MINING USER ACCESS PATTERNS AND IDENTITY INFORMATION FROM WEB LOGS FOR EFFECTIVE PERSONALIZATION

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Esra Satıroğlu

September, 2001

ABSTRACT

MINING USER ACCESS PATTERNS AND IDENTITY INFORMATION FROM WEB LOGS FOR EFFECTIVE PERSONALIZATION

Esra Satıroğlu
M.S. in Computer Engineering
Supervisor: Prof. Dr. Halil Altay Güvenir
September, 2001

Web is a huge source of data in terms of its usage as a result of being visited by millions of people on each day. Its usage data is stored in web server logs which contain a detailed description of every single hit taken by the corresponding web server. Recently, it has been discovered that analysis of this data for understanding the user behavioral patterns may have critical implications. Understanding the behavioral patterns of visitors is especially important for e-commerce companies which try to gain customers and sell products through the web. Interactive sites that recognize their customer and customize themselves accordingly may save lots of money to the companies. Usage Based Personalization is a study on designing such personalized sites. In this thesis, we present a new usage based personalization system. The system we designed and implemented is capable of guessing the web pages that may be requested by the on-line visitors during the rest of their visits. The system shows the subset of these pages with highest scores as recommendations to the visitors as being attached to the original pages. The system has two major modules. The off-line module mines the log files off-line for determining the behavioral patterns of the previous visitors of the web site considered. The information obtained by the off-line module is utilized by the on-line module of the system for recognizing new visitors and producing on-line recommendations. The first criterion for identifying on-line visitors is the paths followed by them. A path of a particular visitor consists of pages retrieved by him throughout his visit to the web site. Another criterion considered by the system is the identity information (IP address or domain name) of the visitors. By using identity information, it is possible to learn old preferences of the visitor himself or visitors from similar domains. The similarity between domains is determined by the help of the domain name hierarchy which is represented by a hierarchical tree structure. The leaves of the tree representing domain name hierarchy contain the domain names of the machines connecting to the Internet while the inner nodes contain the general domains such as *com*, *edu.tr*, etc. In domain name hierarchy, the similarity between two domains increases as the number of common parents of them increases. In the light of these observations, for guessing the navigational behavior of a particular visitor, our system makes use of the common behavioral trends of the visitors whose machines belong to the parent domains of the domain of that visitor. We have tested the system on the web site of CS department of Bilkent University. The results of the experiments show the efficiency and applicability of the system.

Keywords: Personalization, Web Usage Mining, User Access Patterns.

ÖZET

ETKİLİ KİŞİSELLEŞTİRME İÇİN WEB GÜNLÜKLERİNİN VERİ MADENCİLİĞİ YÖNTEMİ İLE ANALİZİ

Esra Satıroğlu
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Halil Altay Güvenir
Eylül, 2001

Web, her gün milyonlarca kişi tarafından ziyaret edilmesi nedeniyle kullanımı açısından büyük bir veri kaynağıdır. Kullanım verileri, web dağıtıcılarına ulaşan her türlü veri isteğinin kaydedildiği web günlüklerinde saklanmaktadır. Yakın geçmişte, bu veri topluluğunun kullanıcı davranış örüntülerini anlamak amacı ile analiz edilmesinin önemli sonuçlar doğurabileceği anlaşıldı. Kullanıcıların davranışsal örüntülerinin anlaşılması özellikle web aracılığı ile müşteri kazanmaya ve ürünlerini satmaya calışan elektronik ticaret şirketleri için önemlidir. Kullanıcılarını tanıyabilen ve kendisini kullanıcılarına göre kişiselleştirebilen etkileşimli web sitelerine sahip olmak şirketlerin oldukça önemli miktarlarda kar etmesini sağlayabilir. Kullanıma dayanan kişiselleştirme, kişiselleştirilmiş web siteleri üretmeye yönelik araştırma alanının genel adıdır. Bu tezde, kullanıma dayanan yeni bir kişiselliştirme sistemi tanıtılacaktır. Bizim tarafımızdan tasarlanıp gercekleştirilen bu sistem, bir web sitesinin ziyaretçilerinin ziyaretlerinin geri kalan kısmında talep edebilecekleri web sayfalarını tahmin etme yeteneğine sahiptir. Sistem, bu sayfalardan en yüksek skorlu olanları ziyaretçilerin görüntüledikleri sayfalara ekler. Tasarlayıp gerçekleştirdiğimiz bu sistemin iki ana modülü bulunmaktadır. Çevrimdışı modül web günlük dosyalarını eski kullanıcıların davranışsal örüntülerini keşfetmek amacı ile çevrimdışı olarak analiz eder. Çevrimdışı modül tarafından elde edilen bilgiler yeni ziyaretçileri tanımak ve onlar için öneriler üretmek amacı ile çevrimiçi modül tarafından kullanılır. Yeni ziyaretçileri tanımak amacı ile kullanılan ilk kriter bu ziyaretçilerin web sitesinde izledikleri yollardır. Yol, bir ziyaretçinin web sitesine ziyareti süresinde görüntülediği sayfalardan oluşur. Yeni ziyaretçileri tanımak amacı ile kullanılan diğer bir kriterde bu ziyaretçilerin siteye bağlandıkları makinanın kimlik bilgisidir. Kimlik bilgisi, makinanın IP adresi veya etki alanı ismidir. Kimlik bilgilerini kullanarak ziyaretçinin kendisinin veya benzer etki alanlarında bulunan ziyaretçilerin eski tercihlerini öğrenmek mümkün hale gelmektedir. İki etki alanı arasındaki benzerlik etki alanı sıradüzeni yardımı ile belirlenir. Etki alanı sıradüzeni, sıradüzensel ağaç yapısı ile temsil edilebilir. Bu ağacın en alt seviyesindeki düğümler İnternet'e bağlanan makinaların etki alanı isimlerini içerir. Diğer düğümler ise *com* veya *edu.tr* gibi genel etki alanlarını kapsar. Bu sıradüzende, iki etki alanı arasındaki benzerlik bu etki alanlarının ortak ata dügüm sayısı arttığı sürece artar. Bu bilgilerin ışığında, tasarladığımız sistem bir ziyaretçinin hangi sayfaları talep edebileceğini anlamak için bu ziyaretçinin İnternet'e bağlanmak için kullandığı makinanın etki alanının ata etki alanlarına ait olan kullanıcıların ortak davranışsal yönsemelerinden yararlanmaktadır. Bu sistem, Bilkent Üniversitesi BilgisayarMühendisliği bolümünün web sitesi üzerinde test edilmiş ve sonuçlar sistemin verimliliğini ve kullanılabilirliğini kanıtlamıştır.

Anahtar sözcükler: Kişiselleştirme, Web Kullanım Analizi, Kullanıcı Erişim Örüntüleri.

Acknowledgement

I would like to express my thanks to Prof. Dr. H. Altay Güvenir for the suggestions and support he provided during this research.

I would also like to thank to Prof. Dr. Cevdet Aykanat and Assist. Prof. Dr. Attila Gürsoy for accepting to read this thesis.

Finally, I would like to express my special thanks to my father Adnan, my mother Emine, my sister Yasemin and my fiance Özgür for their love and great support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data mining and world wide web (www) are two important research areas whose incorporation is named as Web Mining in the literature. Web is a huge source of data in terms of both its content and usage. Every day millions of people navigate through web for different purposes ranging from shopping to searching for information. As the size of the web grows, management of the data available through it becomes more and more difficult. This increase in the size of the data available through the web have made it necessary to find intelligent ways to reach the data and analyze the user behavior on it.

Meanwhile, data mining provides techniques for discovering meaningful and useful information from large volumes of data. The application of the data mining techniques makes it possible to extract various kinds of patterns and relations hidden in the data which may have critical implications if discovered. Web, because of being one of the largest data sources available, needs such a treatment to be kept under control. This need has caused a lot of research on the application of the data mining methodology to the data obtained through the web. Application of data mining techniques to the data coming from the web is named as Web Mining. Web Mining techniques are classified into two according to the type of data that are processed by them [26].

The first type of data provided by the web is named as the Content Data which is the data presented in the pages. Every single person is capable of contributing to the Content Data on the web with no limitation. So, the web contains incredible amount of information on every topic that can be thought of. This information can become valuable if it is retrievable by the other people. Naturally, each piece of content data is put on a web to be found and used by people. But, reaching to the desired information on the web is not an easy task, since the data on it is disorganized and heterogenous. Search engines which have naive approaches that do not go beyond the keyword search on documents seem to solve this problem partly, but the data is not categorized, filtered and interpreted by them.

Web Content Mining aims to develop more intelligent tools for information retrieval [6]. It uses data mining to develop better techniques for organizing the unstructured data and for creating more intelligent agents to help the visitors in finding, filtering and evaluating the desired information sources. Web Content Mining studies are divided into two categories according to their methodologies, namely *database approach* and *agent based approach* [11].

Database approach to web content mining aims to organize the data available on the web by collecting it in multilevel databases [9, 30]. In multilevel databases, the lowest layer contains the pages themselves. As proceeding through the higher levels, data becomes more generalized. The data in a particular layer is obtained via generalization or transformation of the data in the lower layers [30]. With the available powerful querying systems and web mining techniques, the data organized in that way is accessed and analyzed easily [10, 12]. On the other hand, agent based approach to web content mining aims to develop intelligent agents that can discover and organize the information on the web by using domain characteristics, user profiles and user preferences [2, 22, 14]. For example, Harvest [2] and Parasite [22] use domain specific information about documents for interpreting them. In addition, Syskill & Webert determines the interestingness level of the documents for the visitors by looking at user profiles [14].

Another type of data provided by the web is named as Usage Data which consists of the traversal patterns of visitors of the web. Traversal patterns are stored in a server log which keeps a record of every access made to the pages belonging to that server. Log files of popular sites can quickly reach very huge sizes, because there is no limit on the number of people that can visit the site and number of pages that can be visited by each person. Each user visits the site with potentially different expectations and aims which make him/her to follow a certain path and visit some pages. The analysis of such behavioral patterns gains more importance as the size of the web grows and becomes a primary tool for commerce. Web Usage mining is mainly the analysis of the usage data collected in web server logs by using the data mining techniques.

Understanding the behavioral patterns of visitors is especially important for e-commerce companies which try to gain customers and sell products through the web [13]. Their web sites are the only connection between them and their customers, so the design of the web site is critical. Web Usage Mining tools may help companies in designing more attractive sites for their customers and discovering the problems that visitors encounter on their visit to the site. Interactive sites that recognize their customer and present themselves according to their previous experiences about that kind of customers may save lots of money to the company. For example, a site may contain advertisements that may be interesting to the visitor. Or, the links to product pages that are often visited together may be collected in one page for an easy retrieval by the users. In short, the aim is to understand the visitors and respond accordingly.

There are lots of commercial tools which analyze the data in the logs and extract various kinds of statistical information such as the number of visitors of the site, most popular pages, the characteristics of the users, etc. However, the analysis performed by these tools is not enough, although it is valuable [29]. The major drawbacks of these tools are the lack of in-depth analysis, low performance on huge log files, restrictiveness to the predefined reports and inflexibility. Today, it is known that log files contain more valuable and critical information which can be discovered through the application of new or existing data mining techniques. As a result, Web Usage Mining has became a popular research topic for many researchers in the last years.

In this thesis, we present a new usage mining system, called UsageMiner. The system we designed and implemented is capable of guessing the web pages that may be requested by the on-line visitors during the rest of their visits. Then, it shows the subset of these pages with highest scores as recommendations to the visitor. One of the aims of this study is to provide the user with an easy access to the pages that he/she may be interested in. For example, at a particular page the system can discover that the visitor may be interested in another page which can be reached from the current document after a retrieval of a number of pages. On the other hand, after the system recommends this page, if the visitor is really interested, he/she can reach to it just clicking on a link instead of following a number of links. So, navigation on the site becomes easier for the visitors. In addition to those, the system is also beneficial for the new visitors who are not much familiar with the contents of the site. Web site can automatically discover the interests of these visitors and present itself accordingly. Because, it can discover the pages that may be interesting to them and recommends these pages to the visitor. By this way, it becomes much easier for the new visitors to discover the contents of the site that may be interesting for them.

The system we designed is under the category of personalization studies about which we will talk about in the next chapter. It has mainly two major modules, off-line module and on-line module. Off-line module mines the log files off-line for determining the behavioral patterns of the old visitors. Then, the information obtained by the off-line part of the system is utilized for recognizing new visitors and producing on-line recommendations for them. The system tries to recognize the visitor by looking at the identity information and path followed by them. On a given page, it combines the pages that are retrieved in the followup by the old visitors following the same path or coming from similar domains. So, both on-line and off-line parts of the system have two submodules analyzing the behavioral patterns of the visitors in terms of path and identity information.

The general architecture of the system that we developed is depicted in Figure 1.1. As depicted in Figure 1.1, off-line module of the system consists of two major modules, PathInfoProcessor and HostIdentityProcessor. These modules mine the log files off-line and store the results that they obtain into the files called PathInfoFile and HostInfoFile. Besides, on-line module of the system uses PathInfoFile and HostInfoFile together with the log file of the web site for producing on-line recommendations for the new visitors. Details of the system architecture, mining techniques and recommendation process are described in the following chapters.

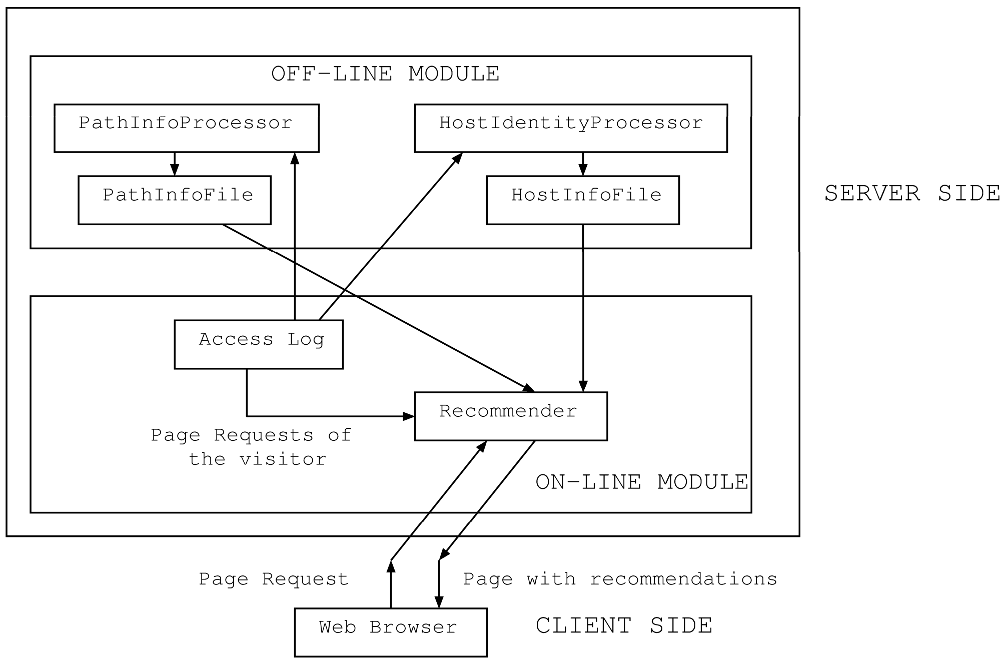


Figure 1.1: General Architecture

In this thesis, we firstly give an overview of the literature published about Web Usage Mining in Chapter 2. In Chapter 3, we will demonstrate the design and implementation details of the off-line module of our system. Chapter 4 is devoted to the details on on-line module. Experimental results will be presented in Chapter 5. Finally, we will conclude with Chapter 6.

Chapter 2

Background

In this chapter, we will give an overview of the existing research on Web Usage Mining. We borrowed the taxonomy of Web Usage Mining studies from [26]. General Web Usage Mining, Site Modification, System Improvement and Personalization are four of the main categories of the research going on related to the Web Usage Mining. General Web Usage Mining systems aim to discover general trends and patterns from the log files either by adapting well-known data mining techniques or by proposing new data mining techniques. The objective of the Site Modification systems is to improve the design of a web site by suggesting modifications in its content and structure. The research on System Improvement focuses on utilizing the web usage mining for improving the web traffic. Lastly, personalization systems aim to understand individual trends to be used for personalizing the web sites. Through out the following sections, we will give detailed description of the projects belonging to each category.

2.1 General Web Usage Mining Systems

General Web Usage Mining systems focus on the analysis of log files using data mining techniques for discovering general access patterns and trends of users [26]. Majority of the studies under this category aim to discover user navigation paths from the log files. In general, navigation path of a visitor is mainly the path followed by him/her through out his/her visit to the web site. It should be noted that each different study enlarges this definition by determining some specifications on what can be accepted as a path. Systems also differentiate on how they use the paths that are found. Some of the general usage mining systems present the user navigation paths without any further processing [4, 1, 15]. Some others provide a query language for analyzing the paths better [25, 7]. In addition, there are studies on clustering paths or just user sessions with the aim of finding similar interest groups among visitors [8, 21]. The other types of studies under this category proposes to adapt well-known data mining techniques such as association rule mining to the problem of web usage mining [6]. In this section, we will explain each of these studies in detail.

2.1.1 Mining Traversal Paths

One of the existing approaches for mining navigation patterns from log files is to make use of well known techniques from data mining. The work reported in [4] is an example for such a study. The aim of the work is to find frequently occurring paths which are named as maximal reference sequences from the log file by using a methodology similar to the association rule mining.

The first step of the proposed solution procedure is to traverse the whole log file for finding maximal forward references for each user. To be able to do that, the log file is divided into user paths where each path contains the accesses belonging to a specific user. Then, each user path is processed to find maximal forward references contained it. A maximal forward reference is defined as a sequence of pages that are visited consecutively by the visitor in which each page is seen only once. Whenever a backward reference to a page previously visited is seen, the current maximum forward reference path is terminated, added into the database and a new one starts. While travelling through the pages, visitors generally turn back to the previously visited pages and choose other links from them. It is pointed out in [3, 4] that the pages seen on the way back to the previous pages are visited only because of their location, but not their content. In the light of this observation, the study concentrates only on forward references.

As an example, assume that the traversal sequence of the Visitor A is as follows: (P1, P2, P3, P4, P3, P2, P7). In this example, Visitor A turns back to the page P3 and P2 consecutively after retrieving page P4. Here, the page P3 is retrieved only for being able to retrieve page P7 from page P2. The algorithm forming maximal forward references solves that problem by removing the backward references from the paths. The algorithm produces the following maximum forward references for Visitor A: (P1 P2 P3 P4, P1 P2 P7)

Once the maximal forward references for each user are formed, the next step of the solution for mining path traversal patterns is ready for the execution. In this step, the database containing maximal forward references for all users is processed to be able to form large reference sequences which are frequently occurring consecutive subsequences among all maximal forward references. Full Scan (FS) and selective scan (SC) are two different algorithms for finding the large reference sequences [4]. FS algorithm is indicated to be similar to the well known algorithm called Direct Hashing with Pruning (DHP) for mining association rules with adaptations to the current problem. Because, the problem of finding large reference sequences from the database of maximal forward references has common points with finding large itemsets from the database of transactions in association rule mining. The main difference between these two problems is that the order of the items in an itemset is not important in association rule mining while it is crucial in mining traversal patterns. So, Full Scan algorithm changes the joining strategy used in the candidate generation phase of the DHP algorithm. Selective Scan algorithm is similar to Full Scan algorithm with optimizations to reduce I/O cost. Detailed description of the mentioned algorithms can be found in [4].

Maximal reference sequences are the subset of large reference sequences so that no maximal reference sequence is contained in the other one. For example, if the large reference sequences are AB, AE, AGH, ABD then maximal reference sequences become AE, AGH, ABD. It is indicated in [4] that the sequences obtained through this way can then be used by web masters in redesigning the links between the pages that are accessed together in making marketting decisions [3].

