An Online Recommender System for Large Web Sites

Ranieri Baraglia and Fabrizio Silvestri
Information Science and Technologies Institute (ISTI)
National Research Council, Pisa, ITALY
{ranieri.baraglia, fabrizio.silvestri}@isti.cnr.it

Abstract

In this paper we propose a WUM recommender system, called SUGGEST 3.0*, that dynamically generates links to pages that have not yet been visited by a user and might be of his potential interest. Differently from the recommender systems proposed so far,* SUGGEST 3.0 *does not make use of any off-line component, and is able to manage Web sites made up of pages dynamically generated. To this purpose* SUGGEST 3.0 *incrementally builds and maintains historical information by means of an incremental graph partitioning algorithm, requiring no off-line component. The main innovation proposed here is a novel strategy that can be used to manage large Web sites. Experiments, conducted in order to evaluate* SUGGEST 3.0 *performance, demonstrated that our system is able to anticipate users' requests that will be made farther in the future, introducing a limited overhead on the Web server activity*[1].

1 Introduction

The continuous and rapid growth of the Web has led to the development of new methods and tools in the Web recommender or personalization domain [4], [2]. In [7] the goal of the Web personalization is defined as "*provide users with the information they want or need, without expecting from them to ask for it explicitly*".

Web Mining has shown to be a viable technique to discover information "hidden" into Web-related data [3]. In particular, Web Usage Mining (WUM) is the process of extracting knowledge from Web users access data (or clikstream) by exploiting Data Mining (DM) technologies [5]. It can be used for different purposes such as *personalization*, *system improvement* and *site modification*.

Typically, the WUM personalization process is structured according to two components performed online and off-line with respect to the Web server activity [5], [16], [6], [1]. The off-line component is aimed at building the knowledge base by analyzing historical data, such as server access log files, that is then used in the online component. The main functions carried out by this component are *Preprocessing*, i.e. data cleaning and session identification, and *Pattern Discovery*, i.e. the application of DM techniques, like association rules, sequential patterns, clustering or classification. The online component is devoted to the generation of personalized content. On the basis of the knowledge extracted in the off-line component, it processes a request to the Web server by adding personalized content which can be expressed in several forms, such as links to pages, advertisements. The main limitation of this approach is the loosely coupled integration of the WUM system with the Web server activity. In fact, the use of two components has the drawback of having "*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, is a problem that has to be solved on a case-specific basis. On the other hand the integration of the off-line and online component functionalities into a single one, poses other problems in terms of the overall system performance, which should have a very low impact on user response times.

In this work, we present a WUM system, called *SUGGEST 3.0*, which is designed to dynamically generated personalized content of potential interest for users of a Web Site. It is based on an incremental personalization procedure, tightly coupled with the Web server. It is able to update incrementally and automatically the knowledge base obtained from historical usage data and to generate a list of page links (*suggestions*). The suggestions are used to personalize *on-the-fly* the HTML page requested. Moreover, the adoption of a LRU-based algorithm to manage the knowledge base permits us to use *SUGGEST 3.0* also on large Web sites made up of pages dynamically generated. Thus, removing a limitation of previous system versions

[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)

that were able to manage Web sites with a fixed number of pages. The system proposed was evaluated by adopting the new quality metric we introduced in [13]. The metric tries to estimate the effectiveness of a recommendation system as the capacity of anticipating users requests that will be made farther in the future.

2 Related Work

In the past, several WUM projects have been proposed to foresee users preference and their navigation behavior. In the following we review some of the most significant WUM projects that can be compared with our system.

Analog [16] is one of the first WUM systems. It is structured according to an off-line and an online component. The off-line component builds session clusters by analyzing past users activity recorded in server log files. Then the online component builds active user sessions which are then classified according to the generated model. 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 [11] Perkowitz *et al.* propose a WUM system, called *PageGather*, that builds index pages containing links to pages similar among themselves. PageGather finds clusters of pages instead of clusters of sessions. Starting from the user activity sessions, it builds the *co-occurrence* matrix M. Each element M_{ij} of M is defined as the conditional probability that page i is visited during a session given that page j has been visited in the same session. A threshold minimum value for M_{ij} allows to prune some uninteresting entries. The directed acyclic graph G associated to M is then partitioned finding the graph's cliques. Finally, cliques are merged to originate the clusters. Page Gather main concern is on the index pages creation. There is not an online component of the WUM system, and the static index pages are kept in a separate "Suggestion Section" of the site. One important concept introduced in [11] is 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*. *SUGGEST 3.0* exploits this concept to obtain a measure of quality for a WUM system.

