

PREDICTING NEXT PAGE ACCESS BY TIME
LENGTH REFERENCE IN THE SCOPE OF EFFECTIVE
USE OF RESOURCES

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

by
Berkan YALÇINKAYA
September, 2002

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Halil ALTAY GÜVENİR (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Özgür ULUSOY

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. İbrahim KÖRPEOĞLU

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

PREDICTING NEXT PAGE ACCESS BY TIME LENGTH REFERENCE IN THE SCOPE OF EFFECTIVE USE OF RESOURCES

Berkan YALÇINKAYA

M.S. in Computer Engineering

Supervisor: Prof. Dr. Halil Altay GÜVENİR

September 2002

Access log file is like a box of treasure waiting to be exploited containing valuable information for the web usage mining system. We can convert this information hidden in the access log files into knowledge by analyzing them. Analysis of web server access data can help understand the user behavior and provide information on how to restructure a web site for increased effectiveness, thereby improving the design of this collection of resources. We designed and developed a new system in this thesis to make dynamic recommendation according to the interest of the visitors by recognizing them through the web. The system keeps all user information and uses this information to recognize the other user visiting the web site. After the visitor is recognized, the system checks whether she/he has visited the web site before or not. If the visitor has visited the web site before, it makes recommendation according to his/her past actions. Otherwise, it makes recommendation according to the visitors coming from the parent domain. Here, “parent domain” identifies the domain in which the identity belongs to. For instance, “bilkent.edu.tr” is the parent domain of the “cs.bilkent.edu.tr”. The importance of the pages that the visitors are really *interested in* and the identity information forms the skeleton of the system. The assumption that the amount of time a user spends on

page correlates to whether the page should be classified as a navigation or content page for that user. The other criterion, the identity information, is another important point of the thesis. In case of having no recommendation according to the past experiences of the visitor, the identity information is located into appropriate parent domain or class to get other recommendation according to the interests of the visitors coming from its parent domain or class because we assume that the visitors from the same domain will have similar interests. Besides, the system is designed in such a way that it uses the resources of the system efficiently. “Memory Management”, “Disk Capacity” and “Time Factor” options have been used in our system in the scope of “Efficient Use of the Resources” concept. We have tested the system on the web site of CS Department of Bilkent University. The results of the experiments have shown the efficiency and applicability of the system.

Keywords: access log file, personalization, identity information, recommendation.

ÖZET

KAYNAKLARIN ETKİN KULLANILARAK BİR SONRAKİ SAYFANIN ZAMAN FAKTÖRÜNE DAYALI OLARAK TAHMİN EDİLMESİ

Berkan YALÇINKAYA

Bilgisayar Mühendisliği, Yüksek Lisans

Tez Yöneticisi : Prof. Dr. Halil Altay GÜVENİR

Eylül,2002

Web erişim dosyası web kullanım madenciliği için gerekli olan değerli bilgileri içeren ve keşfedilmeyi bekleyen hazine sandığı gibidir. Bu dosyaları analiz ederek içinde saklı bu bilgileri kullanılabilir bilgi haline dönüştürebiliriz. Web erişim dosyalarının analizi, kullanıcının davranışını anlamada ve etkinliği artırmak için web sitesinin tekrar dizaynının nasıl yapılacağı hakkında bilgi sağlamaya yarar ve böylece kaynak topluluğumuzun dizaynını geliştirme imkanına sahip olabiliriz. Bu tezde ziyaretçileri web üzerinde tanıyarak ilgilerine göre önermelerde bulunacak yeni bir sistem dizayn ederek geliştirdik. Sistem tüm kullanıcı bilgilerini korur ve siteyi ziyaret eden diğer bir kullanıcıyı tanımak için bu bilgileri kullanır. Ziyaretçi tanıldıktan sonra, sistem bu ziyaretçinin daha önceden web sitesini ziyaret edip etmediğini kontrol eder. Eğer ziyaretçi bu siteyi daha önceden ziyaret ettiyse bu kullanıcının geçmiş hareketlerine dayalı bir önermede bulunur. Aksi takdirde, bu kullanıcının bağlı bulunduğu üst etki alanından gelen ziyaretçilerin ilgi alanlarına göre bir önermede bulunur. Burada, “Ata Etki Alanı” o kullanıcının bağlı olduğu üst etki alanını ifade eder. Örneğin, “bilkent.edu.tr” “cs.bilkent.edu.tr” etki alanının ata etki alanıdır. Ziyaretçilerin gerçekten en çok ilgi duydukları sayfa özelliği ve kimlik bilgisi sistemimizin iskeletini oluşturmaktadır. Bir kullanıcının bir sayfada