2.1.2 Mining Navigation Patterns with Hypertext Probabilistic Grammars

Another study towards the problem of mining access patterns of visitors proposes to model user navigation sessions as a hypertext probabilistic grammar (HPG) [1]. A user session is defined as a sequence of page requests coming from the same machine where the time passing between each request is less then a certain time limit. After the HPG is formed, the paths followed frequently by the visitors are discovered by applying a special case of depth-first search algorithm on it.

The complete formalism behind HPG is beyond the scope of this thesis and interested readers may refer to [1]. In short, each terminal and nonterminal symbol of HPG built from user sessions corresponds to a web page and there is a one-to-one correspondence between terminal and nonterminal symbols. The links between web pages are represented by the production rules of the grammar. Two additional states, S and F are added into the grammar to represent the start and finish of the paths. In the corresponding automata, states represent nonterminal symbols where transitions between states are formed by productions. Each production originating from a state is attached with a value which is the probability that the link corresponding to a production was chosen from the links on a page represented by that state. In case of a start state, the probabilities of the productions are derived from the rate of the number of times that the page is visited to the overall number of hits.

When navigating through a web site, visitors may concentrate on unrelated topics in a single session [1]. Accordingly, the concept of N-Grammar is suitable to be employed in building HPG. N-Grammar dictates that the link that will be chosen by a visitor on any page is effected only by the last N pages retrieved by him/her. In HPG that makes use of the concept of N-Grammar, the number of states may increase too much if N is chosen to be very large. This is due to the fact that each distinct consecutive sequence of N pages visited by any user should have a corresponding state in HPG.

After the construction process, user preferred paths are discovered from HPG by applying depth-first search like algorithm [1]. Before mining, the mining expert should specify support and confidence thresholds which will effect the quality of the paths discovered. Support threshold ensures that the path is frequently visited while confidence threshold ensures that the derivation probability of the corresponding string is high enough. The support value for a particular path is obtained by looking at the probability of the derivation of the first state of this path from the start state. In addition, the confidence value is obtained from the derivation probabilities of other the pages on the path. By the help of support and confidence thresholds, it becomes possible to discover the paths that describe the common visitor behavior best [1].

2.1.3 Analysis of Web Logs through OLAP Mining

A totally different approach to web usage mining is to make use of OLAP (On-line Analytical Processing) technology on mining process. WebLogMiner [29] is one of the tools which aim to incorporate the OLAP technology and the data mining techniques. OLAP techniques are being used to obtain a portion of the data that is interesting to the analyst who can also determine the abstraction level on which the data will be presented. Then, this data can be used as an input to the data mining algorithms. Results obtained through mining the data can also be presented in different ways by using OLAP techniques. So, the mining process becomes more interactive and flexible.

OLAP technology firstly places the data into a data cube which is stored in multidimensional array structures or relational databases. Each dimension of the data cube represents a distinct field of the data, such as URL or domain name. If the data cube has n dimensions, each cell is characterized by having distinct values for the fields represented by these dimensions. Each cell in the datacube stores the number of visitors which have the same values with the values characterising the cell. The advantage of the data cube representation is to make it possible to view from different perspectives and abstraction levels which are performed by the OLAP operations such as drill down, roll up and slice. The following statements are examples for the simple queries that OLAP can answer quickly:

- Hits coming from Turkey, between March 2001 and May 2001

- Hits coming from edu domain on 23/03/2001 with agent Mozilla

In addition to the analysis performed by OLAP technology, WebLogMiner makes use of data mining techniques to analyze the data to answer questions that OLAP can not. For this, it applies well-known data mining techniques such as association rule mining, clustering or time-series analysis on the data stored in the data cube. For example, by performing time-series analysis, it answers the following questions:

- What are the typical page request sequences performed by the visitors? Namely, are there request sequences that are common to most of the visitors?

- What are the event trees belonging to specific time intervals? Here, event trees contains the traversal patterns of the visitors in an aggregated form.

- How the traffic on a web site changes depending on time? Are there particular trends on particular times of a day, month etc..

2.1.4 Web Utilization Miner

Web Utilization Miner (WUM) is another data mining tool designed for mining user navigation patterns from web logs [23, 24, 25]. The distinguishing facility of this tool is to provide a mining language by which users can dynamically specify constraints on the mining result. It is indicated in [24] that being an interactive is an important plus for the data mining system, because interactive mining systems are capable of responding to the specific interests of the users. WUM is composed of two major modules: Aggregation service and the query processor [25].

Aggregation service firstly processes the log file and divides it into the visits that are used in constructing the aggregate tree on which the mining will be performed [25]. The tree is extended by adding each path seen on the log file.

While forming the tree, paths that have a common prefix are merged. So, all paths are represented in the tree at the end. Each node in the aggregate tree contains a URL, occurrence count and the number of visitors reaching that node by following the path starting from the root node. Because there exits visits that contain the same URL more than once, each node is associated with an occurrence count to show which occurrence of the URL this is. The way of storing paths in an aggregate tree was chosen for reducing the space requirements and speed up the mining process. The aggregate tree is built for once and used as input for the other module.

Query processor is the module that performs the interactive mining on the aggregate tree constructed by the Aggregation service [25]. The user can specify structural, textual and statistical constraints on the mining result. For example, the following query, which is expressed in MINT syntax, will result in a graph showing navigation patterns between B.HTML and any page whose support is larger than 1. The wildcard between the nodes means that there may be any number of nodes between X and Y. But the order of the template variables should stay the same as given in the query.

```
select T
    nodes as X Y, template X*Y as T
    and X.name= B.HTML
    and Y.support<1
```

The algorithm that is used for mining according to the given template and other constraints firstly finds all possible bindings for all template variables [23]. At first, all possible bindings for the first template variable are found by checking all nodes in the aggregate tree. This is the first and the last time that the whole tree is processed by the query processor. After that, only the trees rooted at the nodes that contains the URLs bound to the first template variable are processed for finding the possible bindings for the second template variable. Each different binding obtained for the template variables is named as pattern descriptor. That is, pattern descriptors contain identifiers that match to the template variables in given queries and wildcards. Namely, in this step of the algorithm, the structural and textual constraints specified by the user are taken into consideration [23].

Next, the algorithm obtains navigation pattern corresponding to each pattern descriptor found in the first step. The designers of the system define a navigation pattern as a graph formed according to the pattern descriptor. For each pattern descriptor, the algorithm firstly finds all branches of the Aggregate Tree that contains the pattern represented in that descriptor. Then, these branches are merged at their common prefixes and on the identifiers existing in the pattern descriptor. While merging the branches, the counts of the nodes that are merged together are added. After merging, the statistical constraints on the mining result are checked and a descriptor is ignored if these constraints are not satisfied. At the end, remaining navigation patterns are shown to the user.

2.1.5 WebSift

WebSift is a web usage mining system which aims to apply well-known data mining techniques on the usage data obtained through web logs [26, 7]. It divides the mining process into three main phases: Preprocessing, Pattern Discovery and Pattern Analysis. The aim of the Preprocessing step is to turn the raw data in the log file into a form that is suitable for mining. Then, in the second phase of the usage mining process, well known data mining techniques such as association rule mining, sequential pattern mining or clustering are applied on the transactions obtained in the previous phase. In the third phase, creators of the WebSift system propose to provide a query language and visualization facilities. Also, in this phase of the mining process, uninteresting results are filtered by using the Information Filter [7].

Preprocessing step includes cleaning the data, identifying users and sessions belonging to them, completing the missing references in paths and formatting the data to obtain the appropriate transaction type for the type of the mining operation that will be performed [5]. Data cleaning is the removal of irrelevant and redundant data in the log file such as requests for graphics. Besides, user identification tries to identify the requests belonging to each user. Authors indicate that an IP address may not be suitable to differentiate between the users, because two visitors may be using the same IP at the same time. For the solution of this problem, they propose to make use of some heuristics. For example, if two requests come from different types of browsers from the same machine, these requests are accepted to be performed different users. After the users are determined, the accesses belonging to them are divided into sessions. Then, the pages that are not recorded but accessed by the visitor are determined and added to the sessions. Detailed explanation of the reason for such situations is given in Chapter 4. In addition, a detailed analysis of data preparation techniques used in web usage mining are presented in [5].

In the Information Filter, the interestingness level of the rules are determined by looking at the site structure [7]. Currently, the system proposed in [7] is capable of determining the interestingness level of frequent itemsets and association rules with two different techniques: BME (Belief Mined Evidence) and BCE (Beliefs with conflicting Evidence). BME finds the frequent itemsets which contain pages that are not directly linked. Frequent itemsets that contain linked pages are not that interesting because it is already guessed by the site designer who put a link between them. On the other hand, if many visitors retrieve pages that have no link in between together, this may be an interesting result for the site designer who may notice a deficiency in the site design. The pages that are linked, but not in the same frequent itemset may also be interesting. BCE finds that kind of pages. The result shows that the link between these pages is rarely used by the visitors which may give a clue to the site designer for the removal of this redundant link.

2.1.6 WAP Mine

Wap (Web Access Pattern) Mine is an efficient data mining algorithm for discovering web access patterns from the Wap Tree (Web Access Pattern Tree) which is a compact data structure designed for storing the data obtained from the logs [15]. The end result of the algorithm is the set of frequent access patterns which contain pages requested sequentially by enough number of visitors.

Wap Tree is formed by the addition of the frequent access subsequences that take part in the log in hand. Before the construction of the Wap tree, the log is traversed once for finding frequent 1-sequences, URLs that are seen in efficient number of user sessions. Then, the URLs that are not frequent are filtered from the sessions resulting in a frequent access subsequences. Wap Tree is constructed by merging the frequent access subsequences on their common prefixes. Another feature of the Wap Tree is that all nodes that contain the same URL are linked into a queue and another data structure, header table, contains a pointer to the head of all queues.

The foundation of the Wap Mine algorithm is based on a heuristic called Suffix Heuristic [15]. Suffix Heuristic says that if an event (page reference) e is frequent in the prefixes of sequences that have a suffix which contains pattern P as a subsequence, then eP is also a pattern. The algorithm for finding frequent sequences based on that heuristic is named as conditional search which is employed in Wap Mine for mining sequences. Wap Mine algorithm processes each event one by one. For each event (page reference) e_i, it firstly forms the conditional tree for that event. Conditional Wap tree for e_i contains the set of prefixes of the subsequences that contain e_i as a suffix. After this, the algorithm continues to mine Conditional Wap Tree recursively. Finally, the results obtained from mining conditional Wap Tree are concatenated with e_i. Page sequences obtained through this algorithm correspond to the frequent access patterns.

2.1.7 Clustering User Sessions

Another study towards Web Usage Mining proposes to cluster visitors of a web site based on the page requests taking place on the sessions belonging to them [8]. The aim of this study presented in [8] is to discover the groups of pages that are visited together by many visitors. This information can then be used by the Web master in redesigning the Web Site or updating it with extra links between these pages.

In this study, a log file of the web site is initially divided into user sessions. For clustering, each user session should be represented with a vector of pages in which each entry correspond to the time spent on that page. But, it is indicated in [8] that in a web site that have lots of pages, the size of the vectors will increase dramatically. Because, the vectors should have an entry for each page in the web site. To overcome this, session vectors are generalized by using Attribute Oriented Induction method.

Entries in the session vector are generalized by looking at the page hierarchy if the web site. A page hierarchy can be derived from the directory structure of the server and represented in the form of tree. The leaves of the tree correspond to the URLs. The parents of the leaf nodes keep the names of the web pages corresponding to innermost directories containing the URLs represented by their children. The parents of the non-leaf nodes are the web pages representing outer directories containing them. The directories are derived directly from the names of the URLs. For example, the parent of the URL *http://www.umr.edu/~regwww/ugcrc97/ee.html* is *http://www.umr.edu/~regwww/ugcrc97*.

By using the page hierarchy obtained by this way, the pages in the sessions are generalized as much as specified by the mining expert by using the tree climbing method from attribute oriented induction [8]. During generalization, each page is replaced with one of the its parents depending on the level to which pages are generalized. After generalization of the pages in the session vector, the duplicate general pages are merged into one by adding the times spent on them. By this way, the size of the session vectors are decreased dramatically, because the number of general pages is smaller than the number of URLs.

Sessions obtained by this way are then clustered by using BIRCH hierarchical clustering algorithm. In BIRCH, clustering is performed by using a tree structure which it calls as CF tree. In CF tree, leaf nodes contain the current clusters which are the collection of session vectors while the non-leaf nodes store CF vectors which characterize the clusters below them. When a new session vector should be placed into the tree, it goes until a leaf node by choosing the branches that are the closest to him/her. The CF vectors of the parent nodes are updated accordingly. In case there is no matching entry in leaf with given thresholds, the session vector is put into a new entry in the leaf node. If there is no empty entry, the leaf is split into two which may cause additional splits in the parent nodes.

2.1.8 Clustering of the Paths Followed by the Visitors

Another study which makes use of the clustering techniques for analyzing web usage data was presented in [21]. Different from the most of the other usage mining systems, the system presented in [21] provides a profiler for obtaining more accurate, reliable and detailed information about the behavior of the visitors of the web site. The data obtained by this profiler is then used for obtaining the paths followed by the visitor and times spent on each page of the paths. Then, these paths are clustered by using Path-Mining method explained in [21].

The profiler provided by the system works on the client side. It is a Java applet loaded into the client side with the first page request and staying in the client cache afterwards. A call to this applet is added to each page in the site. The aim of the Java applet is to determine exact viewing time of the pages and catching the page views missing due to the retrieval of them from the client cache instead of a server. In addition to the profiler, each link on each page is updated to make it transfer more information to the server side when clicked. This information will be used for determining which links are selected by the visitor. Then, the link names will be added into the paths so that each pair of page requests are separated by the link which is selected for retrieving the second page. The authors indicate that link information may provide additional clues on user behavior especially if two links from the same page are pointing to the same page.

After the page requests and times spent on them are determined for each visitor, paths found are clustered for obtaining the groups of visitors with similar interests by using the Path-Mining methodology [21]. Via the usage of this methodology, the order of the requests in the path is also taken into consideration on the contrary to the work explained in the previous section. To be able to this, the system needs a way of measuring the similarity between two paths. The similarity between two paths is measured by finding the angle between them. Interested readers can refer to [21] for the way of obtaining this angle. Briefly, the angle between two paths are calculated by using the inner product over the feature space where feature space contains all sub paths of these two paths. After the angles between each pair of paths are calculated, the results are fed into k-means algorithm for finding clusters of paths. The resulting clusters are considered to be containing groups of visitors with similar interests [21].

2.2 System Improvement

Research on System Improvement aims to utilize web usage mining for improving the web traffic and increasing the speed at which the visitors are responded. One way to do is to provide the web server with the capability of guessing the pages that may be retrieved by the visitors next and generate the dynamic content of these pages before user retrieves them.

For guessing the pages, the system proposed by Schecter et. al make use of the concept of path profile which is constructed from the data contained in the web logs [20]. Path profile is the set of paths followed by the visitors of a site and the number of people following them. This system provides an efficient technique for generating and storing the path profiles. The paths are stored in the form of a tree in which the paths are merged on their common prefixes. While constructing the tree, only the paths whose maximal prefix is seen in at least T of the paths are added into the tree for reducing the memory cost. By following this rule, the algorithm for forming the tree is run on the tree more than once to be able to obtain all paths suiting to the threshold value.

A path profile is then used by on-line working part of the system for guessing the next access of the user. The system starts with the shortest suffix of the current user path and tries to find a path whose maximal prefix matches with it. As long as a matching path whose maximal prefix equals to the suffix in hand is found, suffix size is increased. Assume that the pages retrieved by the visitor A are as follows: [P1, P2, P3]. In that case, the system initially checks the tree for finding the paths that have a maximal prefix [P3], the smallest suffix of the user path. Then, the suffix size is increased by one and the paths that have [P2 P3] as maximal prefix are found if there exists any. Assume that [P2 P3 PY] is such a path. Increase in the suffix size continues as long as the corresponding paths are found. Assuming that there is a path [P1 P2 P3 PX] in the tree, the system creates the dynamic content for the page PX automatically. If there exists no path having [P1 P2 P3] as maximal prefix, then the system will create the dynamic content for the page PY.

Another prediction technique proposed in [20] is named as point based prediction in which the next page is guessed only by considering the last page retrieved, not the whole path. Experiments with the system show that agreement prediction technique gives the most accurate results. In this technique, dynamic content creation of a particular is done only if both point and path based prediction techniques agree on that page.

2.3 Personalization

As a result of the increasing demand to the e-commerce, many companies are eager to make their sites that exhibit their products more serviceable and effective for their visitors to be able to turn them into customers [13]. The number of people visiting the web site of a company may be too high whereas only small percentage of the visitors may be turning into a customer. The number of customers gained through web site heavily depend on the success of the site and personalization is critical aspect of this success. Web Personalization simply means to understand the needs and interests of the visitors of the site and respond accordingly. Such a web site recognizes each visitor and customizes itself by various ways such as determining the information that should be shown to the visitor or automatically changing the site structure in a way that will be useful and attractive for the current user. Personalization is attractive research topic, because it is critically important for the success of e-commerce companies. Some of the different techniques for personalization will be explained in the subsequent sections.

2.3.1 Content Based Filtering

The main idea of Content Based Filtering is to make use of content similarity between stated user interests and web pages for personalization. WebWatcher [27, 19] is an agent that trusts on content based filtering for personalization of the web sites. It guides visitors during their navigation through the web site according to their interests. At the beginning of a visit, WebWatcher asks user to enter his interest or the thing that he is looking for in the form of keywords. By using this information, WebWatcher highlights the links that are best suited to the needs of the visitor on each page retrieved by him/her through out his visit.

WebWatcher accomplishes the task of choosing the best links for that user by using the information learned from the past users. In addition, the actions performed by each visitor are continuously used as training samples for improving the performance of the tool in future recommendations. Three different learning techniques are tried in WebWatcher: Learning from previous tours, Learning from Hypertext Structure and the combination of first two [27].

First method proposes to store a description for each link in each page in the form of a high dimensional feature vector whose each element is an English word. Interests of the users are also represented by a feature vector. Whenever a visitor follows a link in a page, the interests of the user which consists of some number of keywords is added to the description of that link. What is used in choosing the links to highlight in each page is the descriptions of the links determined as a result of this learning mechanism. While choosing the links that will be recommended to the visitor, WebWatcher calculates the similarity of each link in the page to the interest of the user. The links that will be highlighted are the ones with the highest similarity values with the user interest.

To learn from the hypertext structure, WebWatcher makes use of Reinforcement Learning [19]. If we take an agent moving across states as an example case, the aim of the Reinforcement Learning is to train the agent so that it will reach the final state from the initial state by choosing the best action to take in each state it encounters. Here, the action means choosing a next state to go. Goodness of choosing an action a in state s is represented as $Q(s, a)$. In [27], it is indicated that the optimal strategy is to choose an action that will maximize the Q value for the current state. Turning back to hypertext environment, pages are the states and links are the actions. The system learns $Q(s, a)$ function for each page and word pair which means that the best action to take is different for different words in the same page. So, in each page the system recommends the hyperlinks which maximize the total of Q values belong the current page and words given as an interest of the user.

The third method which is detected to be giving the best results combines the results obtained from first two methods plus two additional methods [27]. The first additional method chooses the links that are mostly preferred while the second method chooses the links whose textual content is most similar to the interest words of the current user.

2.3.2 Usage Based Web Personalization

Most of the recent research on personalization aims to incorporate pattern discovery with personalization, resulting in a usage based Web personalization or customized usage tracking [13]. In that case, profiles of the visitors are dynamically created according their access patterns. Dynamic creation of profiles is advantageous when compared to the profiles specified by the visitors themselves. Older personalization tools and techniques rely on that kind of profiles which are static and most probably biased. As the time passes, user preferences may change although static profiles remain unchanged which decreases the performance of the personalization system. On the other hand, dynamically created profiles capture the current interests of users. Dynamically created profiles can not be hundred percent faultless, but they achieve considerable amount of success in helping users without waiting for the user asking for it.