In [5] and [8] 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. DM techniques (i.e. clustering, association rules, and sequential pattern discovery) are used in the preprocessing phase in order to obtain aggregate usage profiles. In this phase Web server logs are converted in clusters made up of sequences of visited pages, and cluster made up of set of pages 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. 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. WebPersonalizer is a good example of a two-tier architecture for personalization systems.

In [15] *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. Advanced mining algorithms uncover users movement through a Web site. The final result is a collection of valuable browsing patterns which help webmasters better understand users behavior. SpeedTracer generates three types of statistics: user-based, path-based and group-based. User-based statistics pinpoint reference counts and durations of accesses. Path-based statistics identify frequent traversal paths in Web presentations. Group-based statistics provide information on groups of Web site pages most frequently visited.

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 [9].

3 The *SUGGEST 3.0* system

SUGGEST 3.0 is a recommender system aimed to provide the users with useful information about pages they may find of their interest. The personalization is achieved by means of a set of dynamically generated page links.

SUGGEST 1.0 was born as a two-tier system composed by an off-line module which carried out the first stage by analyzing the Web server's access log file, and an online classification module which carried out the second stage. Its main drawback was the asynchronous cooperation between the two modules. In the next version, *SUGGEST 2.0*, the two modules were merged into a single one that performs the same operations but in a complete online fashion. Putting together the two tiers into a single module pushed aside the problem to estimate the update frequency of the knowledge base. *SUGGEST 2.0* was designed to be usable on Web sites made up of pages statically generated, i.e. Web sites with a fixed number of pages. A list containing all the information describing a Web site pages was required as input by *SUGGEST 2.0* at its start-up time. Potential limitation of *SUGGEST 2.0* might be: a) the memory required to store Web server pages is quadratic in the number of pages. This might be a severe limitation in large sites made up of millions of pages; b) it does not permit us to manage Web sites made up of pages dynamically generated.

In order to remove the above limitations in *SUGGEST 3.0* a solution that indexes a page when it is required is applied. This solution can leads to a large increasing of the knowledge base. Therefore, to avoid that, a LRU-based algorithm is adopted. According to this algorithm information related to a page less recently accessed is replaced with the information related to the one currently accessed. The size of the used data structures is specified as a function of some parameters such as the available resources and the performance requirements.

As the previous version, *SUGGEST 3.0* is implemented as a module of the Apache Web server [14], allowing an easy deployment on potentially any kind of Web site currently up and running, without any modification of the site itself.

Schematically, *SUGGEST 3.0* works as follows: once a new request arrives at the server, the URL requested and the session to which the user belongs are identified, the underlying knowledge base is updated, and a list of suggestions is appended to the requested page.

In Algorithm 1 the steps carried out by *SUGGEST 3.0* are presented. At each step *SUGGEST 3.0* identifies the URL u requested and the session to which the user belongs. Successively, by using the session identifier id, it retrieves the identifier of the URL v from which the user is coming from, and the $PageWindow$ containing the list of pages accessed in the current session. According to the current session characteristics, it updates the knowledge base and generates the suggestions to be presented to the user. According to the sequence page accesses, suggestions for users belonging to the same class may be different. All this steps are based on a graph-theoretic model which represents the aggregate information about the navigational sessions.

The session model. In *SUGGEST 3.0* user sessions are identified by means of cookies stored on the client side. The use of the cookie mechanism remove a drawback present in *SUGGEST 2.0* that identified user sessions by applying a heuristic based on the IP address and time-stamp. This solution does not permit us to identify 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. On the other hand a user can disable cookies nullifying our mechanism.

The first time a user issues a Web page request, *SUGGEST 3.0* catches it, and sends back to the user, together with the requested page, a cookie containing a counter of the requested page. This mechanism initiate a protocol that permits us to have, after a prefixed number of requests a valid user session used to make the list of suggestions. After a valid session has been identified a cookie containing a *key* to identify the session is sent back to the user together with the request page. Once a key has been generated by our module, it is used on successive requests to identify the corresponding user session.

