimension is quadratic in the number of pages in the site. This might be a severe limitation in sites whose number of pages can be of many thousands.

4 Conclusions

Web personalization can effectively be achieved with data mining techniques. Traditional WUM systems are composed by an off-line extraction of knowledge from historical data, and an online personalization engine. The online component first understands users behavior as HTTP requests arrive at the Web server, and then personalizes the content of the page requested accordingly. Such asynchronous architecture presents the problem of how to maintain and update the knowledge extracted in the off-line phase.

We propose an architecture for a WUM system which is composed by a single online component, tightly integrated in the functionalities of the Apache Web server. Our system, called *SUGGEST 2.0*, processes the requests arriving at a Web server and generates a set of suggestions to Web pages correlated to the one requested.

As the requests arrive at the *SUGGEST 2.0* module it incrementally updates a graph representation of the Web site based on the active user sessions and classifies the active session using a graph partitioning algorithm. At the end of these operations it finally generates the suggestions, by adding at the bottom of the original page requested, a set of links to potentially interesting pages.

We experimentally evaluated *SUGGEST 2.0* performance by measuring the quality of its suggestions and the impact on Apache throughput and response time. To this purpose we introduced a new quality measure aimed at measuring the ability of the recommender system to anticipate the requests eventually made by the users.

References

1. R. Baraglia and P. Palmerini. Suggest: A web usage mining system. In *Proc. of IEEE Int'l Conf. on Information Technology: Coding and Computing*, April 2002.
2. E. Frias-Martinez and V. Karamcheti. Reduction of user perceived latency for a dynamic and personalized site using web-mining techniques. In *Proc. of WebKDD*, pages 47–57, 2003.
3. B. Mobasher, R. Cooley, and J. Srivastava. Automatic personalization based on web usage mining. *Communications of the ACM*, 43(8):142–151, august 2000.
4. Bamshad Mobasher, Namit Jain, Eui-Hong (Sam) Han, and Jaideep Srivastava. Web mining: Pattern discovery from world wide web transactions. TR 96-050, University of Minnesota, 1996.
5. Miki Nakagawa and Bamshad Mobasher. A hybrid web personalization model based on site connectivity. In *Proc. of WebKDD*, pages 59–70, 2003.
6. Olfa Nasraoui and Christopher Petenes. Combining web usage mining and fuzzy inference for website personalization. In *Proc. of WebKDD*, pages 37–46, 2003.
7. Mike Perkowitz and Oren Etzioni. Adaptive web sites: Conceptual cluster mining. In *Int'l Joint Conf. on AI*, pages 264–269, 1999.
8. Robert Thau. Design considerations for the Apache Server API. *Computer Networks and ISDN Systems*, 28(7–11):1113–1122, 1996.
9. Kun-lung Wu, Philip S. Yu, and Allen Ballman. Speedtracer: A web usage mining and analysis tool. *IBM Systems Journal*, 37(1), 1998.
10. Tak Woon Yan, Matthew Jacobsen, Hector Garcia-Molina, and Dayal Umeshwar. From user access patterns to dynamic hypertext linking. *Fifth International World Wide Web Conference*, May 1996.