Usage-based Web Personalization systems generally comprise two major components: Off-line component and on-line recommendation engine [13]. Off-line component of the system analyzes the log files which contain the footprints of all visitors visiting the site. It firstly puts the data in the logs into a form that is amenable for applying data mining techniques. Analysis of log files by various data mining techniques results in aggregate usage profiles which are common profiles of visitors of the web site. Then, the on-line component of the system matches the current user to these profiles based on his navigation pattern up to that point and customizes the current page accordingly. Customization can be done through recommending some links or putting advertisements or product news that may interest the customer. Through out the following sections, we will explain current Web Personalization systems and tools in more detail.

2.3.2.1 Analog

Analog [28] is one of the first usage-based Web Personalization systems. Its off-line module clusters the users of a web site according to their access patterns. Off-line module firstly processes the log file of the target site to find out user sessions which are represented as n dimensional vectors where n is the number of distinct pages in the site. The weights of the entries corresponding to the pages visited by the visitor is larger than 0, while the weights for non-visited pages are zero in the session vector. So, the system does not take into account the order in which pages are retrieved. After all session vectors are obtained in this way, LEADER algorithm is applied to find clusters. LEADER is a simple clustering algorithm which has some drawbacks. After clustering is completed, median vector of each cluster is computed as a representative of the cluster.

The on-line module of the system recommends some links to the active visitors by looking at the pages that they retrieve before. Active user sessions are represented as n dimensional vectors as in the off-line module. Whenever user retrieves a new page, the session vector belonging to that user is updated accordingly. On-line module of the system tries to match the active user session to existing clusters. User session is accepted to be matching to a particular cluster if the number of common pages between user session vector and cluster median is larger than some threshold value. The pages in the median vectors of matching clusters are then recommended to the user if they are not already retrieved by him/her [28].

2.3.2.2 Web Personalizer

Web Personalizer [13] is one of the other systems that makes use of the explained framework for usage based Web Personalization. The main aim of the off-line module of the system is to obtain aggregate usage profiles which are represented as weighted collection of URLs. The reason for preferring this representation style is to be able to make use of classical vector operations that are used in clustering. Two different methods for forming the aggregate usage profiles are presented in [13].

The first method is to cluster the user sessions by using standard clustering algorithms for grouping the visitors that have similar interests together as in Analog. This method proposes to represent each user session as a n-dimensional vector where n is the number of distinct URLs that exist in the user sessions. The values kept in each entry of the user session vector can be chosen to be binary to indicate the existence or nonexistence of that URL in that session. After putting them into the vector form, user sessions are clustered by using classical clustering techniques from data mining to obtain session clusters. The next step to form Aggregate Usage Profiles is to find the mean vector of each cluster. Entries in the mean vector of a cluster are calculated by finding the ratio of the number of user sessions that contain the URL that is represented by that entry to the total number of sessions in that cluster. As a result of this calculation, some of the URLs are filtered out because of having very low support, which means that only minority of the user sessions in the cluster contain them. The resulting mean vectors are representative aggregate usage profiles for the log data processed.

The other method proposed in [13] clusters URLs instead of sessions. It is indicated that users that have very different sessions may have common interest to a group of URLs. This information will remain undiscovered with the previous method. At the end of this method, each cluster will contain a set of URLs which tend be together in majority of the sessions. Standard clustering algorithms are difficult to be applied in that case because of the nature and the size of the feature space which consists of the sessions. As a consequence, another clustering technique which is named as Association Rule Hypergraph Partitioning (ARHP) is employed by this method. The hypergraph to be clustered by this technique composed of URLs as vertices and the frequent itemsets as the hyperedges which connect the vertices representing the URLs in that itemset. As known, frequent itemsets are formed by a well known technique from association rule mining. Application of the ARHP technique on the hypergraph obtained by this way results in a set of clusters which contain a set of URLs that are frequently accessed together. Usage profile for each session is obtained by associating a connectivity value of the vertex as a weight for the corresponding URL.

As it is in the other usage based personalization systems, on-line component of the system keeps track of the active user sessions to recommend some links attached with significance scores to the users. The way to do this is to find aggregate usage profiles that match to the current user session best. The matching scores are calculated by standard distance and similarity measures between vectors. Besides, site structure becomes effective in calculating the matching scores by increasing the score of the pages that are farther away from the current page.

History depth is an important concept employed by the on-line component of the system for obtaining more successful recommendations. It determines the number of previous pages that will be effective on the recommendation. It is indicated that user sessions are mostly composed of some number of episodes which are paths followed for reaching different kinds of information. The length of episodes is indicated to be 2 or 3 in general. So, by the help of the concept of history depth, it is aimed to make recommendations based on the pages retrieved only in the current episode.

2.4 Site Modification

Another way of benefiting from the usage data discovered from the logs is to use it to improve the design of the web site. In personalization, web sites are dynamically customizing themselves differently for each visitor. On the other hand, site modification systems offer static changes in the structure and content of the web sites to meet the needs of all visitors [17].

IndexFinder [17, 18, 16] is one example for that kind of tools. It aims to discover index pages whose addition is very likely to improve the site design. These pages which are created off-line consist of links to the conceptually related, but currently unlinked pages which coexist in most of the user sessions. The addition of automatically created index pages to the site is performed with the authorization of the Web Master. Index page creation in IndexFinder is performed in three phases: processing logs, cluster mining and conceptual clustering [16].

In the first stage of the algorithm proposed in [17], a log is processed to be divided into visits. What comes next is the calculation of the co-occurrence frequencies between each pair of pages to determine to what extend these pages are related. Co-occurrence frequency between two pages is simply calculated by taking the minimum of the two probabilities, probability of the existence of first page in the visit given the fact that second page is in the visit and visa versa. The co-occurrence frequency between linked pages is taken as zero to avoid uninteresting clusters. After the co-occurrence frequencies are calculated, a similarity matrix is constructed which is then converted into a graph form by taking the pages as nodes and co-occurrence values as edges between these nodes. Naturally, the nodes will be unlinked if the co-occurrence frequency between the pages denoted by them found to be 0 from the similarity matrix. The connected components in the graph built in this way are accepted as clusters. The pages corresponding to the nodes of a connected component found by this way are put into one cluster.

The Cluster Mining algorithm explained in [18] is PageGather which differs from the other clustering algorithms because of not insisting on putting every instance in one and only one cluster. Instead, the algorithm discovers small number of high quality clusters.

The clusters obtained by this way may contain pages that are conceptually unrelated. Yet, the aim of the IndexFinder system is to produce index pages that contain links which are conceptually related in addition to be visited together by the majority of the visitors. This constraint is satisfied by applying a concept learning algorithm on the clusters found in the previous step [16]. To be able to apply that algorithm, first each page should be tagged manually with the correct values for predefined enumerated concepts. Concept Learning algorithm [16] finds the most common and basic concept that summarizes the pages in the cluster. Then, the noisy pages that conflict with the concept found are removed from the cluster while the nonexistent pages that conform to the given concept are joined to it. Each cluster obtained by this way is used to form one index page which is composed of the links to the pages that are in that cluster. The candidate pages are presented to the web master who will give the final decision on the addition of these pages to the site and the location and the title of them.

Chapter 3

UsageMiner

The aim of the UsageMiner module is to discover the common usage patterns of the visitors of a given web site by analyzing the log file of the web site. UsageMiner has two independent submodules: PathInfoProcessor and HostIdentityProcessor. The aim of the PathInfoProcessor submodule is to find user navigation paths hidden in the given log file and then store them in a predetermined form to be utilized by Recommender. On the other hand, HostIdentityProcessor submodule examines the log file for discovering the relations between identity information and navigation patterns of visitors and stores the results that it obtains.

Access logs of all web sites continuously grow bigger as new requests are taken by the web servers which maintain these log files. In this respect, we configured our usage mining system in such a way that it runs automatically on each day at a predetermined time for processing the newly added entries of the log file. For this, as soon as the submodules start execution, they determine the location of the last entry in the growing log file so that they know from where to start execution on the next day. So, on each day the submodules start processing the entries in the access log from where they left on the previous day. In other words, they skip the entries until that line and process the ones that are added afterwards. The information learned from the new entries is then aggregated into the information learned in the previous days.

Before going into the details of this process, we will first explain the first critical operation performed by the UsageMiner module. This operation is called as data preparation in majority of the usage mining systems, since in this step the data is filtered and put into a form that is suitable to be mined. In Section 3.1, we will explain some of the data preparation techniques utilized by the UsageMiner. Following this, we will explain the work performed by PathInfoProcessor and HostInfoProcessor modules in more detail.

3.1 Data Preparation

The only data source for our usage mining system is the server log files of the web site considered. Log files contain a huge amount of data which is irrelevant to the usage mining process. In data preparation phase, raw data contained in the log file is filtered out to eliminate these irrelevant entries. The second important data preparation task is to put the relevant data into a form that is amenable for mining. In addition to the common data preparation tasks, both PathInfoProcessor and HostInfoProcessor require specialized and distinct data preparation operations. We will explain these specific operations in the sections devoted to the submodules. On the other hand, data preparation tasks common to the both submodules will be explained in this section. First of all, we will give a detailed description of the server log files.

3.1.1 NCSA Combined Log Format

Main data source for our usage mining system is the server log files which are created and maintained by the web servers. Server access log files store an entry for every single request the server gets. The format of the log file produced by the web server depends on the configuration of the server. Two of the possible log file formats are *common log format* and *combined log format* which differ in the amount of information that they store related to each request. The log files of the Computer Engineering Department of Bilkent University web server (www.cs.bilkent.edu.tr) are created in the NCSA combined log format. Figure 3.1 demonstrates a fragment of the log file of this web site.

```
d75246a.dorm.bilkent.edu.tr - - [21/Apr/2001:22:06:04 +0300]
"GET /courses.html HTTP/1.1" 304 -"http://www.cs.bilkent.edu.tr/"
"Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)"
d75246a.dorm.bilkent.edu.tr - - [21/Apr/2001:22:06:06 +0300]
"GET / will/courses/CS101/ HTTP/1.1" 304 -
"http://www.cs.bilkent.edu.tr /courses.html"
"Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)"
d75246a.dorm.bilkent.edu.tr - - [21/Apr/2001:22:06:10 +0300]
"GET / will/courses/CS101/examgrades.htm HTTP/1.1" 304 -
"http://www.cs.bilkent.edu.tr/ will/courses/CS101/"
"Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)"
ch4blm.bellglobal.com - - [21/Apr/2001:22:06:11 +0300]
"GET / oulusoy HTTP/1.1" 301 332
"http://www.cs.bilkent.edu.tr/ oulusoy/proj5.html"
"Mozilla/4.0 (compatible; MSIE 5.5;Windows 98)"
ch4blm.bellglobal.com - - [21/Apr/2001:22:06:12 +0300]
"GET / oulusoy/ HTTP/1.1" 200 4992
"http://www.cs.bilkent.edu.tr/ oulusoy/proj5.html"
"Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)"
```

Figure 3.1: Fragment of a combined log file

In the following lines, we will explain what each entry stands for.

- Remote Host: This field contains the hostname of the connecting machine. If the machine does not have DNS hostname, IP Address is used.

  75246a.dorm.bilkent.edu.tr - - [21/Apr/2001:22:06:04 +0300]
  "GET /courses.html HTTP/1.1" 304 -"http://www.cs.bilkent.edu.tr/"
  "Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)"

- Ident and Authuser: Ident is the remote login name of the user. If the requested document is password protected, Authuser field contains the user name. Since usually web browsers do not send this information, this field is empty.

  75246a.dorm.bilkent.edu.tr **- -** [21/Apr/2001:22:06:04 +0300]
  "GET /courses.html HTTP/1.1" 304 -"http://www.cs.bilkent.edu.tr/"

"Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)"

- Date: This field contains the date and time of the request.

  75246a.dorm.bilkent.edu.tr - - **[21/Apr/2001:22:06:04 +0300]**
  "GET /courses.html HTTP/1.1" 304 -"http://www.cs.bilkent.edu.tr/"
  "Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)"

- Request [Method URL Protocol]: Request coming from the visitor is exactly kept in this field. Method is set to GET for page requests or POST for form submissions. URL part is reserved for the related URL. Protocol is HyperText Transfer Protocol used.

  75246a.dorm.bilkent.edu.tr - - [21/Apr/2001:22:06:04 +0300]
  "GET /courses.html HTTP/1.1" 304 - "http://www.cs.bilkent.edu.tr/"
  "Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)"

- Status: Status field contains the return status of the request which shows whether the transfer was successful or not.

  75246a.dorm.bilkent.edu.tr - - [21/Apr/2001:22:06:04 +0300]
  "GET /courses.html HTTP/1.1" **304** - "http://www.cs.bilkent.edu.tr/"
  "Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)"

- Bytes: The number of bytes sent by the server is stored in bytes field.

  75246a.dorm.bilkent.edu.tr - - [21/Apr/2001:22:06:04 +0300]
  "GET /courses.html HTTP/1.1" 304 **-** "http://www.cs.bilkent.edu.tr/"
  "Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)"

- Referrer: The URL that directs the user to the current document. If the user reaches to the current document directly by its address, this field contains "-".

  75246a.dorm.bilkent.edu.tr - - [21/Apr/2001:22:06:04 +0300]
  "GET /courses.html HTTP/1.1" 304 - **"http://www.cs.bilkent.edu.tr/"**
  "Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)"

- User Agent: This field contains the browser and the operating system used by the visitor.

75246a.dorm.bilkent.edu.tr - - [21/Apr/2001:22:06:04 +0300]
"GET /courses.html HTTP/1.1" 304 - "http://www.cs.bilkent.edu.tr/"
"Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)"

3.1.2 User Session

In this section, we will define the basic concepts such as *visitor*, *file request*, *page request* or *user session* which we will use many times throught the rest of the thesis.

- **Visitor:** A person accessing the files on a web site from a particular machine. We try to identify and distinguish visitors by using their IP addresses or host names. We assume that two requests coming from the same IP address or host name are performed by the same visitor. Actually, IP addresses and host names may not be enough for identifying the visitors in some cases where the visitors are behind a proxy server or corporate firewalls. Because, the requests coming from the machines behind a proxy server will contain the IP address of the proxy server instead of the real IP addresses. So, we are not able to differentiate between these kinds of machines as many other usage mining systems. Most accurate solution for this problem seems to be relying on the user cooperation although some heuristics may be used for differentiating between visitors. For example, the requests coming from same IP Address with different agents may be the indication of two different visitors. Nevertheless, we can not guarantee that we will be able to solve that problem completely with these kinds of heuristics.

- **Valid File Request:** Any type of data including graphics, scripts or html pages requested by the visitor and submitted to him by the corresponding web server.

- **Valid Page Request:** Any successfully answered request for one of the actual web pages taking place in the web site in process. We need to differentiate between requests for actual web pages and the other types of files. Whenever a page containing images, sound facilities, etc. is requested by a particular visitor, all of the files utilized by that page are retrieved automatically by the web server which adds a new record to the log file for each of these file transfers. So, we need to filter the log file for obtaining actual page requests. This can be performed easily by checking the filename suffixes of the requested files. The entries which contain request for certain types of files with suffixes such as GIF, JPG, WAV, CLASS, TXT, etc. are filtered before running the mining algorithms. In addition to these, we need to eliminate unsuccessful requests that take place in the log file. A request is unsuccessful if it is not answered by the corresponding web server. These kind of situations can easily be detected by looking at the status field of the log entries. For example, if the status code belonging to a particular log entry is 404 then we should ignore that entry, because the requested page of that entry was not found by the web server.

- **User Session:** Ordered set of page requests performed by a particular visitor on a given time interval, which is generally 30 minutes. According to this definition, there must be less than 30 minutes between the first and last page request in the user session. In Definition 2, user session concept is defined more formally. Before this, we will define the concept of log entry in Definition 1 to form a basis for the next definition.

Definition 1 Log File, L is defined as the collection of log entries where each log entry $l \in L$ has the following attributes:

- $l.ident$ is either the ip address or the visitor
- $l.time$ is the time of the request
- $l.referrer$ is the referrer page for the request and
- $l.target$ is the requested URL

Definition 2 User session S is defined as $S =< ident_S, PR_S >$ where

$$PR_S = (R_1^S.referrer, R_1^S.target), ...(R_m^S.referrer, R_m^S.target)$$

$R_k^S \in L$ and $R_k^S.ident = S.ident$ for $0 < k <= m$

and $R_m^S.time - R_1^S.time < 30$

Each server log is composed of a set of user sessions. For each user session $S \in L$, $ident_S$ is the host name or the ip address of the visitor having that session while PR_S (page requests of session S) is the set of accesses taking place in this session. What we call as access is a (referrer, target) pair corresponding to a valid page request generated by the visitor having this session. As indicated in Section 3.1.1, the referrer of a particular access is the page containing the link that is selected for retrieving the target which is the page requested in this access. The accesses are not added into the user session if they contain a invalid page requests, for example the requests containing invalid URL name. Note that, throughout the rest of the thesis, the term user session is being used to refer the PR set of it in some cases.

To sum up, removing the requests for irrelevant types of files and unsuccessful requests is the first operation performed in data preparation phase. It is followed by the determination of the user sessions embedded in the log file. Mainly, these two operations are the common data preparation tasks for PathInfoProcessor and HostIdentityProcessor. In fact, data preparation and mining are performed interactively in these modules. So, these operations are not totally independent from each other. We will explain how user sessions are formed and how mining is performed in detail in the following sections.

3.2 PathInfoProcessor

PathInfoProcessor which is responsible for mining user navigation paths from a given access log file is the first submodule of the UsageMiner system. First of all, in Section 3.2.1 we will describe what we mean by the term *path* and the method of finding paths from a given user session. Formation of the paths is a special data preparation task performed by PathInfoProcessor. In the latter sections, we will explain how the paths found in data preparation phase are utilised by PathInfoProcessor in detail.

3.2.1 Data Preparation for mining paths

In this section, we will explain the method that we utilized for finding the paths followed by the visitors given the sessions belonging to them. Pages are linked to each other in the web environment. So, they can be retrieved by following the links on other pages in addition to reaching them directly via their addresses. In such an environment, visitors follow certain paths throughout their visit to the web sites. All paths start from a particular page in the site and expand by the addition of the new pages retrieved by following the links on the previously retrieved pages or typing their addresses. In the light of these observations, path of a particular visitor is the ordered list of pages which are requested by him/her consecutively. Our aim is to detect these paths as accurately as possible. Unfortunately, inferring user paths is not an easy task as it seems because of the nature of the web environment.

The difficulty comes with the usage of *BACK* and *FORWARD* buttons provided by most of the web browsers. As known, recently requested pages are cached by the web browsers which display these cached copies of pages if visitors backtracks to them by using *BACK* and *FORWARD* buttons. Since no new page is being requested in such situations, web server does not become aware of the user behavior, so this kind of situations are not reflected into the access log files. As an example, consider the user session depicted in Figure 3.2 (note that we did not give the complete addresses for the pages for saving from space. Namely, in all addresses we ignored the *http://www.cs.bilkent.edu.tr* prefix which is common to all URLs belonging to the CS department web server. For example, */∼sesra* is representing the page *http://www.cs.bilkent.edu.tr/∼sesra.*)

The visitor who is the owner of this session firstly retrieves the page */∼sesra.*

```
U= {(-, /~sesra), /*1*/
        (/~sesra, /~sesra/courses.html), /*2*/
        (/~sesra/courses.html, /~sesra/courses/cs101), /*3*/
        (/~sesra/courses/cs101, /~sesra/courses/cs101/grades.html), /*4*/
        (/~sesra/courses.html, /~sesra/courses/cs102), /*5*/
        (/~sesra/courses/cs102, /~sesra/courses/cs102/lectnotes.html)/*6*/ }
```

Figure 3.2: Example User Session (Note that the numbers attached to the accesses are showing the order of the accesses in the session).