[2]Network Address Translators.

The keys are used to access a hash table which contains the corresponding session identifiers ($session\_id$). Therefore the computational cost of such operation is constant ($O(1)$).

To catch information about navigational patterns, *SUGGEST 3.0* models the page accesses information as a undirected graph $G = (V, E)$. The set V of vertices contains the identifiers of the different pages hosted on the Web server. Based on the fact that the interest in a page depends on its content and not on the order a page is visited during a session [1], we assign to each edge E a weight computed as:

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

where N_{ij} is the number of sessions containing both pages i and j, N_i and N_j are respectively the number of sessions containing only page i or page j. Dividing by the maximum between single occurrences of the two pages has the effect of reducing the relative importance of links involving index pages. Such pages are those that, generally, do not contain useful content and are used only as a starting point for a browsing session. 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.

In order to manage large Web Sites that may require an adjacency matrix that exceeds the maximum available memory, we adopted a LRU-based strategy to store in M only those pages that have been recently accessed by some users. Obviously at some point, a requested page might not be present in M (nor in L). Therefore, it must be inserted into M (and L) by replacing the entry related to the least recently accessed page. Therefore, a data structure, called LRU_map that maps the univocal page identifiers to entries of both the matrix M and the clustering structure L is adopted.

Clustering Algorithm. As in the previous version, *SUGGEST 3.0* finds groups of strongly correlated pages by partitioning the graph according to its connected components. In Algorithm 2 the steps performed in this phase are presented. *SUGGEST 3.0* actually uses a modified version of the well known incremental connected components algorithm [12]. Starting from u a Depth First Search (DFS) on the graph induced by M is applied to search for the con-

```
Internal state:
  • The matrix $M$ representing the current adjacency matrix for the site;
  • The list $L$ of clusters;
  • The list $PageWindows$ indexed by session identifiers.
  • The mapping function $LRU\_map$ used map page identifiers to entries of $M$ and $L$.
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 identifier of the URL $u$ by accessing a trie built on top of all of the URLs considered.
update_LRU($LRU\_map$, $page\_id_u$);
// Performs the update of the LRU-based mapping structure.
$session\_id$ = Identify_Session();
// Retrieves the session identifier by using cookies.
$page\_id_v$ = Last_Page($session\_id$);
// Returns the page the user is coming from in the current session.
update_LRU($LRU\_map$, $page\_id_v$);
$PW$ = Page_Windows[$session\_id$];
// Retrieves the $PageWindow$ identifier by using the current session identifier.
if (!Exists($page\_id_u$, $page\_id_v$, $PW$) then
    // Exists returns true iff the pair ($u$,$v$) is already present in $PageWindows[session\_id]$.
    $M[LRU\_map[page\_id_u], LRU\_map[page\_id_v]]$++;
    if (($W_{uv}$ > minfreq) & ($L[LRU\_map[page\_id_u]] \neq L[LRU\_map[page\_id_v]]$)) then
        MergeCluster(L[$LRU\_map[page\_id_u]$], L[$LRU\_map[page\_id_v]$]);
        // Merges the two clusters containing $u$ and $v$.
    end if
    $M[LRU\_map[page\_id_u], LRU\_map[page\_id_u]]$++;
    $New\_L$ = Cluster($M$, $L$, $LRU\_map[page\_id_u]$); // Updates the knowledge base.
    $L = New\_L$;
end if
Push($u$, $PW$);
$S$ =Create_Suggestions($PW$, $L$, $LRU\_map[page\_id_u]$);// Generates the list of suggestions.
return($S$);
```

Algorithm 1: The operations performed by *SUGGEST 3.0*.

nected component reachable from u. Once the component has been found, *SUGGEST 3.0* checks if there are any nodes not considered in the visit. If so, it means that a previously connected component has been split, and therefore, it needs to be identified. To do this the DFS is again applied by 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 linear in the number of edges of the complete graph G. To reduce the contribution 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, called $minfreq$. Elements M_{ij} of M (i.e. links between pair of pages) whose values are less than $minfreq$ are poorly correlated and thus not considered by DFS 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 significant enough.

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 sizes will vary, inducing a variation in the number of edges considered in the clusters restructuring phase.