geçirdiđi zaman faraziyesi, o sayfanın kullanıcı için, geçiř veya içerik sayfası olarak sınıflandırılması ile ilgilidir. Diđer kriter, yani kimlik bilgisi tezimizin diđer önemli noktasını teşkil etmektedir. Ziyaretçinin geçmiş tecrübelerine dayalı önerme mevcut olmadığında, aynı etki alanındaki ziyaretçilerin benzer ilgilere sahip olacağını farz ettiđimizden dolayı bu kullanıcının ait olduđu ata etki alanındaki ziyaretçilerin ilgilerine göre önermede bulunabilmek için kimlik bilgisi uygun ata etki alanı veya sınıf içerisine yerleřtirilmektedir. Bunun yanında, sistem kaynakları daha verimli kullanacak şekilde tasarlanmıřtır. Sistemimizde “Bellek Yönetimi”, ”Disk Kapasitesi” ve “Zaman Faktörü” opsiyonları “Kaynakların Etkin Kullanımı” konsepti dahilinde kullanılmıřtır. Sistemi Bilkent Üniversitesi Bilgisayar Mühendisliđi web sitesinde test ettik. Deneylerin sonucu bize sistemin verimliliđi ve kullanılabilirliđini göstermiřtir.

Anahtar sözcükler: eriřim dosyası, kişiselleřtirme, kimlik bilgisi, önerme.

Acknowledgement

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

I would also like to thank to Assoc. Prof. Dr. Özgür ULUSOY and Asst. Prof. Dr. İbrahim KÖRPEOĞLU for accepting to read and review this thesis and valuable comments.

I would also like to thank to Turkish Armed Forces for giving us such an opportunity.

Finally, I would like to express my thanks to my wife for her love and support.

Contents

1. Introduction	1
2. Background	2
2.1 General Web Usage Mining	11
2.2 Business Intelligence	14
2.3 System Improvement.....	16
2.4 Site Modification.....	18
2.5 Personalization.....	19
3. NextPage	28
3.1 Log Analyzer	29
3.1.1 Data Preparation	30
3.1.1.1 Elimination of irrelevant items	33
3.1.1.2 Elimination of entries containing frame pages	34
3.1.2 Determination of Frame Pages	36
3.1.3 Session Identification.....	38
3.1.4 Classifying Identity Information	50
3.1.4.1 IP Addresses	50
3.1.4.2. Fully Qualified Domain Names	52

3.1.5 Inserting Identity Information into the tree.....	55
3.1.6 Storing Session Information and Indexing	61
3.2 Recommendation Engine.....	68
3.2.1 Discovery of the pages to be recommended	69
4. Efficient Use of Resources	78
4.1 Efficient use of the main memory.....	78
4.2 Efficient use of the disk capacity	80
5. Evaluation	88
6. Conclusions and Future Work	98