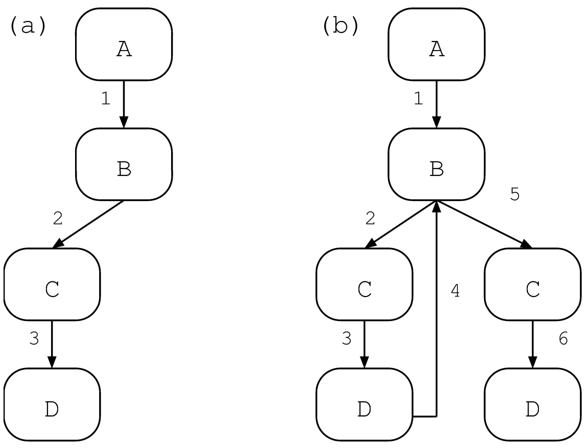
Then, he retrieves the pages */~sesra/courses.html*, */~sesra/courses/cs101* and */~sesra/courses/cs101/grades.html* consecutively. Each of these three pages are retrieved by following the links on the previously retrieved pages. We draw this conclusion by looking at the referrer field of the corresponding accesses. When we come to the fourth access, the referrer page of this access is not the page retrieved in the previous access. On the contrary, */~sesra/cs102* seems to be retrieved by following a link on page */~sesra/courses.html*, because this page is the referrer for this access. This means that the visitor had turned back to the page */~sesra/courses.html* before retrieving the page */~sesra/cs102* in such a way that this move was not noticed by the web server. In this case, we consider that the visitor had pressed the *BACK* button for turning back to the */~sesra/courses.html* page.

Figure 3.3 shows the user session given in Figure 3.2 graphically. In this representation, each page retrieved by the visitor is represented with a distinct box. The paths followed by the visitor can be discovered easily by analyzing this figure. For example, the paths obtained from the user session depicted in Figure 3.3 are listed below.

Path 1: /~sesra → /~sesra/courses.html → /~sesra/courses/cs101 →
/~sesra/courses.html/cs101/grades.html

Path2: /~sesra → /~sesra/courses.html → /~sesra/courses/cs102 →
/~sesra/courses.html/cs102/lecnotes.html

For this user session, there exits one additional path which contains the requested pages in order (Path 3).



A: /~sesra   B: /~sesra/courses.html C: /~sesra/courses/cs101

D: /~sesra/courses/cs101/grades.html E:/~sesra/courses/cs102

E: /~sesra/courses/cs102/lecturenotes.html

Figure 3.3: Navigational patterns in an example user session

Path 3: /∼sesra → /∼sesra/courses.html → /∼sesra/courses/cs101 →
/∼sesra/courses.html/cs101/grades.html →
/∼sesra/courses/cs102 → /∼sesra/courses.html/cs102/lecnotes.html

We have to include this path because it may help us in discovering potentially important information. If supported with the other paths from the other user sessions, this path may be important in drawing the following conclusion: The visitors who are interested in the home page of CS101 course are also interested in the home pages of the CS102 course. At this point, some readers may think that including only the last path could be enough for this session without any further processing. But, if we do not form the second path for that user session, it will cost us losing an important information from which we can draw the following conclusion after checking all user sessions: The visitors retrieving pages */∼sesra* and */∼sesra/courses.html* in order are possibly interested in page */∼sesra/courses/cs102*.

In our system, we benefit from the tree representation for finding the paths hidden in the given user session. For this, the system produces a tree data structure for representing each user session found from the log file. The algorithm

Input : User Session containing a set of accesses
Output : PathFindTree containing paths
Procedure:

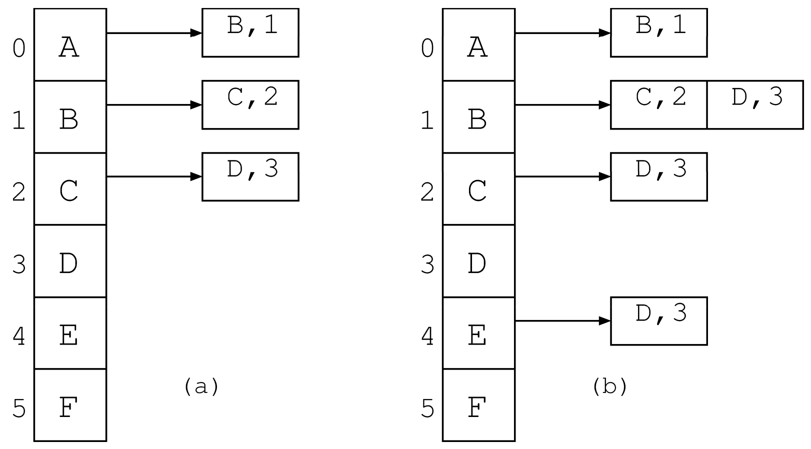
```
[1]IndexOfLastNode ← -1
[2]For each access, (Referrer, Target) pair in the user session
[3]   If this is the first access in the user session
[4]      If the Referrer is empty
[5]         PathFindTree[++IndexOfLastNode].URLname ← Target
[6]      End If
[7]      Else
[8]         PathFindTree[++IndexOfLastNode].URLname ← Referrer
[9]         PathFindTree[++IndexOfLastNode].children[0]← Target
[10]        PathFindTree[IndexOfLastNode].URLname  ← Target
[11]     End Else
[12]  End If
[13]  Else if the Referrer is empty
[14]     lastchild←PathFindTree[IndexOfLastNode].lastchild++
[15]     PathFindTree[IndexOfLastNode].children[lastchild]← Target
[16]     PathFindTree[++IndexOfLastNode].URLname ←Target
[17]  End Else If
[18]  Else
[19]     If the Referrer equals to LastPageRetrieved
[20]        lastchild←PathFindTree[IndexOfLastNode].lastchild++
[21]        PathFindTree[IndexOfLastNode].children[lastchild]← Target
[22]        PathFindTree[++IndexOfLastNode].URLname ←Target
[23]     End If
[24]     Else
[25]        searchindex←IndexOfLastNode
[26]        while (searchindex $\geq$ 0) and ( found == 0)
[27]           if PathFindTree[searchindex].URLname == Referrer then found=1
[28]        End While
[29]        if found==1
[30]           lastchild←PathFindTree[searchindex].lastchild++
[31]           PathFindTree[searchindex].children[lastchild]← Target
[32]           PathFindTree[++IndexOfLastNode].URLname  ←Target
[33]        End If
[34]     End Else
[35]  End Else
[36] LastPageRetrieved ← Target
[37] End While
```

Figure 3.4: Algorithm for forming PathFindTree from a given user session

which forms the tree data structure given a particular user session is depicted in Figure 3.4. In this algorithm, which is called as PathFind, the referrer (or the target if the referrer is empty) of the first access in the user session is put into the root of the tree (Line 3 - Line 15). Starting from this point, the algorithm processes each access in the session consecutively. Whenever the referrer of a particular access which is in process is same as the target of the previous access, new node is created to contain the target page and added as a child to the latest node created (Line 22 - Line 27). In this case, there is no backtracking, because the visitor retrieves a page by choosing a link from the previous page retrieved. In cases where the referrer field of the access is empty, the same thing is applied because the visitor retrieves a new page directly by writing its address (Line 16 - Line 20). We think that if the visitor retrieves a new page by this way, the path should continue with this page.

On the other hand, if the referrer page of a particular access is not the same as the target page of the previous access, which means that the visitor did not follow a link placed on the previously retrieved page, the algorithm takes the necessary actions to handle backtracking (Line 22 - Line 39). At this point, what is known is the referrer page for the current access. So, the action that should be taken is to perform a backward search in the tree data structure for finding the most recently created node that contains the referrer page (Line 29 - Line 32). Whenever such a node is found, the search operation terminates. At this point, the target page of the current access is added as new child to the node found (Line 33 - Line 38). The data structure that we utilized in this algorithm provides an easy access to the nodes containing the most recent accesses to all pages in this user session.

At this point, we will explain the data structure we designed. In fact, the data structure we created is the extension of the tree data structure. The plus of the data structure we created is to make it possible to understand the order in which the tree nodes are created. For this, we keep all nodes of the tree in a linked list. The children of each node are kept in an inner list originating from the corresponding node in the outer list. Each node of each inner list has a field named as *ContinueAt* which shows the index of the child represented by



A: /~sesra   B: /~sesra/courses.html C: /~sesra/courses/cs101

D: /~sesra/courses/cs101/grades.html E:/~sesra/courses/cs102

E: /~sesra/courses/cs102/lecturenotes.html

Figure 3.5: PathFindTree, data structure created for path finding

that node in the outer list. For example, the data structure created for the user session depicted in Figure 3.2 is depicted in Figure 3.5.b.

Figure 3.5.a shows the PathFindTree just before the processing of 5^{th} access in given user session. Here, the referrer page existing in the 5^{th} access is not equal to the target page existing in the 4^{th} access. In this case, the algorithm starts searching the outer list of the PathFindTree starting from the last node added so far, 4^{th} node for finding the most recent node containing the referrer page of the 5^{th} access, which is */~sesra/courses.html*. The search terminates at the first node which contains that page. In our case, 1^{st} node is the most recent node containing the referrer page of the 5^{th} access. So, the algorithm adds a new child to 1^{st} node. New child contains the page */~sesra/courses/cs102*. After the addition of the second path, the data structure is as shown Figure 3.5.b.

After the PathFindTree is constructed in this way, the next step of the path finding process is ready for execution; finding the paths hidden in PathFindTree. The algorithm for finding paths is shown in Figure 3.6. The algorithm starts by finding the set of two node paths that contain the root and the pages retrieved after root (Line 2 - Line 8). Root of the tree contains the referrer page or the target page if the referrer is empty of the first access. After this, the paths are incrementally enlarged in each iteration of the while loop (Line 11 - Line29). Whenever a node that is added into a path has children, the algorithm creates a new copies of the path as much as the number of children minus one. New paths continue to expand independently in the following iterations. The paths obtained after each iteration in our example are shown below.

- first iteration: A B
- second iteration: A B C and A B E
- third iteration: A B C D and A B E F

3.2.2 Mining Process

In the previous section, we explained the method that we used for finding the paths existing in a particular user session. In this section, we will explain the work performed by PathInfoProcessor whose main goal is to mine user navigation paths from the access log files. Main steps of the work performed by PathInfoProcessor is depicted in Figure 3.7. In fact, this section is devoted to the explanation of each of these steps in detail.

Initially, the first action taken by PathInfoProcessor is to load the data stored in the path file produced in the last run of the program into the memory and convert it into a suitable form to be updated by the new data which is the usage data belonging to the visitors visiting the site since the latest run of the PathInfoProcessor. The next step contains the processing of the new data. Initially, the log file is searched for finding the last processed entry in it. The entries after that one are the ones which are new and unprocessed, because of being added into the log file after the most recent run of PathInfoProcessor. In step 2, these new entries are processed for finding the user sessions and the paths hidden in these sessions. These paths are then added into the PathTree.

The algorithm corresponding to the step 2 of PathInfoProcessor algorithm is depicted in Figure 3.8. This algorithm is the one which processes the new

Input : PathFindTree
Output : Paths
Procedure:

```
[1] $k \leftarrow 0$, $lastpath \leftarrow$ -1
[2] while ($k \leq$ PathFindTreetree[0].lastchild)
[3]     lastpath++
[4]     pathlist[lastpath].path[0] $\leftarrow$ PathFindTree[0].URLname
[5]     pathlist[lastpath].path[1] $\leftarrow$ PathFindTree[0].children[k].URLname
[6]     pathlist[lastpath].continueat $\leftarrow$ PathFindTree[0].children[k]. continueat
[7]     pathlist[lastpath].lastnode$\leftarrow$1
[8] End While
[9] while (pathadded == 1)
[10]    pathadded$\leftarrow$0, pathindex$\leftarrow$0
[11]    while (pathindex $\leq$ lastpath)
[12]        if ((nextnode $\leftarrow$ pathlist[pathindex].continueat) > -1)
[13]            numofpaths $\leftarrow$ PathFindTree[nextnode].lastnode
[14]            lastnode $\leftarrow$ pathlist[pathindex].lastnode++
[15]            pathlist[pathindex].children[lastnode]$\leftarrow$
                PathFindTree[nextnode].children[0].URLname
[16]            pathlist[pathindex].continueat $\leftarrow$
                PathFindTreetree [nextnode]. children[0]. continueat
[17]            pathadded$\leftarrow$1, currentnode$\leftarrow$1
[18]            while (currentnode $\leq$ pathtree[nextnode].lastnode)
[19]                lastpath++
[20]                for ($i \leftarrow 0$, i < pathtlist[pathindex].lastnode)
[21]                    pathtlist[lastpath].children[i] $\leftarrow$
                        pathtlist[pathindex].children[i]
[22]                End For
[23]                 pathtlist[lastpath].children[i] $\leftarrow$
                PathFindTree [nextnode]. children[currentnode]. URLname
[24]                    pathtlist[lastpath].continueat $\leftarrow$
                PathFindTree [nextnode]. children[currentnode]. continueat
[25]                currentnode++
[26]            End While
[27]        End If
[28]    End While
[29] End While
```

Figure 3.6: The algorithm for extracting paths from a given PathFindTree

Step 1: Load the past data produced in the previous days into PathTree

Step 2: Update the PathTree with the new entries from the active log file

Step 3: Convert the PathTree into a form that can be handled by Recommender

Step 4: Prune and store the tree

Figure 3.7: Main steps of PathInfoProcessor algorithm

entries of the log file and discovers the user sessions. In this algorithm, the new entries in the log file are read one by one in the order of creation. For each line read, the algorithm checks all user sessions that it keeps in its active user sessions database. Whenever the algorithm encounters with a page request from a new host, it creates a new user session and adds into the database containing all other user sessions (Line 4 - Line 9). On the other hand, if the request was created by an existing host, then it is added into the user session belonging to this host (Line 10 - Line 13). Finally, if the algorithm discovers that 30 minutes has passed since the creation of a particular user session, then this user session is ended (Line 16 - Line 24). In this case, this user session is removed from the active sessions database. Before doing this, the algorithm finds the paths followed by the visitor having that session (Line 17). The paths belonging to the user session are found by using the algorithms explained in Section 3.2.1. After all paths existing in the user session are found, these paths are stored for later use (Line 18 - Line 22). In Section 3.2.3, we will explain the method for storing paths.

3.2.3 Path Tree

After finding the paths for ended user sessions, PathInfoProcessor should store these paths in some form so that they could be utilized by the on-line part of the system. In our solution, paths are stored in the form of tree. StorePath algorithm is responsible for adding the given path into the tree called PathTree.

At this point, it should be noted that paths are not added into the PathTree as they are formed. Whenever a path is found, only its prefix containing first

Input : Log File
Output : PathTree
Procedure:

```
[1] For each new entry in the given log file
[2]   Read the next entry from log file
[3]   Parse the log entry to obtain the Identity, Referrer, Target and Time
      for the request
[4]   if the request is generated by a new visitor
[5]      Create a new user session
[6]      Owner of the user session $\leftarrow$ Identity
[7]      Starting time for User session $\leftarrow$ Time
[8]      First access of the user session $\leftarrow$ (Referrer, Target)
[9]   End If
[10]  Else
[11]     If the request is valid
[12]        Add a new access, (Referrer, Target) to the session belonging to
            the current visitor
[13]     End If
[14]  End Else
[15]  for each user sessions, $S$ active in the session database
[16]     if the session, $S$ is created 30 minutes before
[17]        Find paths for this visitor
[18]        for each path
[19]           for each node, $N$ of the path
[20]              Add the sub-path starting from $N$ and containing
                  the next $HistoryDepth$ nodes into the $PathTree$
[21]           End For
[22]        End For
[23]        Remove the user session, $S$ from the active sessions database
[24]     End If
[25]  End For
[26] End For
```

Figure 3.8: The Algorithm for processing the new entries of the given log file

n nodes is added into the PathTree. The parameter n is called as *HistoryDepth* and it is the maximum number of pages that can take place in any path. The criterion of *history depth* is utilized by the majority of the usage mining systems. According to this criteria, only the last n pages requested by the visitor effect the page that will be retrieved next where n is the history depth. The pages retrieved before this are accepted as irrelevant to the current access. The value of n is generally adjusted experimentally. Utilizing from the history depth criterion requires that it is not necessary to keep paths that have a length larger then n. As a result, if the *HistoryDepth* is set to be n, a path starting at any node ends in n^{th} node of the real path.

In addition, it should be noted that all prefixes of a given path can also be accepted as a path. Because, they also hide a navigation pattern in them. As a result of these, for each path found by the PathFind algorithm, the system considers each n consecutive node sequence starting from each and every node in the original path as a candidate path to be added into the path tree. Naturally, the paths may have smaller lengths if there exists a smaller number of nodes between the starting node of the new path and the ending node of the original path. For example, the paths formed out of a real path (A, B, C, D, E) are (A, B, C), (B, C, D) and (C, D, E) if *HistoryDepth* is set to 2.

As indicated, paths are stored in the form of a tree which is a well-known data structure. The reason behind preferring the tree data structure is the easiness and efficiency that it provides for both off-line and on-line parts of the system by reducing the space and the time requirements dramatically. Consider the following two paths:

Path 1: /~sesra → /~sesra/courses → /~sesra/courses/cs101

Path 2: /~sesra → /~sesra/courses → /~sesra/courses/cs102

First two pages existing in these paths are same. In the tree representation, these two paths are merged, so their common prefix is stored only once. When all paths followed by the visitors are stored in such a compact data structure, recommendation process also gets much more painless. The algorithm forming

Input : PathTree and the path
Output : Updated Path Tree
Procedure:

```
[1] $CurrentNode \leftarrow$ Root of the tree
[2] for each page, $P$ in the path
[3]     $CurrentPage \leftarrow P$
[4]     $Childnode \leftarrow$ Child of the current node with label CurrentPage
[5]     if (*ChildNode* != NULL) /*If there exists such a child*/
[6]         Increase the count of *ChildNode*
[7]         $CurrentNode \leftarrow$ *ChildNode*
[8]     End If
[9]     else
[10]        $ChildNode \leftarrow$ Create a new child belonging to CurrentNode with
            label *CurrentPage*
[11]        Count of $ChildNode \leftarrow 1$
[12]        $CurrentNode \leftarrow$ *ChildNode*
[13]    End Else
[14]End For
```

Figure 3.9: The Algorithm for adding a given path into the PathTree

the *PathTree* is depicted in Figure 3.9.

In the *PathTree*, each node except from the root node contains a page and corresponding count. The root of the tree just combines the nodes below it. It does not represent any particular page in the site, because paths may start at any page in the site. In this tree, the node sequence obtained as a result of a walk from a root node to any of the leaves is one of the paths followed by the visitors of the web site in consideration. In addition, the count value associated with a particular node, N is the number of visitors reaching to the page stored in N from the root by retrieving the pages that are placed in the nodes which are in-between the root and N.

While adding a particular path into *PathTree*, *AddPath* algorithm either creates a new node or increments the count of the existing node for each page belonging to the given path. It starts from the root node whose children are checked until finding the one which contains the first page in the path. If such a node is found, its count is increased by one, after which the children of that node are traversed for finding the second page in the path. If there does not exist such a node, the algorithm creates a new node containing the first page and adds it as a new child to the root node. In general, in cases where the node, say n_i that is reached by matching the first *i-1* pages of a given path has no matching child containing the i^{th} page of the path, the algorithm creates a new child for node n_i containing the i^{th} page. In this case, all of the remaining pages starting from the i^{th} one are placed into the subtree rooted at n_i. On the other hand, if node n_i has a child containing the i^{th} page of the path, the algorithm just increases the count of n_i and continues searching for the rest of the pages of the path in the subtree rooted at n_i. The tree built this way contains all possible paths.