Suggestions Building. After the clustering step, *SUGGEST 3.0* has to construct the suggestions list for the current user request. This is done in a straightforward manner by finding the cluster that has the largest intersection with the *PageWindow* related to the current session. In Algorithm 3 the steps performed in this phase are presented. 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 *PageWindow* size and thus is constant ($O(1)$).

4 Suggest Evaluation

Measuring the performances of recommendation systems poses more than one problem. It is difficult to characterize the quality of the suggestions obtained and to quantify how useful the system is. In order to evaluate both the effectiveness (i.e. the quality of the suggestions) and efficiency (i.e. overhead introduced on the overall performance of the Web server) of *SUGGEST 3.0* several tests were conducted.

All tests were run on a processor Intel Celeron 2,4 GHz with 256 MBytes of RAM, an ATA 100 disk with 30 GBytes, and operating system Linux Suse 8.2 (kernel 2.4.20).

The *SUGGEST 3.0* effectiveness was evaluated by using a performance parameter we introduced in [13]. Such parameter 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:

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

```
Internal state:
  • The mapping function $LRU\_map$ used map page identifiers to entries of $M$ and $L$.
Input:
  • The matrix $M$.
  • The clustering structure $L$. $L[i] = c$ iff the page identifier $i$ is assigned to the cluster $c$.
  • The page identifier $u$.
Output: An updated clustering structure.

ret_val = $L$; clust=$L[LRU\_map[page\_id_u]]$;
$C = \{n \in [1..|L|] \mid L[n] = \text{clust}\}$;
// $C$ is the set containing all the nodes of the cluster identified by $clust$.
$h$ = pop($C$); // Extracts the first element from $C$.
ret_val[$h$] = $h$; // Sets $h$ equal to the cluster identifier for $h$.
$clust = h$;
$F = \emptyset$;
while $h \neq NULL$ do
  for all ($i \in C$ s.t. $W_{hi} >$ minfreq) do
    remove($C$,$i$); // Removes the node $i$ from the set $C$.
    push($F$,$i$); // Inserts the node $i$ into $F$.
    retval[$i$] = $clust$; // Assigns the node $i$ to cluster $clust$.
  end for
  if $F \neq \emptyset$ then
    $h$ = pop($F$);
  else
    if ($C \neq \emptyset$) then
      $h$ = pop($C$);
      $clust = h$;
    else
      $h = NULL$;
    end if
  end if
end while
return(ret_val);
```

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

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

With this measure we are not able to capture the potential impact of the suggestions on the user navigational session. For example, if a page that the user would visit 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 she/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

```
Internal state:
  • The mapping function $LRU\_map$ used map page identifiers to entries of $M$ and $L$.
Input:
  • The $PageWindow$ related to the current session.
  • The clustering structure $L$. $L[i] = c$ iff the page identifier $i$ is assigned to the cluster $c$.
  • The page identifier $page\_id_u$.
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)
  &$(|\{n \in L \mid L[n] = L[PageWindow[i]]\}| > minclustersize)$ )
  then
    max_rank = rank[$i$];
    clust=L[PageWindow[$i$]];
  end if
end for
$C = \{n \in L \mid L[LRU\_map[page\_id_u]] = \text{clust}\}$;
// Returns the page which have not been visited before.
for all ($c \in C$) do
  for ($i$ = 0; $i <$ NUMSUGGESTIONS; $i$++) do
    if (($c \notin PageWindow$) $\vee (W_{cu} < W_{u,\text{ret\_val}[i]})$) then
      push($ret\_val$,$i$); ret_val[$i$] = $c$;
    end if
  end for
end for
return($ret\_val$);
```

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

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 [8] it is sufficient to take $f(k) = 1$, or any constant value. For instance, 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 lin-

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

Table 1. Access log files used to measure the suggestions quality.

early when the position occupied into the session increase.

To evaluate the *SUGGEST 3.0* effectiveness experimental evaluation was conducted by using three real life access log files of public domain[4]: Berkeley, NASA, USASK, produced by the Web servers of the Computer Science Department of Berkeley University, Saskatchewan University and Kennedy Space Center, respectively. The characteristics of the datasets we used are given in Table 1. For each test we generated requests to an Apache server running *SUGGEST 3.0* and recorded the suggestions generated for every navigation session contained within the access log file considered.