List of Figures

1.1 Main Architecture of Web Usage Mining	6
2.1 Knowledge Discovery Domains of Web Mining	8
2.2 Architecture of the whole system	10
3.1 An example entry in the access log file.....	30
3.2 A series of entry with frame pages.....	35
3.3 The entry after modification	35
3.4 An example entry with search engines in Referrer field.....	36
3.5 The same entry after modification	36
3.6 Algorithm Frame_Detector	37
3.7 A sample user session	40
3.8 Algorithm used for session identification	42
3.9 An example illustrating the creation of a new session node.....	44
3.10 Algorithm Eliminate_Session	47
3.11 A fragment of the sessions created.....	49
3.12 Domain Name Hierarchy.....	54
3.13 A series of identities.....	56
3.14 The root structure of the tree	57

3.15	Algorithm Insert_Tree.....	58
3.16	Classification of the IP addresses.....	59
3.17	Tree structure holding the identity information.....	62
3.18	Algorithm Construct_Tree.....	63
3.19	The tree constructed before the execution of the module.....	64
3.20	Algorithm Create_Result_File.....	66
3.21	PHP script embedded into an HTML page.....	69
3.22	Algorithm used in the FindPage.....	71
3.23	A part of the tree with the example entries.....	72
3.24	The same part of the tree after updating the index fields.....	73
3.25	Index_Table.....	74
3.26	Algorithm Discover_Pages.....	74
3.27	Index_Table for the given identities.....	76
4.1	Algorithm Use_Memory_Efficient.....	79
4.2	Calculation of the time spent.....	83
4.3	Algorithm Forget.....	86
5.1	Main phases of the Log Analyzer module.....	92
5.2	A Sample Fragment of the IndexFile.....	94
5.3	A Sample Fragment of the Result_File.....	95
5.4	A Sample Output of the Recommendation Engine.....	97

List of Tables

3.1 Example entries in the Index File	63
3.2 Example identities with their start and end indexes.....	71
5.1 Test results of the Preprocessing Algorithm for 10-day period. Size values are in byte	89
5.2 Test results of the Session Identification Algorithm.....	90
5.3 Test results of the identity information and the size of the Index and Result File. Size values are in byte.....	91
5.4 Test results of each phase of the Log Analyzer module. Time values are in seconds.	92
5.5 Test Results of Forgetting Algorithm. Time values are in seconds while the sizes are in bytes.....	96

Chapter 1

Introduction

The World Wide Web is a large, distributed hypertext repository of information, where people navigate through links and view pages through browsers. The huge amount of information available online has made the World Wide Web an important area for data mining researches.

The ease and speed with which business transactions can be carried out over the web has been a key driving force in the rapid growth of electronic commerce. Electronic commerce is the focus of much attention today, mainly due to its huge volume. The ability to track browsing behavior of the users has brought vendors and end customers closer than ever before.

Web personalization can be described as any action that makes the web experience of a user personalized to the user's taste. The experience can be something as casual as browsing the web or as significant as trading stocks or purchasing a car. Principal elements of web personalization include modeling of web objects (pages, etc.) and subjects (users), categorization of objects and subjects, matching between and across objects and/or subjects, and determination of the set of actions to be recommended for personalization.

Personalizing the web experience for a user is the main idea for the most web usage based applications. Nowadays, making dynamic recommendations to the user based on his/her past experiences has become very attractive for many applications. The examples of this type of recommendations can be especially found in e-commerce applications.

Understanding the common behavioral patterns of the customers makes the e-commerce companies gain more customers and sell more products through the web. The design of an e-commerce site is critical since their web site is a gateway to their customers. All identity and behavior information about their customers are kept in the access log files as a hidden treasure. Any company that uses web usage mining techniques to filter out the information in access log files has more chance than the others by making their sites more attractive based on the common behavioral patterns of the customers. Nowadays, all e-commerce companies apply data mining techniques on access log files to get more information about their customers and to recognize them through the web. It is a fact that e-commerce sites that have an ability recognizing their customers, adapting their sites or making dynamic recommendations according to the past experiences of the customers save lots of money to the company.