As an example, consider that at some point on the execution of the PathInfoProcessor, the *PathTree* obtained is like the one shown in the first figure in Figure 3.10. Assume that we want to add the path P=(K, B, T) into this tree. Figure 3.10 demonstrates the step by step addition of this path into the given *PathTree*. In this example, the nodes corresponding to the first two pages of the path could be located in the tree, so the algorithm just increases the counts of these node. On the other hand, the algorithm creates a new node for the last page in the path.

3.2.4 Storing PathTree

As indicated earlier, the tree created by PathInfoProcessor will be used by Recommender, on-line module of the system for finding recommendations matching to the needs of the visitors. For this reason, PathInfoProcessor should store the *PathTree* in such a way that it is understandable and accessible by Recommender. Here, it is very important that Recommender should spend minimal amount of time on reaching and using that data, because the speed is critical for the success of recommendation process. In the light of these observations, we firstly decided to design Recommender in such a way that it keeps the *PathTree* in the same way with the PathInfoProcessor. But, our experiments have shown that the creation of this tree is a costly operation.

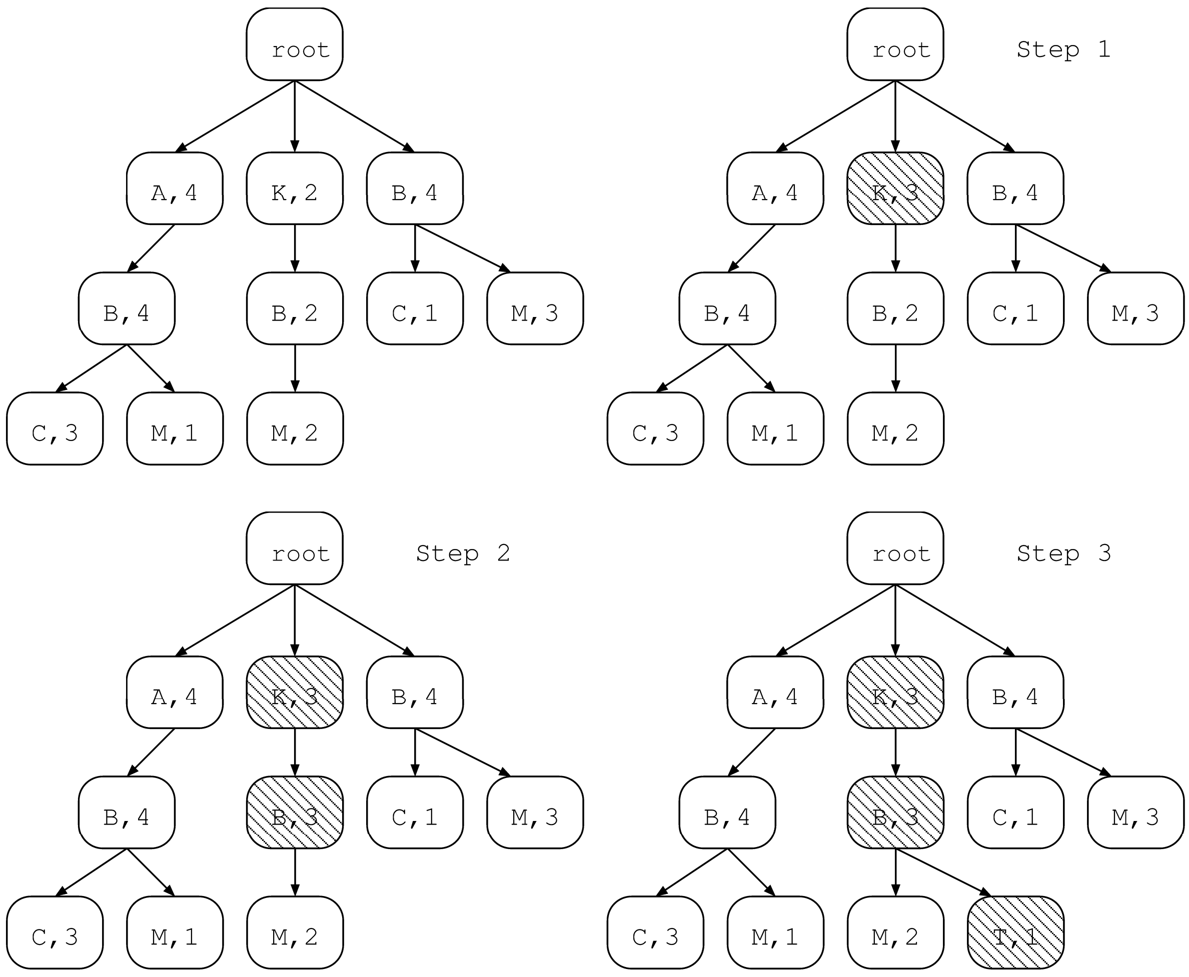


Figure 3.10: Step by step construction of PathTree

For this reason, we stored the tree in such a form that the Recommender will spent minimum amount of time on creating and accessing it. Our solution for this is to convert the *PathTree* into an one-dimensional form, namely store it in an array called *NewPathTree*. In this case, there is one to one correspondence between the entries of the array and nodes of the *PathTree*. Each node of the *NewPathTree* contains a page, count and the starting and ending indexes, *ChildStartIndex* and *ChildEndIndex* of the entries containing the children of the node represented in that entry. As a result, if the system needs to find a children of a particular node in *NewPathTree*, it finds them in the entries between *ChildStartIndex* and *ChildEndIndex* where these indexes are obtained from the entry which contains the node in consideration. As a result of these, during the process of converting the *PathTree* into *NewPathTree*, the system does not lose the parent-child relationships. During this operation, only the representation of the data changes. Figure 3.12 demonstrates the algorithm performing the conversion operation.

As an example, *NewPathTree* corresponding to the original *PathTree* shown in first figure of Figure 3.10 is shown in Figure 3.11.

0	root,-,1,3
1	A,4,4,4
2	K,2,5,5
3	B,4,6,7
4	B,2,10,10
5	C,1,0,0
6	M,3,0,0
7	C,3,0,0
8	M,1,0,0
9	M,2,0,0

A:	/~sesra
B:	/~sesra/courses.html
C:	/~sesra/courses/cs101
D:	/~sesra/courses/cs101/grades.html
E:	/~sesra/courses/cs102
E:	/~sesra/courses/cs102/lecturenotes.html

Figure 3.11: Converted tree

While converting the *PathTree* into a one-dimensional form, PathInfoProcessor performs pruning on it at the same time. Pruning is done by removing the redundant data to decrease the size of the *PathTree*. This operation is simple yet critical for the success of the recommendation process and really necessary in our case, because the system adds all paths found into the *PathTree* and there may be paths that are followed rarely. After the *PathTree* obtained completely, it is easy to detect these paths by just looking at the counts of the nodes. While forming the *NewPathTree*, if a particular node of the *PathTree* has a count smaller than

Input: PathTree and MinValue
Output: NewPathTree
Procedure:

```
[1] for each child, $c_i$ of the root node
[2]     if $c_i$ has a child whose count is larger than $MinValue$
[3]         $lastnode{+}{+}$
[4]         $NewPathTree[lastnode].URLname \leftarrow child_i.URLname$ and
            $NewPathTree[index].count \leftarrow child_i.count$
[5]         $NewPathTree[lastnode].ChildStartIndex \leftarrow$ -1
[6]         $NewPathTree[lastnode].childpointer \leftarrow child_i.ChildNode$
[7]     End If
[8] End For
[9] $NewPathTree[0].startindex \leftarrow 0$
[10] $NewPathTree[0].endindex \leftarrow lastnode$
[11] $processed \leftarrow 0$
[12] while ($processed < lastnode$)
[13]    $nodeinprocess{+}{+}$
[14]    if $NewPathTree[nodeinprocess].startindex > 1$ then $nodeinprocess{+}{+}$ End If
[15]    else
[16]        Get the child pointer of $NewPathTree[processed]$
[17]        $startindex \leftarrow lastnode$
[18]        for each child, $c_i$ of $NewPathTree[processed]$
[19]            if ($c_i.count > MinValue$)
[20]                $lastnode{+}{+}$
[21]                $NewPathTree[lastnode].URLname \leftarrow child_i.URLname$ and
                    $NewPathTree[index].count \leftarrow child_i.count$
[22]                $NewPathTree[lastnode].ChildStartIndex \leftarrow$ -1
[23]                $NewPathTree[lastnode].childpointer \leftarrow child_i.ChildNode$
[24]            End If
[25]        End For
[26]        if ($startindex == lastnode$)
[27]            $NewPathTree[nodeinprocess].startindex \leftarrow 0$
[28]        End If
[29]        else
[30]            $NewPathTree[nodeinprocess].ChildStartIndex \leftarrow startindex+1$
[31]            $NewPathTree[nodeinprocess].ChildEndIndex \leftarrow lastnode$
[32]        End Else
[33]    End Else
[34] End While
```

Figure 3.12: The Algorithm for changing the format of the PathTree

the minimum value needed, this node will not be added into the *NewPathTree* as a child to its old parent. Also, the subtree rooted at that node will not be represented in the *NewPathTree*. The minimum value needed to make the node relevant is called as *MinValue*. In Chapter 5, we will discuss on finding the correct value for this variable.

As a result, the program performs two different and important tasks together, changing the format of tree and pruning. Once the tree in the new format is obtained, it is easy to store it into a permanent storage to be utilized by Recommender. The entries in the *NewPathTree* are printed one by one in separate lines of the file called *PathInfoFile*. Each line of this file contains page address, count value and *ChildStartIndex* and *ChildEndIndex* values. Namely, the i^{th} line of the *PathFile* corresponds to the i^{th} entry of the *NewPathTree*.

3.3 HostIdentityProcessor

HostIdentityProcessor submodule aims to discover relations between identity information and navigation patterns of visitors. More specifically, the dependencies of the navigation patterns of the visitors on the identity information of them are discovered to be used by the Recommender, which will use this knowledge together with the path information of the visitors for guessing the next requests of the visitors. Before going into the details of this process, we will explain the way to obtain identity information of visitors in Section 3.3.1.

3.3.1 Obtaining Identity Information of the visitors

As indicated before, the first field of each entry existing in the log file is either the IP address or fully qualified domain name of the visitor causing that entry. In this section, we will explain the details related to these concepts.

Firstly, each computer connecting to the Internet has a unique *IP address*. IP (Internet Protocol) addresses are comprised of four numbers separated by periods.

Each of these four numbers must be between 0 and 255, e.g. 139.169.11.23. Naturally, IP addresses are incomprehensible for the humans who may not associate any meaning to the numbers forming them. Because of this reason, computers are also associated with a particular name which is called as the *fully qualified domain name*. By the help of fully qualified names, it becomes possible to learn more about the computers connecting to the Internet.

Fully qualified domain names should have at least two fields which are separated by periods. This time, fields are composed of sequence of characters where each character is either a digit between 0 and 9 or a letter between a and z. Here, the values of fields forming the domain names are not chosen randomly. Even more, each computer is associated with a domain name according to the predefined hierarchy built on the domain names which is called as *domain name hierarchy*.

Topmost level in the hierarchy contains the root domain. The nodes below the root contain the top-level domains which are listed below.

- com: Companies
- edu: Universities
- mil: Military Organizations
- gov: Government Organizations
- net: Internet Service Providers
- org: Nonprofit Organizations
- Country Names such as tr, de, uk, etc .

Each top-level domain in the domain name hierarchy has a second and lower level domains. All domains contain the domains below them in the hierarchy. Domain name hierarchy can also be represented by a hierarchical tree structure where each node of the tree represents a domain name. Except from the leaves, all nodes in the tree contain the names of general domains. So, machines are placed at the bottom of the tree. General domains which are placed on inner nodes contain more than one computer. In fact, all of the general domains contain the domains below them in the tree. As the general domain approaches to the root, the number of machines contained in it increases. Figure 3.13 contains an example domain name hierarchy. Note that the hierarchy shown in this figure is only the small subset of the real domain name hierarchy.

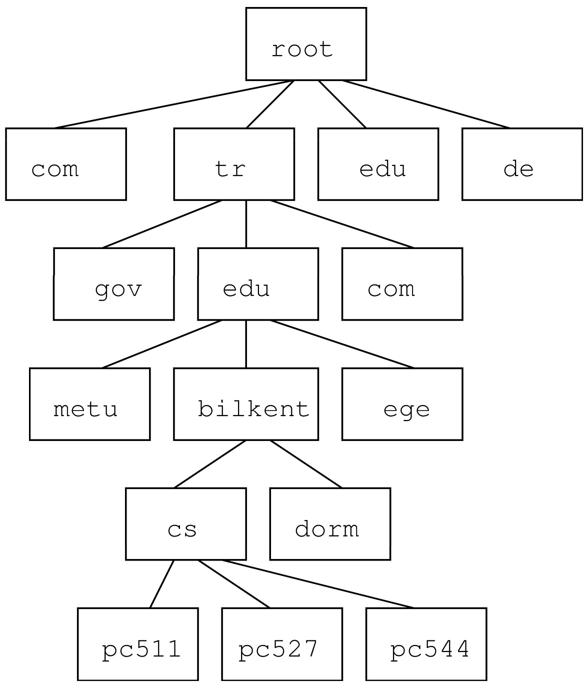


Figure 3.13: Example Domain Hierarchy

The fully qualified domain name for a particular host is obtained by concatenating the domain names of the nodes that are placed on the way from the root to the leaf node containing the host. As an example, consider the fully qualified domain name *pc511.cs.bilkent.edu.tr* which can be obtained by using the domain name hierarchy depicted in Figure 3.13. It is apparent from the figure that this domain name is obtained by concatenating the domain names from root domain to the leaf containing *pc511* subdomain. In the following lines, we explain what each field in given fully qualified domain name means.

tr: Turkey

edu.tr: Educational organizations in Turkey

bilkent.edu.tr: Bilkent University in Turkey

cs.bilkent.edu.tr: CS Department of Bilkent University

pc511.cs.bilkent.edu.tr: The machine named as pc511 on CS department of Bilkent University.

In this example, the topmost domain containing the corresponding machine is denoted with *tr* which is the rightmost field of the fully qualified domain name. As proceeding into lower levels in the domain hierarchy, the domain gets more specific. Of course, the number of machines on *Bilkent* domain is very much smaller then the number of machines in *tr* domain.

The names of the top-level domains are assigned by the corporation named as The Internet Corporation for Assigned Names and Numbers (ICANN). The domains below the top-level are managed by their administrators. Domains are generally divided into zones depending on the units which are either geographically or administratively differentiate from each other. The zones are generally suited to the natural organization of the institutions. For example, universities may divide their domain space into zones such that each department is assigned with a different zone. Domain name hierarchy is built by naming these zones.

3.3.2 Identity Information Usage

We believe that the identity information may present very useful clues in predicting the next requests of the visitors. For example, consider a visitor retrieving the homepage of a particular instructor of CS department of the Bilkent University. Only by looking at the identity of that visitor, we can make a prediction on the purpose of his visit. For example, if the visitor is from the *dorm* subdomain of *Bilkent* domain which contains the dormitories of the university, he/she may request a homepage of one of the undergraduate courses taught by the instructor, because dormitories are mostly occupied by the undergraduate students. As another example, consider a visitor retrieving the homepage of CS department of Bilkent University. If the visitor is from another university or a company, she/he may be interested in the research going on in the department.

Unfortunately, it is not possible to foresee whether our predictions are really valid in real life without analyzing the behavior of the visitors. Of course, it is always possible to make such predictions manually and to test whether they really work in real life or not. But this is not a feasible way to make use of identity information, because the predictions produced in this way are so subjective that may cost us missing hidden patterns between navigation patterns and identity information.

In our system, the discovery process of dependencies between identity information and navigation patterns is fully automated for obtaining more objective and complete results. In this case, the system only trusts on the usage behavior of previous visitors. Briefly, we foresee that if the recent visitors coming from a particular domain show some trends on their navigational behavior, the new visitors from the similar domains will also be inclined to show the same trends. In the light of this assumption, our solution procedure summarizes the common trends in navigational behavior of visitors belonging to each domain.

We think that the most valuable information about a particular visitor is obtained by analyzing directly the past behavior of him. So, our system keeps a personal record for every single visitor. On the other hand, the site may have new visitors about whom the system has no previous knowledge. Even in these cases, it is possible to learn about possible interests of these visitors. Especially for that kinds of visitors, we propose to make use of the domain name hierarchy concept explained in the previous section. Even for the old visitors, usage of this concept provides valuable information.

If a particular visitor has not visited the site recently, we will try to guess the behavior of the visitor by looking at the visitors from the similar domains. Similarity between domains is determined by looking at their position in the domain name hierarchy. The similarity between two domains increase as the number of common parents of them increase in the domain hierarchy tree. For example, two hosts are most similar if their immediate parents are same. On the other hand, if only the top-level domains containing the hosts are the same, the similarity between them is less. If the top-level domains containing them are not even same, the hosts have no similarity. The reasoning behind these assumptions is that the probability that professions, interests etc.. are same for two hosts increase as they become closer. For example, most of the hosts from *dorm.bilkent.edu.tr* subdomain belong to the students of Bilkent University, because dorms are occupied by students and they have similar interests in general, such as courses. Or, visitors from *edu.tr* domain are either students or instructors. In this case, the hosts are less similar. Even in this case, they may have common interests, compared to the visitor from *com* domains.

We will try to clarify these issues with an example. Assume that the page */courses.html* which contains the courses taught in the CS department of Bilkent University has two visitors, A and B whose fully qualified domain names and navigation patterns are shown below.

A: pc511.cs.bilkent.edu.tr and /courses.html → /cs521.html
B: pc522.cs.bilkent.edu.tr and /courses.html → /cs521.html

Suppose that after these visitors left the site, the page */courses.html* was requested by two new visitors, C and D. Additionally assume that, visitor C has visited the site recently and retrieved the pages */courses.html* and */cs412.html* consecutively while visitor D performed no visit to the site recently. In such a situation, HostIdentityProcessor produces the page */cs412.html* as a recommendation for visitor C by just looking at his past interests. On the other hand, as indicated visitor D whose fully qualified domain name is *pc513.cs.bilkent.edu.tr* has not visited the site recently, so the system does not have a specific information on him. Besides, the system can learn about the possible interests of this visitor by considering the parent domains of it. In this example, all of the tree visitors (A, B and D) are from *cs.bilkent.edu.tr* domain. Two visitors (A and B) from this general domain request page */cs521* after */courses.html* page. As a result, there is a probability that visitor D will also retrieve that page. The probability increases as the number of visitors from *cs.bilkent.edu.tr* domain following the same path increases, because this indicates a more common trend among visitors from that domain.

3.3.3 Processing Identity Information of the Hosts

In this section, we will explain the method utilized by the HostIdentityProcessor for mining the web logs with the aim of discovering relations between navigational patterns and identities of the hosts. For this, the first possible approach that we considered was to embed the identity information into the *PathTree* obtained by the PathInfoProcessor. But, we thought that this is not a feasible solution for this problem, because it will cause a decrease in the efficiency of the Recommender system by expanding the *PathTree* too much. Each link in the path tree should be associated with the list of domain names where the number of distinct domains can easily grow so high. Because of this reason, the processes of obtaining identity related information and path information are completely separated from each other in our system. According to our solution procedure, HostIdentityProcessor does not consider the paths followed by the visitors. It only searches for the pages that are retrieved by a particular visitor after a retrieval of a particular page.