For each dataset we measured Ω varying the *minfreq* parameter. Figure 1 shows the results obtained. Moreover, we also plotted the curve relative to the suggestions generated by a random suggestion generator (labelled rnd in Fig. 1). 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 3.0* show a higher quality, that, in all the datasets, reaches a maximum for *minfreq*=0.2.

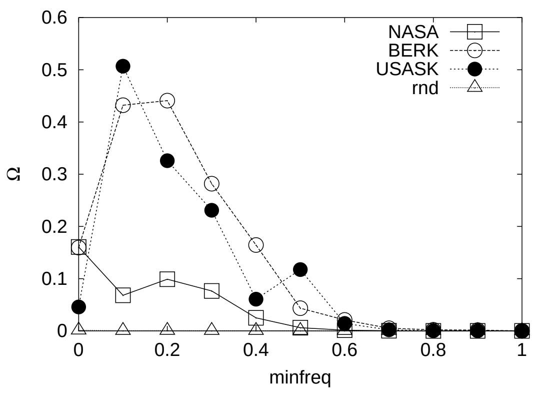

Figure 1. Coverage of suggestions for the NASA, BERK, USASK access log files, varying *minfreq*.

For low values of the *minfreq* parameter, good values are obtained for the quality of the suggestions.

In order to evaluate the overhead introduced by *SUGGEST 3.0* on the overall performance of the Apache Web server we conduced the experimental evaluation using a "clone" (i.e. the same structure and pages) of the Web site of the Computer Science Department of Berkeley University. To this end the Apache's `ab`[5] benchmarking tool was used. `ab` is able to carry out how many user requests the server performs per second. In all the tests $100,000$ user requests were performed, and the number of requests performed per second was changed by varying the number of requests simultaneously issued. The three lines in the following graphs refer to standard Apache, Apache using the *SUGGEST 3.0*, and to Apache using the *SUGGEST 2.0*.

In Figure 2, we plotted the execution time (expressed in milliseconds) that a Apache process spends to satisfy an HTTP request. We vary the degree of concurrency by submitting an increasing number of requests to the server.

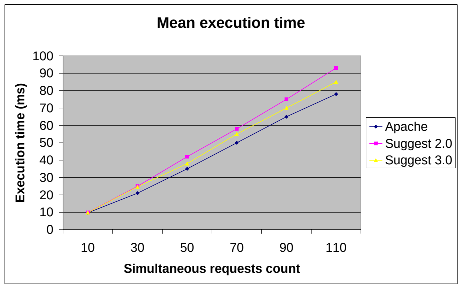

Figure 2. Apache response time with and without the SUGGEST module.

In all the three cases, as the number of concurrent requests increases the request response time increases proportionally. It is due to the mutual exclusive accesses to shared memory areas by the Apache processes. However, the overhead introduced by *SUGGEST 3.0* is relatively limited, and it obtains good performance also when a high number of requests is performed.

Figure 3 shows the number of HTTP requests served per second. In all the tests, *SUGGEST 3.0* obtained performance close to those obtained by the Apache server, and better than those obtained by *SUGGEST 2.0*.

The performance improvement obtained by *SUGGEST 3.0* is due to a better usage of the data structures. Both the versions use a *trie* (a data structure with logarithmic access time in the number of its elements) to index the Web pages. *SUGGEST 2.0* was usable only on Web sites with an *a priori* known number of pages (i.e. elements to index), and, therefore, with a fixed page access time. Instead, since *SUGGEST 3.0* is able to manage Web sites with dynamic pages it will obtain the performance similar to those obtainable by *SUGGEST 2.0* on the same site when the number of accessed pages reaches its maximum value.

These experimental results show that the impact of *SUGGEST 3.0* on the Web server is relatively limited, both in

[4]`www.web-caching.com`

[5]http://httpd.apache.org/docs/programs/ab.html

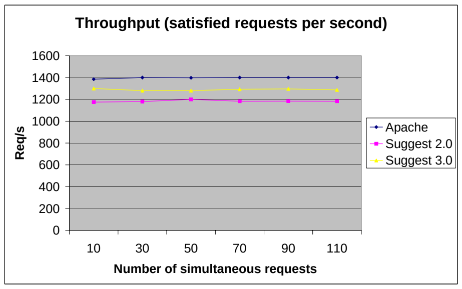

Figure 3. Apache throughput with and without the SUGGEST module.

terms of throughout and in terms of the time need to serve a single request. This limited impact on performance makes *SUGGEST 3.0* suitable to be adopted in real life production servers.