Most existing tools provide mechanism for reporting user activity in the servers and various forms of data filtering. By using these tools, determination of the number of accesses to the server and to individual files, most popular pages, the domain name and URL of the users who visited the site can be solved, but not adequate for many applications. These tools do not help the Webmaster for the analysis of data relationships among the accessed files and the directories within the web site such as [13][14]. These tools have no ability in-depth analysis and also their performance is not enough for huge volume of data. Researches have shown that the log files contain critical and valuable information that must be taken out. It makes web usage mining a popular research area for many applications in the last years.

Another important point of the web usage mining arises in the efficient use of resources. Because the size of the access log files increases in a high rate, the system must handle this option in the scope of using the resources efficiently. Otherwise, if this option could not be taken into account, the system may be off in the future. All limitations including the memory and the resources the system have, must be taken into consideration while an application is being developed. In this context, the system must start a new process to make the usage of resources more efficient when the limits exceed the threshold determined before.

In the thesis, we present a new usage mining system, called as NextPage. The main idea is the prediction of the next page to be retrieved by recognizing the visitor and analyzing the session information belong to the visitor. As discussed above, one way to recognize the user is to use cookies. The main purpose of using cookies in applications is to identify users and possibly prepare customized web pages for them. When you enter a web site using cookies, you may be asked to fill out a form providing such information as your name and interests. This information is packaged into a cookie and sent to your web browser that stores it for later use. The next time you go to the same web site, your browser will send the cookie to the web server. The server can use this information to present you with custom web pages. So, instead of seeing just a generic welcome page you might see a welcome page with your name on it. For example, when you browse through an "online shopping mall" and add items to your "shopping cart" as you browse, a list of the items you have picked up is stored by your browser so that you can pay for all of the items at once when you are finished shopping. It is much more efficient for each browser to keep track of information like this than to expect the web server to have to remember who bought what, especially if there are thousands of people using the web server at a time.

Cookies are harmless in general and the option of turning off the "Always confirm before setting a cookie" feature in your browser is recommended. In case of

being turned on the feature described above really makes the user annoyed. The wide range usage of cookies compel the companies use them to have a chance to exist in the future.

There may be certain cases when you will want to reject cookies, but these probably do not come up that often. Let's say you are visiting a site using a browser that is not on your own personal machine - like a public terminal, or your boss's machine at work. In that case, you might not want a record of your shopping cart, or the sites that you visit, to be kept around where anyone can look at them. Since the browser saves a copy of the cookie information to your local hard drive, it leaves a record that anyone can rifle through if they have the inclination. Another thing to think about is the rare case when some secret or valuable piece of information is being transferred via a cookie. Some of the more advanced web sites will actually do login authentication through HTTP cookies. In this case, you may want to make sure the cookies you are served encrypt your password before reflecting that information back across the net to your personal browser. For sensitive information, use the golden rule: If everyone can see what is being sent, then anyone can find that information by looking at your cookie file or by filtering through the traffic in your vicinity on the net. However, if the information is encrypted (that is, you can not actually read your password by looking in your cookie file), then it is probably OK.

In this regard, the disadvantage of rejecting the cookies made us to accept another way of recognizing the visitor. The way we have chosen is to keep all information about the visitors in the server side and use this information by online mechanism of the system after obtaining the identity information of the visitors through the web and recommend them the pages according to the profile of the visitor.

The system designed and implemented here focuses on the problem of prediction, that is, of guessing future requests of the user for web documents based on their previous requests. The result of the system is a list of pages as a

recommendation set at the end of the web document. The goal of making recommendation to the user is to provide the user an easy access to the pages that he/she may be interested in. Our starting point of the design of the system is to make the user's surfing easier by recommending the pages that can be only accessed after a retrieval of a number of pages in any particular page. As a result, the visitor may reach to the page by just clicking on its link instead of making a number of retrieval.