The main data structure created for keeping the hosts and their navigational patterns is depicted in Figure 3.14. Almost all operations taking place in that module interact with this list ($HostInfoList$) which acts as a database indexed according to the identifiers (IP address or domain name) of the hosts or general domains. It is mainly a list of records where each record contains the identifier, boolean variable showing whether the identifier is fully qualified domain name or not and a pointer to the data structure containing the navigational patterns of the visitor or visitors represented in the current entry. These navigation patterns belong to a particular visitor if the identifier field contains an IP Address or fully qualified domain name of him. On the other hand, if the identifier field contains the name of a particular general domain such as *cs.bilkent.edu.tr* then the navigational patterns of all visitors belonging to that domain are kept in

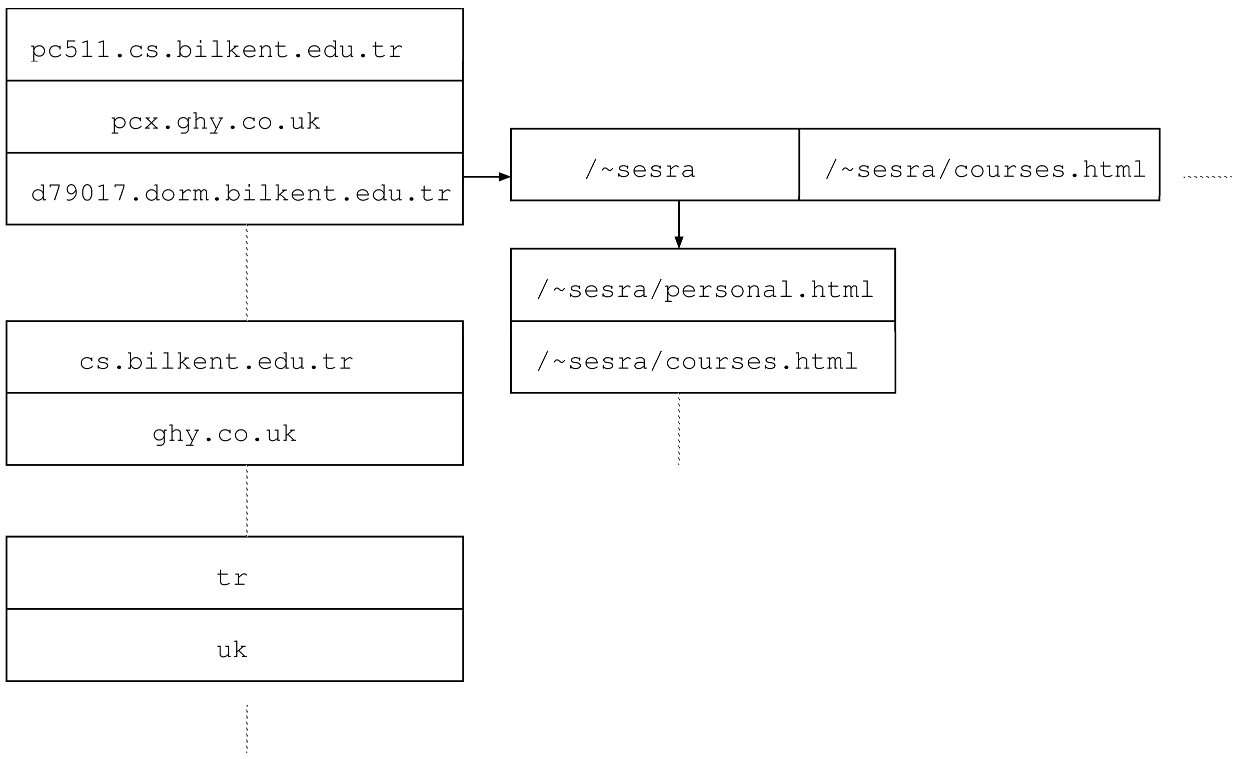


Figure 3.14: HostInfoList

this entry. Navigational patterns are kept in a list of records where each record contains a referrer page and a pointer to the list of pages that are retrieved immediately after this page. So, the system learns which pages are retrieved after a retrieval of a particular page by each host.

Main steps of the HostIdentityProcessor algorithm are shown in Figure 3.15. In the first step of the algorithm, data obtained in the previous days is loaded from the *HostInfoFile* into the *HostInfoList*. Then, the *HostInfoList* is updated with the data added into the log file since the last run of the program. In the third step of the algorithm, the data obtained as result of an execution of the first two steps is stored into permanent storage. Finally, last two steps are devoted to the task of handling general domains. In these steps, general domains are found and common navigational behavior of the visitors from these domains are discovered. Detailed explanation of each of these steps will be presented in the sequel.

Step 1: Load the past data produced in the previous days into $HostInfoList$

Step 2: Update $HostInfoList$ with the new entries from the active log file

Step 3: Print the data file corresponding to $HostInfoList$

Step 4: Determine all parent domains and add them into the $HostInfoList$

Step 5: Find the data related to general domains and save it to data file

Figure 3.15: Main steps of the work performed by HostIdentityProcessor

Step 2 of the HostIdentityProcessor algorithm is performed by the algorithm depicted in Figure 3.16. The logic behind the creation and termination of user sessions is same as the PathInfoProcessor algorithm, in which the sessions are removed from the session database 30 minutes after their creation. Before the removal, HostIdentityProcessor firstly calls the FindDomain function which is responsible for finding the index of the entry containing the given host in the $HostInfoList$. In this case, the FindDomain function tries to locate the host which owns the user session in process. FindDomain algorithm traverses the entries in the HostInfoList one by one until finding the index of the entry containing the given domain name or IP address. If it could not find such an entry, then it adds a new entry containing to the $HostInfoList$. In each case, it returns the index of the entry containing the host. After the index of the entry containing the given host is found, the data stored in that entry is updated by the addition of the navigational patterns hidden in the user session. For this, each access belonging to the user session is reflected into the $HostInfoList$ by calling the function AddAccess (Figure 3.17).

AddAccess function is responsible for embedding the given access which consists of (*SourcePage, TargetPage*) pair into the navigational pattern data stored in the given entry of the $HostInfoList$. The reason for this operation is that the host represented in that entry retrieves the page named *TargetPage* after retrieving the page named *SourcePage*. As indicated, the navigational patterns belonging to a particular host are kept in a list of lists which is pointed by the entry containing that host in $HostInfoList$. Each entry in the outer list named

Input : Web Log File
Output : HostInfoList
Procedure:

```
[1] For each new entry in the given log file
[2]   Read the next entry from the log file
[3]   Parse the log entry to obtain the *Identity*, *Referrer, Target and Time*
      for the request
[4]   if the request is generated by a new visitor
[5]      Create a new user Session
[6]      Owner of the user session $\leftarrow$ *Identity*
[7]      Starting time for User session $\leftarrow$ *Time*
[8]      First access of the user session $\leftarrow$ (*Referrer, Target*)
[9]   End If
[10]  Else
[11]     if the request is valid
[12]        Add a new access (Referrer, Target) to the session belonging to
            the current visitor
[13]     End If
[14]  End Else
[15]  for each user session, $S$ in the session database
[16]     if the session, $S$ is created 30 minutes before
[17]        Find the index of the connecting machine in the $HostInfoList$
            (if it does not exist in the $HostInfoList$, add a new entry for this machine)
[18]  i=0
[19]  while ($i \leq$ size of $HostInfoList$) and ($found$==0)
[20]     if ($HostInfoList[i].Identitiy == Identity$)
[21]        Determine if identity is domain name or ip address
[22]        for each access ($Referrer$, $Target$) in the user session, $S$
[23]           *AddAccess (Indentity, Referrer, Target)* /*to add the access
[24]        Remove the user session, $S$ from the active sessions database
[25]        End For
[26]     End If
[27]  End For
[28] End While
```

Figure 3.16: The Algorithm for processing the new entries of the log file

Input : SourcePage, TargetPage, IndexOfHost
Output : HostInfoList updated with the given access
Procedure:

```
[1] $TempSource \leftarrow (HostInfoList$[IndexOfHost] $\rightarrow SourcePages)$
[2] while ($TempSource$ is not null) and ($SourcePageFound$==0)
[3]    if ($TempSource.URLname$ == $Referrer$)
[4]       $SourcePageFound \leftarrow 1$
[5]       $TempTarget$ = (($TempSource \rightarrow TargetPages$)
[6]       while ($TempTarget$ is not null) and ($TargetPageFound$==0)
[7]          if ($TempTarget.URLname$ == $TargetPage$)
[8]             $TempTarget.count$++
[9]          End If
[10]      End While
[11]      if ($TargetPageFound$==0)
[12]         Add a new node containing the $TargetPage$ to the tail of
             ($TempSource \rightarrow TargetPages$) list
[13]      End If
[14]   End If
[15] End While
[16] if ($SourcePageFound$==0)
[17]   Add a new node containing $SouurcePage$ to the tail of $SourcePages$ list
[18]   ($TempSource \rightarrow TargetPages$) $\rightarrow$ new node containing page named $TargetPage$
[19]End If
```

Figure 3.17: AddAccess algorithm for adding a given access into the $HostInfoList$

SourcePages contains a particular page, P and a pointer to the list named *TargetPages* which contains the pages that are visited immediately after P. To be able to perform the addition of a new access, AddAccess function first searches the *SourcePages* list for finding the *SourcePage* in it. If it could not find such an entry which means that this page was not retrieved by this visitor recently, then *SourcePages* list is enlarged with the addition of the *SourcePage* to the tail of it. In such cases, *TargetPage* is added into the first node of the *TargetPages* list originating from the newly added node of *SourcePages* list. In cases where the *SourcePage* is located, the algorithm searches the *TargetPages* list originating from the node containing the source page. If a node containing the *TargetPage* is found, the algorithm just increments its count by one. Otherwise, which means that the host has not retrieved the page named *TargetPage* after the page named *SourcePage* recently, a new node containing *TargetPage* is added into the tail of the *TargetPages* list. Here, the process of adding the given access into the $HostInfoList$ terminates.

3.3.4 Storing the data on HostInfoList

The third step of the work performed by the HostIdentityProcessor is to store the data in $HostInfoList$ into the permanent storage to be used by the Recommender (Figure 3.18). For this, HostIdentityProcessor saves the contents of the $HostInfoList$ into a file called *HostInfoFile* by calling the function named StoreDomainInformation (Figure3.2.3). The StoreDomainInformation algorithm processes the each entry in $HostInfoList$ one by one. For each entry, it firstly saves the name of the host kept in this entry. Each of the consecutive lines start with a particular source page existing in a *SourcePages* list of the entry in process and continues with the target pages that take place in the *TargetPages* list originating from the node of that source page. At the end of this process, HostInfoFile contains the navigational behavior of all hosts encountered by the system. The data file created by this way is so huge that it would be a time consuming task for the Recommender module to locate the data related to a particular visitor.

To resolve this, HostIdentityProcessor creates an auxiliary file as an index to

```
Input : $HostInfoList$
Output : HostInfoFile and IndexFile
Procedure:
[1] for each entry, $e_i$ in the $HostInfoList$
[2]     $StartPosition \leftarrow$ Current position of the cursor in HostInfoFile
[3]     Print $HostInfoList[i].Identity$ into the HostInfoFile
[4]     for each source page, SP$\in$ ($HostInfoList[i]->SourcePages$) list
[5]         Print the name of the source page into next line
[6]         for each target page in, ($SP \in SourcePagee \rightarrow TargetPages$ list
[7]             Print the name of the target page and print empty space
                into the current line
[8]         End For
[9]     End For
[10] $EndPosition \leftarrow$ Current Position of the cursor in DataFile
[11] Print $HostInfoList[i].Identity$, $StartPosition$ and $EndPosition$
        into the IndexFile
[12] Pass to the next line in both files
[13] End For
```

Figure 3.18: Storing the contents of the $HostInfoList$ into HostInfoFile

the *HostInfoFile*. Each line of the index file called *IndexFile* contains an identifier (IP address or domain name) and starting position of the data belonging to the host having this identity. The index file created by this way is much smaller than $HostInfoFile$. Recommender can easily find the position of the data related to a particular host in *HostInfoFile* by searching the index file. So, the system does not spend any time on searching real data file called *HostInfoFile* which is much larger than the index file. So, the huge size of the data stored does not decrease the performance of the on-line part. Figure 3.19 demonstrates the formats of the *IndexFile* and *HostInfoFile* created for the example $HostInfoList$ shown in Figure 3.14.

3.3.5 Handling General Domains

As indicated before, the system should have some information about the common behavioral trends of the visitors belonging to the general domains such as

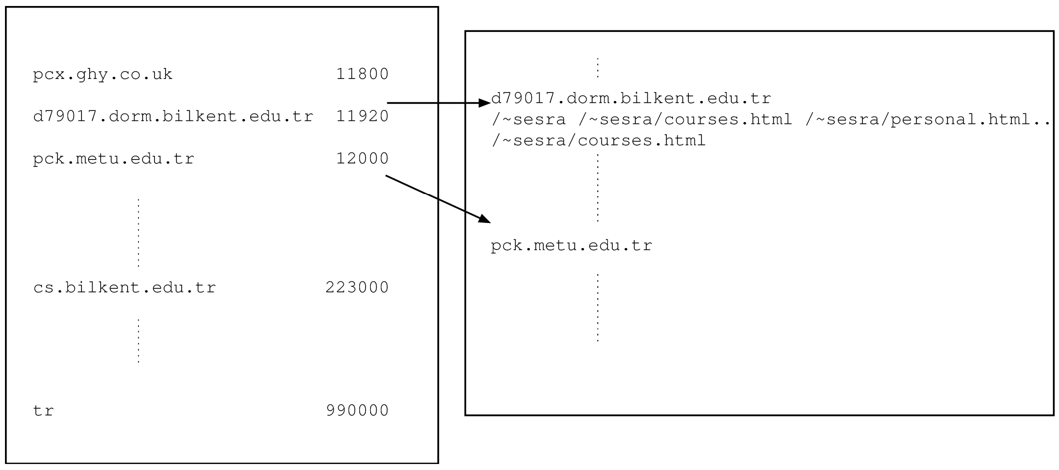

pcx.ghy.co.uk 11800
d79017.dorm.bilkent.edu.tr 11920
pck.metu.edu.tr 12000
cs.bilkent.edu.tr 223000
tr 990000
d79017.dorm.bilkent.edu.tr
/~sesra /~sesra/courses.html /~sesra/personal.html..
/~sesra/courses.html
pck.metu.edu.tr

Figure 3.19: IndexFile and DataFile created by HostInfoProcessor

cs.bilkent.edu.tr, us, ibm.com, etc. In fact, it is possible to extract this information on-line by combining the data belonging to the hosts that take place in the target general domain. But, this is a time consuming task and the speed is critical for on-line process. So, this operation should be performed off-line by HostIdentityProcessor.

HostIdentityProcessor firstly finds all names of general domains by analyzing the fully qualified domain names existing in the $HostInfoList$ in the 4^{th} step of its work. For this, it calls AddParents function which traverses the $HostInfoList$ for once for finding the entries that contain a fully qualified domain name. For each entry found, it generates all parent domains of the fully qualified domain existing in that entry and adds them into the $HostInfoList$ if they were not already added. For example, as a results of a processing of a fully qualified domain name *pc511.cs.bilkent.edu.tr*, the following general domain names are already to $HostInfoList$: *cs.bilkent.edu.tr, bilkent.edu.tr, edu.tr* and *tr*.

The next step is to discover and store the common navigational behavior of the visitors of each general domain that take place in the $HostInfoList$ by calling ProcessGeneralDomains function (Figure 3.20). This operation is time consuming, because for each general domain, D all of the entries of the $HostInfoList$ until the starting point of the general domains are traversed for finding the ones that contain hosts which belong to D in domain name hierarchy. Whenever such a host is found, the data belonging to it is added into the data stored in entry containing the general domain. For example, the following list contains some of the hosts in *tr* general domain: *pc5611.cs.bilkent.edu.tr, u55.ege.edu.tr*, etc. In fact, the number of machines is naturally much and much more than this. Whenever ProcessGeneralDomains function is processing *tr* domain, it must merge the navigational behavior of all such machines from *tr* domain that take place in the $HostInfoList$.

The data related to the general domains is stored into *HostInfoFile* and *IndexFile* immediately after the addition of the data related to all machines that take place on them. Then, the data belonging to the general domain is completely removed from the $HostInfoList$. The reason for this is to reduce the memory consumption. For example, if we kept the data belonging to all general domains such as *tr, com, edu.tr*, etc. in memory at the same time, the amount of memory consumption would be too much compared to the case where the data related to only one general domain is in memory at any moment.

Input : *HostInfoList*
Output : Navigational Patterns belonging to the general domains are printed into HostInfoFile and IndexFile
Procedure:

```
[1] for each general domain, $G$ in the $HostInfoList$
[2]     for each host, $H$ in $HostInfoList$
[3]         if ($H$ is a subdomain of $G$
[4]             for each source page, $S$ recorded in (H $\rightarrow$ $SourcePages$) list
[5]                 for each target page, $T$ originating from $S$
[6]                 Add the access (S, P) into the entry owned by $G$ in $HostInfoList$
[7]                 End For
[8]             End For
[9]         End If
[10]    End For
[11]    Print the data related to $G$ into $HostInfoFile$
[12]    Print the starting position of the data related to $G$ in $HostInfoFile$
        into the $IndexFile$ together with the name of the $G$
[13]    Remove the navigation pattern data belonging to $G$ from HostInfoFile
[14] End For
```

Figure 3.20: Processing general domains

Chapter 4

Recommender

In this chapter, we will present the on-line part of the system we designed and implemented. Main goal of the Recommender is to recommend some links to the on-line visitors. This module is capable of guessing the web pages that may be interesting to and therefore requested by a particular visitor by discovering the behavior of the past visitors showing similar navigational behavior and/or having similar domain or IP information with him/her. The success of the system depends on the level at which the recommended links are useful for the visitors.

The Recommender system has two separate components, one working in the client side and the other in the server side. The component in the client side is a simple Java applet which should be added to each page for making the recommendation facility available. It is possible to add Java applets into html codes of the web pages by having a call to them in the html codes of the pages. For example, Figure 4.1 shows the html code that is calling the applet that we implemented for showing recommendations. In such situations, the applet is downloaded from the server and run in the client browser together with the other files including text and graphics forming the page. Java applets running in the client browsers are capable of making a network connection to the server side to exchange data. The first main responsibility of the Java applet is to start recommendation process by recognizing the visitor via finding the IP Address and if available fully qualified domain name of the client computer and activating the

```
<applet code=recommend  width="500"  height="500">

     <param name="MaxNumOfRecommendations"    value="7">
     <param name="BackgroungColor"  value="250,150,100">
     <param name="FirstLine"    value="Recommendations">

</applet>
```

Figure 4.1: Example html page calling an applet

component in the server side which will produce the recommendations. Secondly, it ends the recommendation process by showing the recommendations provided by the server side in the location of the applet.

The component in the server side is a *cgi* program in which the real process of finding recommendations takes place. Java applets are capable of communicating with cgi scripts placed on the server side. To be able to do this, the applet firstly opens a connection through the cgi script. Via this connection, data exchange between the applet and cgi program becomes possible. In our case, the applet sends the IP address and fully qualified domain name of the visitor to the cgi program after opening a connection with it. The task of cgi program is to find recommendations and send them back to the java applet via same the connection. In the following section, we will explain the work performed in the server side in more detail.

4.1 Recommendation Producing

The first task performed by the cgi program placed on the server side is to find the pages retrieved by the current visitor recently. This is performed by initiating a search on the active access log file of the web site. As indicated before, web servers add a new entry to the access log file for each page request they get in such a way that the first field of each entry contains either the IP address or fully qualified domain name of the visitor making that request. So, it becomes possible to determine the accesses coming from a particular host if the domain name and IP address identifying it is known. In our case, the identification of the visitor is provided by the applet running in the client side. The only thing that should be done for finding access of the visitor is to search the access log file for finding the requests made by him/her. Because of the huge sizes of the access log files, it is more appropriate to start searching from the end of the log file. The reason for this is that the system only looks for the recent accesses and new entries are added into the end of the file as they arrive.