5 Conclusions

In this work we have presented a new WUM recommender system, called *SUGGEST 3.0*, that is able to dynamically generated personalized content in order to make easier the Web user navigation. The proposed system is composed by a single component, tightly integrated with the Apache Web server. It is based on an incremental procedure, that is able to update incrementally and automatically the knowledge base obtained from historical usage data and to generate a list of links to pages (suggestions) of potentially interest for the user. The suggestions are used to personalize *on-the-fly* the HTML page requested. The adoption of a LRU-based algorithm to manage the knowledge base, permits us to use our system also on Web sites that exploit pages dynamically generated (i.e. Web site made up of a not fixed number of pages). Experimental results show that *SUGGEST 3.0* is able to generate valid suggestions with a limited overhead on the Web server. Moreover, the exploitation of the suggestion can lead to reduce the average session length improving the performance of the Web server.

As future work we are planning to evaluate *SUGGEST 3.0* when running on real Web sites. Moreover, we are going to extend the suggestions creation phase by finding the most ”interesting” pages among those present in the selected cluster. This ”interestingness” property should be evaluated by computing a sort of PageRank value of the pages in a Web site. To this end we think to extend the classical PageRank algorithm [10] to evaluate the page relevance using both the information about the site linkage structure, and the information extracted from the historical Web usage data (i.e. in our case stored in the adjacency matrix M).

6 Acknowledge

The authors would like to thank Paolo Palmerini who originally developed the seminal ideas on Suggest. Next, we thank Paolo Nicolai, Iliano Riccardi, and Massimo Serranó for the contribution given to develop this project during their master thesis work.

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] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Trans. on Information Systems*, 22(1):143–177, January 2004.

[3] R. Kosala and H. Blockeel. Web mining research: A survey. *ACM SIGKDD*, 2(1):1–15, July 2000.

[4] E. Magdalini and M. Vazirgiannis. Web mining for web personalization. *ACM Trans. on Internet Technology*, 3(1):1–27, February 2003.

[5] B. Mobasher, R. Cooley, and J. Srivastava. Automatic personalization based on web usage mining. *Communications of the ACM*, 43(8):142–151, august 2000.

[6] B. Mobasher, N. Jain, E.-H. S. Han, and J. Srivastava. Web mining: Pattern discovery from world wide web transactions. TR 96-050, University of Minnesota, 1996.

[7] M. D. Mulvenna, S. S. Anand, and A. G. Buchener. Personalization on the net using web mining. *Communication of ACM*, 43(8), 2000.

[8] M. Nakagawa and B. Mobasher. A hybrid web personalization model based on site connectivity. In *Proc. of WebKDD*, pages 59–70, 2003.

[9] O. Nasraoui and C. Petenes. Combining web usage mining and fuzzy inference for website personalization. In *Proc. of WebKDD*, pages 37–46, 2003.

[10] L. Page, S. Brin, R. Motwani, and T. Winograd. The Pagerank Citation Ranking: Bringing Order to the Web. Technical report, Stanford University, 1998.

[11] M. Perkowitz and O. Etzioni. Adaptive web sites: Conceptual cluster mining. In *Int'l Joint Conf. on AI*, pages 264–269, 1999.

[12] J. G. Siek, L. Lee, and A. Lumsdaine. *Boost Graph Library, The: User Guide and Reference Manual*. Addison Wesley Professional, 2001.

[13] F. Silvestri, R. Baraglia, P. Palmerini, and S. M. On-line generation of suggestions for web users. In *Proc. of IEEE Int'l Conf. on Information Technology: Coding and Computing*, April 2004.

[14] R. Thau. Design considerations for the Apache Server API. *Computer Networks and ISDN Systems*, 28(7–11):1113–1122, 1996.

[15] K.-l. Wu, P. S. Yu, and A. Ballman. Speedtracer: A web usage mining and analysis tool. *IBM Systems Journal*, 37(1), 1998.

[16] T. W. Yan, M. Jacobsen, H. Garcia-Molina, and D. Umeshwar. From user access patterns to dynamic hypertext linking. *Fifth International World Wide Web Conference*, May 1996.