Another question that deserves attention is what the system recommends any visitor who has never visited the site before. In these cases, the system parses the IP address or FQDN of the visitor to find its parent domain. The system also keeps all information about all parent domains reside in the World Wide Web. If the system produces no recommendation for a new visitor, then it searches the next access pages to be recommended in the sessions of the parent domain of the visitor. The system repeats this process until it has enough number of recommendation determined by the web master.

The system developed is under the category of usage-based personalization. It has two main modules, Log Analyzer and Recommendation Engine. Log Analyzer module analyzes the log file kept by the server to determine the patterns and information about the visitors. The main files formed by Log Analyzer are the file containing the session information of the visitors (Result File) and the file containing the indexes (Index File) of sessions belong to the visitors. The information obtained by the Log Analyzer module is used by Recommendation Engine module to produce recommendation set for the visitor. Recommendation Engine acquires the identity and document information by the help of PHP script code that is embedded into the HTML page. Then, it searches the pages to be recommended in the Result File by using the index variables kept in Index File. After processing and producing the recommendation, Recommendation Engine shows them to the visitor in a table at the bottom of the document.

The general architecture of the system can be summarized as in Figure 1.2. As shown in the figure, Log Analyzer mines the log data to produce information and pattern about the visitors. Recommendation Engine module uses the Index and Result File formed by the Log Analyzer module by executing a CGI program. Log Analyzer module runs offline at specific times while Recommendation Engine module runs online for every request for the resources keeping PHP script code in. In the following chapters, the details of the system will be discussed in more detail.

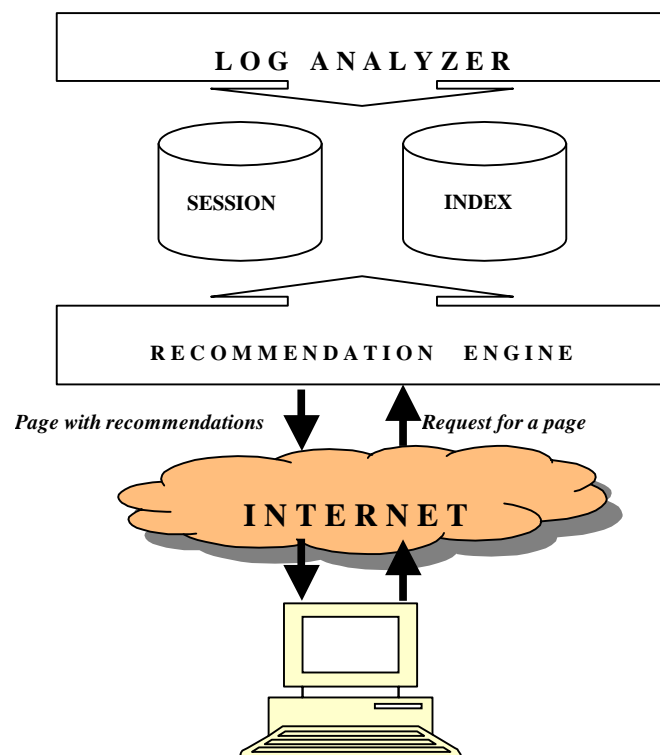


Figure 1.1: Architecture of the system

An overview of the previous work done related to the thesis will be given in Chapter 2. The detailed explanation of Log Analyzer and Recommendation Engine module will be given in Chapter 3. Chapter 4 is devoted to the efficient use of the resources. The results of the experiments and evaluation will be discussed in Chapter 5 and we will conclude with Chapter 6.