For speeding up the process of finding accesses, the cgi program reads the data from the log file in chunks instead of reading one line at a time. Then, it divides the data read into lines and starting from the last line obtained from the current chunk, it checks whether the access in that line was performed by the given host or not. The accesses containing the valid page requests and performed by the given visitor form the user session and will be used by the cgi program for finding recommendations.

At this point, we should explain what we mean by the concept *recent*. The first criterion for accepting a particular page request as recent is that it should be performed in last thirty minutes. Secondly, we make use of the HistoryDepth criteria for determining if the request is recent or not as indicated before. We accept that only the last n pages retrieved by a particular visitor is effecting the pages that will be retrieved next where n is the history depth. The pages retrieved before that are accepted as irrelevant to the current access. In Chapter 5, we will explain the results obtained by changing the HistoryDepth criteria in off-line and on-line parts of the system.

At this point, the system knows about the each on-line visitor is the identity of the visitor and the path followed by him/her throughout his visit to the site. A path is a ordered sequence of pages retrieved by a particular visitor. First page in the path is the oldest page requested. Naturally, the length of the path can not exceed n, so if the length of the user path is n at any moment, retrieval of a new page will result in excluding the first page in it.

In the following three sections, we will explain how the recommendations are found according to the path and the domain information and then how they are merged to form the final recommendation set.

4.2 Finding recommendations according to the path

The first set of pages that contain candidate recommendations are found by considering the path followed by the visitor. The goal is to match the path followed by the visitor with the ones followed by the previous visitors as much as possible. The idea behind this approach is hidden in the statement that if considerable amount of previous visitors following a particular path P in the web site retrieves the same set of pages S in the follow-up, the new visitors of the site following the same path may also have an interest with the pages in S. Because following the same path indicates an interest on similar topics. Simply put, the heuristic we use here is that visitors from the same domain will have similar interests.

Some of the available Web Usage Mining systems do not consider the order in which the pages are requested by the visitors. Instead, they accept user session as an unordered set of pages and try to match the current user session with the past sessions by finding the number of common page requests in them. On the contrary, we think that the order in which the pages are requested provides more specific and personalized information about the visitors. On the other hand, keeping the order information requires much more processing in both on-line and off-line part of the system. Because, if we consider the paths, the size of the data that should be stored increases dramatically.

In the light of these observations, the algorithm which is used for searching the *PathTree* to find out whether the given path exists in it or not is named as PathRecommender. If the path exists, PathRecommender then finds the pages that are following this path in the *PathTree*.

PathRecommender algorithm firstly reads the file containing the *PathTree* created by PathInfoProcessor and creates the same data structure for storing the paths. The data structure created is a list of records where each record contains a page, count and the indexes of the entries containing the first and last child of the node.

While searching the *PathTree* for a particular path, the algorithm firstly checks the children of the root for finding the first page of the path. The first entry of the PathTree contains the root node and starting end ending indexes of the children of the root. So, the nodes between these indexes are the ones checked for the first node in the path.

If such a node is found, the algorithm searches the subtree rooted at that node for the rest of the path. Namely, the children of the node found in the $i-1^{th}$ iteration are traversed for finding the i^{th} page of the path at any iteration i. At any iteration, the algorithm terminates if no matching node is found. This means that given path is not followed by enough number of visitors which results in nonexistence of it in the *PathTree*. As a result, no page recommendation can be produced for that path if the algorithm terminates in this way.

The location of the children of a particular node is found by using the *ChildStartIndex* and *ChildEndIndex* of it. For any node, the entries between the *ChildStartIndex* and *ChildEndIndex* contain the children belonging to that node.

If the search for each page succeeds on all iterations then it means that the path is completely located in the tree. In this case, the path obtained as a result of a walk from the root of the tree to the last node corresponding to the last page in the path is equal to the given path. Henceforth, all nodes belonging to the subtree rooted at the last node represent the pages that are requested by the past visitors following the given path. So, they form the set of recommendations obtained for that path.

It should be noted that each page, P_x obtained by this algorithm is assigned a weight value as an indicator of the significance of the statement that the visitors following the path P retrieve the page P_x in the followup. The weights are calculated by finding the ratio of the number of visitors retrieving P_x after following P to the total number of visitors following the path P.

As an example, consider the *PathTree* shown in Figure 4.2.a. Assume that we want to form a recommendation set for a particular visitor following the path (A, B). According to the algorithm described above, the recommendation set contains

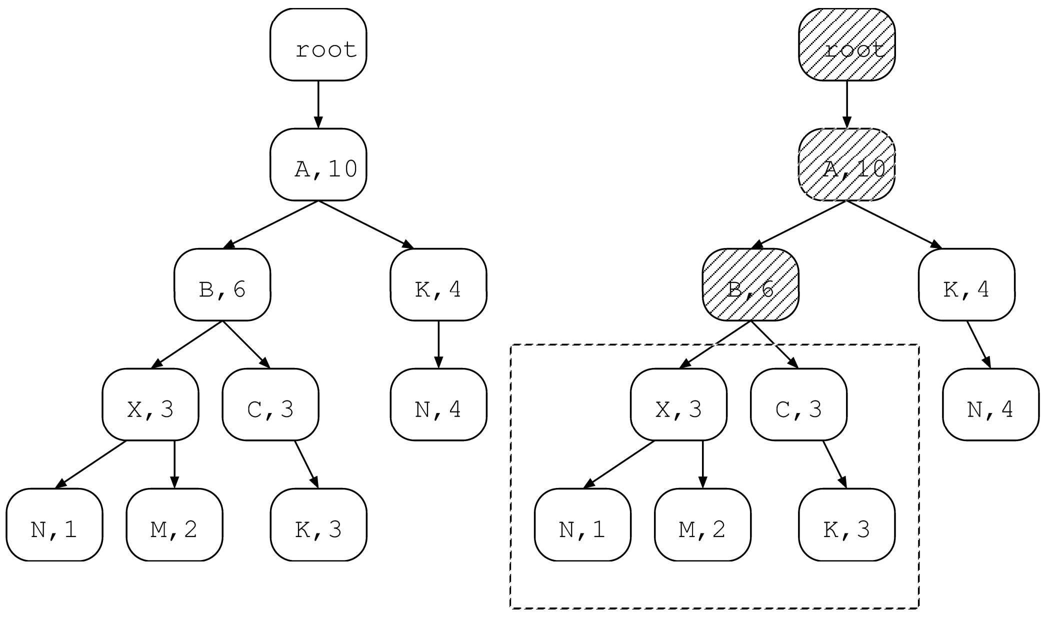


Figure 4.2: Finding recommendations according to the path information

the pages that are taken into a window in Figure 4.2.b. In this example, the score of the page N as a recommendation is 1/6, because only one visitor following the given path retrieves this page in the followup.

In addition to those, PathRecommender algorithm is firstly called to find out whether the exact path followed by the visitor exists in the *PathTree* or not. But, it may not always be possible to find the exact matching for the current user path. Especially in these cases, PathRecommender algorithm should be called to find the matching for all postfixes of the complete path. For example, the postfixes of the path (A, B, C, D) are (B, C, D), (C, D) and (D). Even in cases where the complete path is located in the $PathTree$, paths obtained by getting the postfixes of the real path are used to find recommendations. On the other hand, as the length of postfix path decreases, the strength of the obtained recommendations decrease. This is because, the similarity between the interests of the visitors becomes less.

4.3 Finding recommendations according to the identity information

Second set of recommendations is produced by looking at the identity information of the visitors. In producing recommendations according to the path information, the aim was to realize the interests of the visitor by looking at the navigational behavior him/her. The identity of the visitor has no importance in either off-line or on-line part of this process. On the other hand, the past behavior of the visitor himself/herself or similar visitors can be a strong indicator of the future behavior of him/her. With this reasoning, our system keeps a personal record for each of the different hosts visiting the web site. In addition, common behavioral patterns of the visitors existing in parent domains of the existing domains are also stored. By this way, it becomes possible to produce more personalized recommendations.

The algorithm for producing recommendations according to the identity information of the visitor is named as PersonalRecommender. Simply, the aim of the PersonalRecommender is to calculate the set of pages that may be requested by a particular visitor according to the last page request and fully qualified domain name or IP address of him/her. The algorithm reveals a set which consists of (*page, score*) pairs as an output. Each pair contains a page and associated score which shows how valuable this page as a recommendation.

Assume that PersonalRecommender is searching for the pages that may be retrieved after a particular page, P by the visitor having identity V. The algorithm firstly opens the *IndexFile* created by HostIdentityProcessor for finding the starting position of the data related to the given host in real data file, *HostInfoFile*. As indicated, *IndexFile* contains a huge number of lines, each containing IP address, fully qualified domain name or general domain name and the starting position of the related data in the data file. Until finding the line containing V, PersonalRecommender processes each line one by one. It compares the identifier placed on each line with V. If the host with identity V had performed a recent Visit to the site, it should have a record in *IndexFile* and *HostInfoFile* and the location of the data related to it is found during the search performed on *IndexFile*.

The ending position of the data related to the given host is simply determined by taking the starting position placed in the next line of the $IndexFile$. By this way, the system learns the exact position of the data belonging to the given host in $HostInfoFile$. Then, the algorithm directly reads the data between these positions as a chunk and divides it into lines. Each of the resulting lines, except from the first one contains a source page and the pages which are retrieved after the retrieval of the source page. The next task is to find the line containing P as a source page. If such a line is found, all other pages in that line are candidate recommendations.

As in PathInfoProcessor, each candidate page obtained by this algorithm is assigned a weight for showing the strength of the recommendation. For each page, the weight is calculated by finding the ratio of the number of times that this page is retrieved to the number of times the source page is retrieved.

PersonalRecommender algorithm initially tries to build a recommendation set by looking at the past behavior of the visitor himself/herself. On the other hand, if this visitor has not visited the web site or has not retrieved the page P recently, then it is not possible to produce recommendations for him/her by looking at his/her past behavior. In this case, PersonalRecommender makes use of the domain name hierarchy if the host has a domain name available instead of an IP address. In such situations, recommendation set is composed of recommendations resulting from the processing of parent domains of the host. This is achieved by just applying the same procedure explained above, this time considering the parent domain.

As explained in the previous sections, each domain has a number of parent domains which contain hosts with similar characteristics with it according to the domain name hierarchy. But, as the parent approaches to the root node of the domain hierarchy tree, this similarity decreases. In the light of these observations, the weight of the recommendation is decreased as the domain used for producing it gets more general. Naturally, the recommendations obtained by using the fully qualified domain name of the visitors are the ones which are most valuable, because they are produced only by considering the current visitor.

4.4 Production of the Final Set Of Recommendations

By making use of the path information and identification of the visitors, Recommender system mostly produces too many pages as candidates to be recommended to the visitor. But, it would not be very helpful and effective for the visitor if all of these recommendations are directly presented to him/her, because in such a case he/she has to spend a lot of time and energy on searching through a long list of recommendations for choosing the best ones. In fact what is convenient is to provide the system with a good filtering mechanism capable of choosing the pages that are better then the others.

In the light of these observations, both PersonalRecommender and PathRecommender assign a score to each page they found as a recommendation. Scores are assigned by considering a number of criterions. Throughout the previous sections, we talked about these criterions occasionally. To summarize, there are two criterions effecting the score of the recommendations obtained by PathRecommender. These criterions are listed below.

- Length of the path: As the length of the path increases, recommendation gains more score.
- Ratio of the number of people that retrieve this page to the number of people following the given path: As this value increases, recommendation gets more valuable.

The criterions that are effective on the score of the recommendations produced by PersonalRecommender are as follows.

- Location of the domain name in the domain name hierarchy: As the domain causing the recommendation approaches to the root, the score of the recommendation increases.

- Ratio of the number of times that this page is retrieved after the retrieval of the given page by the visitor himself/herself or the visitors belonging to the given domain: As this value increases, recommendation gets more valuable.

As a result, score of a particular page, P as a recommendation according to the path and identity information is as defined in definition.

Definition 3 a) Score of a particular page, P produced by PathRecommender is named as PathScore, PS for this page and $PS = (N2/N1)^{*}\ L$ where

- L = Length of the path considered
- $N1$= Number of people following the given path and
- $N2$= Number of people that retrieve this page afterwards

b) Score of a particular page, P produced by HostInfoRecommender is named as $HostInfoScore, HIS$ for this page and $PS = (N2/N1) * D$ where

- D = Number of dots('.') in the domain name
- $N1$= Number of times given source page retrieved
- $N2$= Number of times page P is retrieved afterwards

In Definition 3, HostInfoRecommender calculates the number of dots in the domain name. By this way, we try to understand how general the domain name is. As the number dots decreases in it, the domain name gets more general. For example, *edu.tr* has less dots then *pc511.cs.bilkent.edu.tr* and it is more general. So, recommendations produced by considering *edu.tr* are less valuable because of the reasons explained before. In fact, this is not an exact measure but beneficial in terms of reflecting the domain name hierarchy into the recommendation process.

By taking these criterions into consideration, HostInfoRecommender and PathRecommender produce two separate set of pages with associated scores. The next step is to form a single set out of these two sets. For this, firstly combined scores of the pages are calculated by adding up the *HostIdentScore* and *PathInfoScore* calculated for them. Naturally, if a particular page is produced as a recommendation in only one set, its score on the other set is 0. After these scores are calculated, they are put ordered. Topmost *MaxNumOfRecoms* pages are the recommendations that will be shown to the visitor. The designers of the page that will contain the applet can decide on the maximum number of recommendations that will be shown to the visitor.

After the recommendation set is finalized, the list of links is sent to the client browser via the connection opened before. Here, they are shown to the visitor with their scores as being attached to the current page. Whenever a user retrieves a new page, the applet showing them stops and the applet in new coming page opens a new connection with the server side and shows the new recommendations to the visitor.

Chapter 5

Evaluation

In this chapter, we will demonstrate the results of the experiments that we conducted for testing the capabilities of the usage mining system that we implemented. We performed our experiments on the web site of the CS department of Bilkent University. This web site contains the home pages of instructors and graduate students along with the home pages of the courses taught in the department. Besides, it contains some pages introducing the department. The CS department web site provides a good medium for conducting our experiments, because it is well-organized and popular enough to see the performance of the system in real life.

The organization of this chapter is as follows: Firstly, we will present the results of the experiments conducted for measuring the time and space requirements of the system. Then, we will give examples for the recommendation sets produced for different visitors and discuss about the results.

5.1 Time and Space Requirements of the system

In this section, time and space requirements of the off-line module of the UsageMiner system will be presented. As indicated before, we programmed off-line

Day	Log Size	Step 1	Step 2	Step 3	Step 4	Path File Size
1	7445258	0	5	0.1	0.3	347440
2	6634418	0.3	8	0.2	0.4	579733
3	6659211	0.4	9	0.2	0.3	800701
4	6369272	0.3	8	0.3	0.6	1061362
5	5848219	0.4	6	0.5	0.5	1221223
6	4768460	0.8	6	0.7	0.6	1387598
7	6989044	0.6	6	0.5	0.5	1701529

Table 5.1: Test Results for PathInfoProcessor Algorithm for one weak. Time values are in seconds while the sizes are in bytes.

module in such a way that it runs automatically at a predetermined time (when the load is the minumum) on each day. At each run, it processes the new entries arising in the log file. These entries are the ones that are added into the access log file until the last run of the program. Besides, this module is composed of two submodules, PathInfoProcessor and HostIdentityProcessor both operating on the same log file independent from each other. We will firstly present the test results for PathInfoProcessor.

Table 5.1 shows the test results for the execution of the PathInfoProcessor on one weak period, between the dates 01/05/2001 and 08/05/2001. In this table, the first column contains the number of the day. Namely, i^{th} row contains the test results for the i^{th} day. The second column is reserved for keeping the size of the log data processed on each day. In other words, the number in the second column of the i^{th} row contains the size of the data added into the log file in the i^{th} day. As shown in Figure 3.7, the PathInfoProcessor algorithm has four main steps. Next four columns in the table are showing the execution times of each of these steps for each run of the program. Finally, the numbers in the last column are the sizes of the PathInfoFiles obtained as a result of each run of the program. As indicated, at any day the Recommender utilizes the PathInfoFile produced by PathInfoProcessor in the previous day.

The results indicate that the most significant time consumption occurs during the execution of the 2^{nd} step of the PathInfoProcessor algorithm. The reader is reminded that in this step new entries in the log file are processed for forming

Log Size	H. Depth	MinValue	Step 1	Step 2	Step 3	Step 4	PathInfoFile
230127517	8	1	0	664	2	2	7400715
230127517	5	3	0	634	0.9	0.8	1381923
230127517	7	5	0	634	0.8	0.6	977672

Table 5.2: Performance of the PathInfoProcessor on a large log file. Time values are in seconds while the sizes are in bytes.

user sessions and finding the paths followed by the visitors owning them. Then, these paths are added into the PathTree. Excessive number of I/0 operations performed in this step is the reason for the large time consumption.

To see the effect of $MinValue$ and $HistoryDepth$ variables on the results obtained by PathInfoProcessor, we conducted another set of experiments on which we utilised a larger log file. This log file contains the log data created between the dates 01/05/2001 and 06/06/2001. This is a long time period, so the size of the log file is quite huge. Table 5.2 demontrates the results of these experiments. Each row displays a test results for different experiment. For each experiment, we set different values for $MinValue$ and $HistoryDepth$ variables while keeping the log file same.

Test results demonstrated in Table 5.2 show that decreasing the $HistoryDepth$ variable causes a decrease in the size of the PathInfoFile obtained, because $HistoryDepth$ determines the sizes of the paths that are added into the PathTree. If it increases, larger paths are produced and added into the PathTree. So, the depth of the PathTree increases as $HistoryDepth$ gets larger. In this respect, the size of the data that should be stored in PathInfoFile is also larger.

Furthermore, the size of the PathInfoFile decreases as the value of the variable, $MinValue$ increases. Becuase more nodes are eliminated during the pruning phase of the algorithm if the $MinValue$ gets larger. As a result, PathTree stored into the PathInfoFile gets smaller.

As indicated above, the speed is critical for the success of Recommender algorithm, because the visitors of a web site are generally satisfied with quick answers. The size of the PathInfoFile is very effective on the speed of the on-line module.

Day	Log Size	Step 1	Step 2	Step 3	Step 4	Step 5	Domain	Index
1	7445258	0	10	0.5	1	1	476427	41756
2	6634418	0.3	4	0.2	1	1	667936	60778
3	6659211	0.4	5	1	1	1	881837	82498
4	6369272	0.3	5	1	1	1	1068382	96565
5	5848219	0.4	5	0.5	1	1	1165213	108060
6	4768460	0.3	5	0.7	1	2	1322892	119357
7	6989044	0.6	8	0.6	2	2	1588609	140561

Table 5.3: Test Results for HostIdentProcessor Algorithm for one weak

This is due to the fact that the data stored in this file is read completely into the memory for each web page loaded by the visitors. So, keeping the $HistoryDepth$ variable smaller and $MinValue$ variable larger could be good choice. On the other hand, if we choose very large or small values the quality of the recommendations may decrease. So, we have decided to increase $MinValue$ as the log file processed gets larger. In other words, our system automatically sets larger values for $MinValue$ as the size of the log file increases.

We have also conducted experiments for measuring the time and space requirements of the HostIdentityProcessor algorithm. Table 5.3 demonstrates the results of the experiments. As in PathInfoProcessor, we run the HostInfoProcessor algorithm on one weak data. As it is in Table 5.1, the first column of the table depicted in Table 5.3 is reserved for the day numbers while the second column is reserved for keeping the size of the log data processed on each day. Different from PathInfoProcessor, the work performed by HostIdentityProcessor algorithm is divided into five steps. So, the next five columns in the table are showing the execution times of each of these steps for each run of the program. Finally, the numbers in the last two columns are the sizes of the HostInfoFile and IndexFile obtained as an output of the corresponding run.

The resuts show that the largest time consumption is occured during the execution of the Step 2 of HostIdentityProcessor algorithm. As indicated before, the reason for this is the number of I/O operations performed in this step. Additionally, there is a significant time consumption in last two steps of the algorithm. In these steps, general domains are formed and the data related to them is discovered.

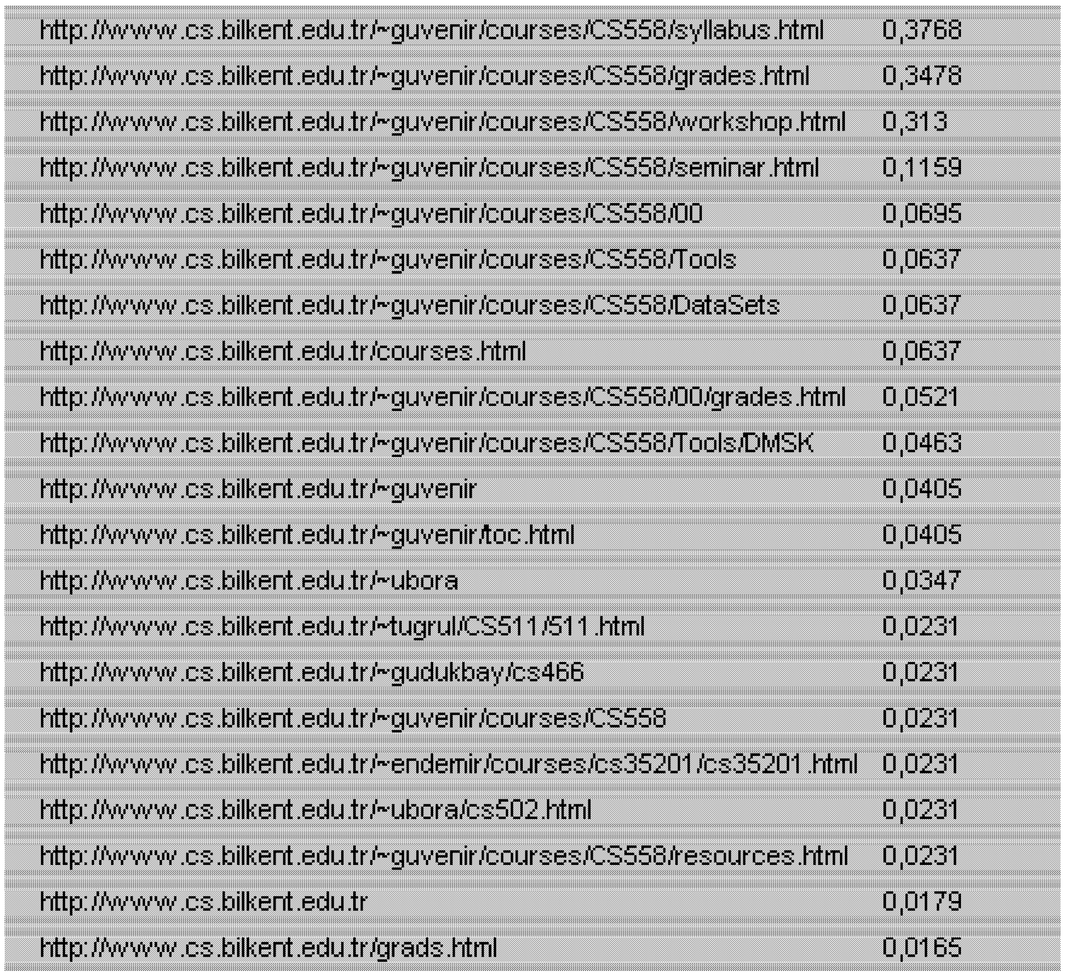
The results demonstrated in Table 5.3 indicate that the size of the IndexFile produced is much smaller than the HostInfoFile as we planned. The huge size of the HostInfoFile produced by the system does not cause a decrease in the performance of the Recommender because the Recommender does not get this file into the memory as a whole. As we explained, it only reads the IndexFile for determining the location of the target data in the HostInfoFile. So, keeping the size of the IndexFile small is enough for guaranteing the high performance in recommendation process.

5.2 Sample Outputs

In this section, we will give examples for the outputs produced by the on-line part of the system, Recommender. In fact, we will present the pages which were created in two seperate cases to be able to give an idea about the presentation of the recommendations to the visitor.

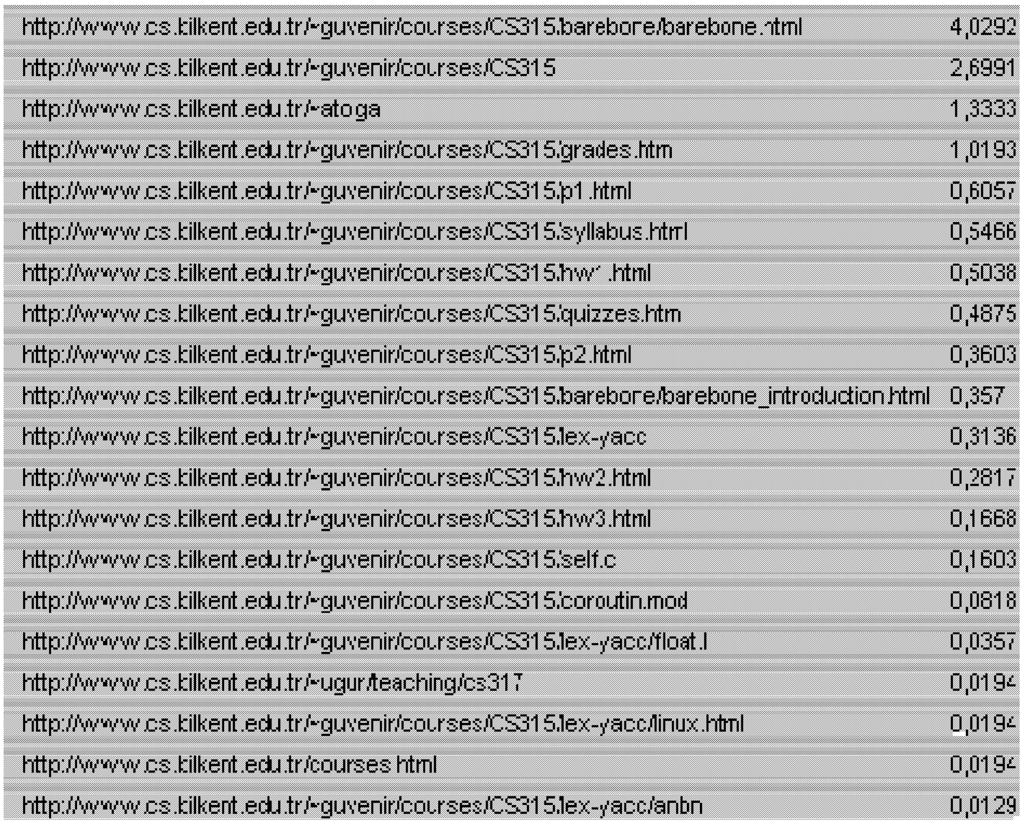
In the first example, fully qualified domain name of the visitor in consideration is *d71020.dorm.bilkent.edu.tr*. In addition the path followed by him/her contains the following nodes: (*/courses.html*, */~guvenir/courses/cs558/*). The output of the program for this visitor is shown in Figure 5.1.

In this example, some of the recommended pages are linked to the lats page retrieved by the visitor. One of the interesting recommendations is the one containing the page */~tugrul/cs511/cs511.html*. Here, both the last page retrieved last by the visitor and this page are the home pages of the graduate courses in the department. So, the system discovers that if a particular visitor is interested in an home page of particular graduate course, he/she may have an interest on other course pages too.



http://www.cs.bilkent.edu.tr/~guvenir/courses/CS558/syllabus.html	0,3768
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS558/grades.html	0,3478
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS558/workshop.html	0,313
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS558/seminar.html	0,1159
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS558/00	0,0695
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS558/Tools	0,0637
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS558/DataSets	0,0637
http://www.cs.bilkent.edu.tr/courses.html	0,0637
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS558/00/grades.html	0,0521
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS558/Tools/DMSK	0,0463
http://www.cs.bilkent.edu.tr/~guvenir	0,0405
http://www.cs.bilkent.edu.tr/~guvenir/toc.html	0,0405
http://www.cs.bilkent.edu.tr/~ubora	0,0347
http://www.cs.bilkent.edu.tr/~tugrul/CS511/511.html	0,0231
http://www.cs.bilkent.edu.tr/~gudukbay/cs466	0,0231
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS558	0,0231
http://www.cs.bilkent.edu.tr/~endemir/courses/cs35201/cs35201.html	0,0231
http://www.cs.bilkent.edu.tr/~ubora/cs502.html	0,0231
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS558/resources.html	0,0231
http://www.cs.bilkent.edu.tr	0,0179
http://www.cs.bilkent.edu.tr/grads.html	0,0165

Figure 5.1: Recommendation set for the first visitor



http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/barebore/barebone.html	4,0292
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315	2,6991
http://www.cs.bilkent.edu.tr/~atoga	1,3333
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/grades.htm	1,0193
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/p1.html	0,6057
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/syllabus.html	0,5466
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/hw1.html	0,5038
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/quizzes.htm	0,4875
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/p2.html	0,3603
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/barebore/barebone_introduction.html	0,357
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/lex-yacc	0,3136
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/hw2.html	0,2817
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/hw3.html	0,1668
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/self.c	0,1603
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/coroutin.mod	0,0818
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/lex-yacc/float.l	0,0357
http://www.cs.bilkent.edu.tr/~ugur/teaching/cs317	0,0194
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/lex-yacc/linux.html	0,0194
http://www.cs.bilkent.edu.tr/courses.html	0,0194
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315/lex-yacc/anbn	0,0129

Figure 5.2: Recommendation set for the second visitor

In the second example, fully qualified domain name of the visitor in consideration is $d7512b.dorm.bilkent.edu.tr$. In addition the path followed by him is as follows: ($/\sim guvenir$, $/\sim guvenir/courses/$, $/\sim guvenir/courses/cs315/$). The output of the program for this visitor is shown in Figure 5.2.

In this case, the visitor retrieves a set of pages until reaching to the home page of a particular undergraduate course having the address, ($/\sim guvenir/courses/cs315/$). In this case, the recommendation set contains the addresses of some of the other undergraduate courses such as $/\sim ugur/teaching/cs317/$.

Chapter 6

Conclusion and Future Work

In this thesis, we presented a new usage based personalization system. The system we designed and implemented has two major modules, UsageMiner and Recommender. UsageMiner operates off-line for mining the web server log files. On the other hand, Recommender is capable of producing online recommendations for the visitors by using the information learned by Usage Miner.

The system we presented in this thesis achieves the task of presenting more personalized content in web pages. In static web pages, every single person is presented with the same content. On the other hand, each visitor of the web sites has different interests and looks for different kinds of information. Considering this, personalized web sites try to discover the needs and interests of the visitors dynamically and customize themselves accordingly. As a result, the visitors of the personalized web sites have the opportunity of reaching to the information that they desire much more easily than it can be possible in the static web sites.

Personalization of the web sites is really a challenging operation. The biggest challenge in this area is the existence of the excessive number of visitors encountered by the web sites. As the number of visitors of the target web site gets larger, it becomes much more difficult to differentiate between these visitors. One solution for this could be increasing the amount of the data used in recognizing the visitors. For example, we can choose to store much more data in the files produced by off-line part of the system. Unfortunately, the solution is not that easy because we have to consider the performance of the online module. Storage of more data means much more processing for the on-line part of the system. This is a serious drawback for this solution because the speed is critical in online processes. Users of the web are generally so impatient that they are dissatisfied with the slow requests. These facts means we should try to minimize the amount of information that we stored without decreasing the quality of the recommendations.

Another problem that we encountered while working on the system is related to the task of recognizing the relevant data to be used as an input to our mining algorithms. The amount of the irrelevant data contained in the log files is generally larger than the amount of the relevant data. Even the relevant data requires to be preprocessed carefully before it can be fed into the mining algorithms. For this reason, data filtering and preparation is a critical and time consuming operation for web usage mining. The are lots of sources for the irrelevant data in web usage mining. One source of the irrelevant data is the visitors which visit the site without any goal. These kinds of visitors may retrieve the pages in the site randomly. It is important to filter this kind of data for the success of recommendation process. To solve this problem, we consider the data relevant only if it is supported by enough number of visitors.

The experiments that we performed for testing our system show that the recommendations produced are successful in most of the cases. Of course, the best way to measure the success of the recommendations is to observe the number of times in which our recommendations are utilized by the visitors. If the recommended pages are really interesting for the visitors, they will follow the recommended links more. These kinds of analysis can provide a good feedback for measuring the performance of the system and improving it.

The experiments with the off-line part of the system show that the mining process is very costly in terms of time. On the other hand, this is not important for the success of our system, because off-line part of the system works off-line as the name implies. The main objective in this module is to learn as much as possible about the visitors and reduce the size of the data that will be used by Recommender as much as possible without losing the critical information.

One of the drawbacks of the system is that it is difficult to find a good way for combining the two distinct recommendation sets produced by looking at the path and the identity information of the visitors. The scores of the pages in these sets are calculated by different methods. In spite of this, we just add up the scores of the pages in these sets while merging them. But, we think that there could be better ways of calculating the page scores. For the solution of this problem, better merging strategies can be developed in the future.

In fact, the research that we performed is very open to be expanded because the idea of utilizing domain information in recommendation process is new although there are studies on using path information in producing recommendations. In the future work, the system may be modified in such a way that it only utilizes domain information in producing recommendations. Because, domain information says much more about a particular visitor when compared to the path followed by him/her. Lots of visitors in the site may follow the same path and all of these visitors are considered to be same in producing recommendations according to the path information. On the other hand, when we utilize identity information of the visitors we have the opportunity of learning about the behavior of the visitor himself/herself or the similar visitors. Following this kind of a strategy may also be good for solving the problem of calculating the scores of the recommendations.

In fact, the best way to design the off-line part of the system could be giving the major emphasis on domain information of the visitors while still making use of the paths followed by them. In this case, the paths followed by each of the visitors should be stored in such a way that they could be discovered easily by the Recommender. One way to this could be the formation of a tree containing the paths for each and every visitor. These trees could then be saved into the permanent storage and the file containing these trees may be indexed according to the domain names of the visitors having them as we applied in our system. These trees could than be utilized by Recommender for producing more personalized recommendations for the visitors.

As a conclusion, the system we designed and implemented produces successful results and the results are open to be improved by future works. The work we performed could give birth to the new directions of research.

Bibliography

[1] J. Borges and M. Levene. Data mining of user navigation patterns. In *Proc. WEBKDD*, pages 92–111, 1999.

[2] C. M. Brown, B. B. Danzig, D. Hardy, U. Manber, and M. F. Schwartz. The harvest information discovery and access system. In *Proc. 2nd International World Wide Web Conference*, 1994.

[3] M.-S. Chen, J. Han, and P. S. Yu. Data mining: an overview from a database perspective. *IEEE Trans. On Knowledge And Data Engineering*, 8:866–883, 1996.

[4] M.-S. Chen, J. S. Park, and P. S. Yu. Efficient data mining for path traversal patterns. *Knowledge and Data Engineering*, 10(2):209–221, 1998.

[5] R. Cooley, B. Mobasher, and J. Srivastava. Data preparation for mining world wide web browsing patterns. *Knowledge and Information Systems*, 1(1):5–32, 1999.

[6] R. Cooley and J. Srivastava. Web mining: Information and pattern discovery on the world wide web. In *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97)*, 1997.

[7] R. Cooley, P.-N. Tan, and J. Srivastava. Discovery of interesting usage patterns from web data. In *WEBKDD*, pages 163–182, 1999.

[8] Y. Fu, K. Sandhu, and M. Shih. Clustering of web users based on access patterns. In *Proceedings of the 1999 KDD Workshop on Web Mining*, 1999.

[9] I. Khosla, B. Kuhn, and N. Soparkar. Database search using informatiun mining. In *Proc. of 1996 ACM-SIGMOD Int. Conf. on Management of Data*, 1996.

[10] D. Konopnicki and O. Shmueli. W3qs: A query system for the world wide web. In *Proc. of the 21th VLDB Conference*, pages 54–65, 1995.

[11] R. Kosala and H. Blockeel. Web mining reseach: A survey. *SIGKDD Explorations*, 2(1):1–15, 2000.

[12] L. Lakshmanan, F. Sadri, and I. N. Subramanian. A declarative language for querying and restructuring the web. In *Proc. 6th International Workshop on Research Issues in Data Engineering: Interoperability of Nontraditional Database Systems (RIDE-NDS'96)*, 1996.

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

[14] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill & webert: Identifying interesting web sites. In *Proc. AAAI Spring Symposium on Machine Learning in Information Access*, 1996.

[15] J. Pei, J. Han, B. Mortazavi-asl, and H. Zhu. Mining access patterns efficiently from web logs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 396–407, 2000.

[16] M. Perkowitz. Etzioni: Adaptive web sites: Conceptual clustering mining. In *Proceedings of the 16th International Joint Conference on AI (IJCAI-99)*, 1999.

[17] M. Perkowitz and O. Etzioni. Adaptive Web sites. *Communications of the ACM*, 43(8):152–158, 2000.

[18] M. Perkowitz and O. Etzioni. Adaptive sites: Automatically synthesizing web pages. In *Proc. of the Fifteenth National Conference on Artificial Intelligence*, pages 727–732, July,1998.

[19] T. J. R. Armstrong, D. Freitag and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. In *AAAI Spring Symposium on Information Gathering*, 1995.

[20] M. K. S. Schechter and M. Smith. Using path profiles to predict HTTP request. In *Proceedings of 7th International World Wide Web Conference*, 1998.

[21] C. Shahabi, A. Zarkesh, J. Adibi, and V. Shah. Knowledge discovery form users web-page navigation. In *Proceedings of the IEEE RIDE '97 Workshop*, April,1997.

[22] E. Spertus. Parasite: mining structural information on the web. In *Proc. of 6th International World Wide Web Conference*, 1997.

[23] M. Spiliopoulou. The laborious way from data mining to web log mining. *Computer Systems*, 14(2):113–120, 1999.

[24] M. Spiliopoulou, L. Faulstich, and K. Winkler. A data miner analyzing the navigaitional behavior of web users. In *Proc. of International Conference of ACAI'99: Workshop on Machine Learning in User Modelling*, 1999.

[25] M. Spiliopoulou and L. C. Faulstich. WUM: a Web Utilization Miner. In *Workshop on the Web and Data Bases (WebDB98)*, pages 109–115, 1998.

[26] J. Srivastava, R. Cooley, M. Deshpande, and P. Tan. Web usage mining: Discover and applications of usage patterns from web data. *SIGKDD Explorations*, 1(2):12–23, 2000.

[27] D. F. T. Joachims and T. Mitchell. Webwatcher: A tour guide for the world wide web. In *Proceedings of IJCAI97*, 1997.

[28] T. Yan, M. Jacobsen, H. Garcia-Molina, and U. Dayal. From user access patterns to dynamic hypertext linking. *Computer Networks*, 28(7):1007–1014, May,1996.

[29] O. Zaiane, M. Xin, and J. Han. Discovering web access patterns and trends by applying olap and data mining technology on web logs. In *Advances in Digital Libraries*, pages 19–29, April,1998.

[30] O. R. Zaiane and J. Han. Resource and knowledge discovery in global information systems: A preliminary design and experiment. In *Proc. of the First Int'l Conference on Knowledge Discovery and Data Mining*, pages 331–336, 1995.