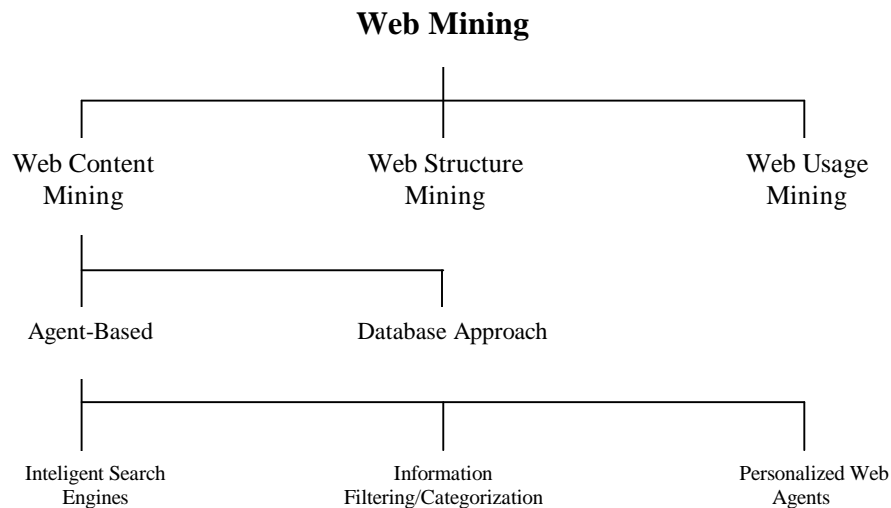
Chapter 2

Background

In this chapter, we discuss related work in the literature and present the relevant background concepts for the thesis. Web servers register a log entry for every single access they get. A huge number of accesses (hits) are registered and collected in an ever-growing access log file. By mining the access log files maintained by the web servers we may enhance server performance, improve web site navigation, improve system design of web applications.

Data mining and World Wide Web are two important and active areas of current researches. A natural combination of the two areas, sometimes referred to as Web Mining, has been the focus of several recent research projects and papers. Web mining can be described as the discovery and analysis of useful information from the World Wide Web [1]. Main goal of web mining is the extraction of interesting and useful patterns and information from activities related to the World Wide Web. This means the automatic search of information resources available online. The search may be either in Web Content Mining or in Web Usage Mining. Web Mining can be roughly classified into three knowledge discovery domains as shown in Figure 2.1: *Web Content Mining, Web Structure Mining and Web Usage Mining*

Web content mining, is described as the process of information or resource discovery from millions of sources across the World Wide Web. Web Content Mining studies can be divided into two main approaches, namely *agent-based*



approach and database approach [1].

Figure 2.1: Knowledge Discovery Domains of Web Mining

Generally, agent-based web mining systems can be placed into three categories. *Intelligent Search Agents* uses domain characteristics and user profiles to organize and interpret the discovered information such as Harvest[2], Parasite[3] and Shop-Boot[4]. *Information Filtering/Categorization* uses various information retrieval techniques[5] and characteristics of open web documents to automatically retrieve, filter and categorize them. *Personalized Web Agents* learn user preferences and discover web information sources based on these preferences and those of other individuals with similar interests such as WebWatcher[6], Sykill & Webert[7].

The aim of database approaches to web mining is to organize semi-structured web pages into more structured collections of resources. Then known database querying systems and data mining techniques are applied on these databases created

to analyze them. Database approach is divided into two classes. Multilevel Databases store all semi-structured hypertext documents at the lowest level of the databases and uses them for higher levels to have Meta data and generalizations. On the other hand, Web Query Systems make the analysis of the data created easier. They use standard database query languages such as SQL for the queries that are used in WWW such as W3QL[8].

Web structure mining is the application of data mining techniques for the data describing the organization of the content. Design of a web site centers around organizing the information on each page and the hypertext links between the pages in a way that seems most natural to the site users in order to facilitate their browsing and perhaps purchasing. In this context, Intra-page structure information includes the arrangement of various HTML or XML tags within a given page. The principal kind of inter-page structure information is hyperlinks connecting one page to another page in a web site. In other words, it is focused on the structure of the hyperlinks within the web itself. Most research on the web structure mining can be thought of a mixture of content and structure mining and add content information to the link structures such as Clever System[10] and Google[11].

Web Usage Mining focuses on techniques that could predict user behavior while the user interacts through the web. We define the mined data in this category as the secondary data since they are all the result of interactions. We could classify them into the usage data that reside in the web clients, proxy servers and web servers[9]. The web usage mining process could be classified into two commonly used approaches[12]. The former approach maps the usage data into relational tables, whereas the latter approach uses the log data directly by utilizing special preprocessing techniques. Web usage mining can also be defined as the application of data mining techniques to discover user web navigation patterns from web access log data[9]. Log files provide a list of the page requests made to a given web server in which a request is characterized by, at least, the IP address of the machine placing

the request, the date and time of the request and the URL of the page requested. From this information, it is possible to derive the user navigation sessions within the web site where a session consists of a sequence of web pages viewed by a user in a given time window. Any technique to identify patterns in a collection of user sessions is useful for the web site designer since it may enhance the understanding of user behavior when visiting the web site and therefore providing tips for improving the design of the site.

Web usage mining has mainly three phases: *preprocessing*, *pattern discovery* and *pattern analysis*. Preprocessing consists of converting the usage, structure and content information contained in the various available data sources into the data abstractions necessary for pattern discovery. Pattern discovery can be divided into the categories, statistical analysis, association rules, clustering, classification, sequential patterns and dependency modeling[9]. Pattern analysis is the last step of web usage mining that aims to filter out interesting rules or patterns from the set found in the pattern discovery phase. The most common way of pattern analysis is a query mechanism such as SQL.

The main application areas of web usage mining can be depicted in Figure 2.2

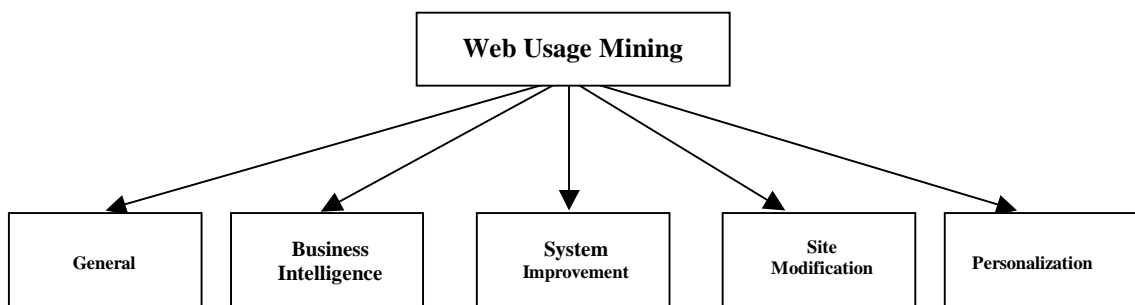


Figure 2.2: Main Application Areas of Web Usage Mining

As shown in the figure, usage patterns extracted from web data have been applied to a wide range of research areas. Projects such as WebSIFT [9], WUM [10], SpeedTracer [30] have focused on web usage mining in general.

2.1 General Web Usage Mining

The aim of a general web usage mining system is to discover general behavior and patterns from the log files by adapting well-known data mining techniques or new approaches proposed. Most of the researches aim to discover user navigation paths or common behavior from access log files whereas some of the studies focus on clustering to find the similar interest groups among visitors.

One of the studies, Hypertext Probabilistic Grammars [12], focuses on mining access patterns of visitors. In this study, user navigation session is defined as a sequence of page requests such that no consecutive requests are separated by more than a specific time period. These user navigation sessions derived from log files are then modeled as a hypertext probabilistic grammar (HPG). There are two states, S and F, which represent the start and finish states of the navigation sessions. The set of strings, which are generated with higher probability, correspond to the navigation trails preferred by the user. Moreover, the concept of an N-grammar is used to determine the assumed memory when navigating within the site. For a given N it is assumed that only N previously visited pages influence the link the user will choose to follow next. After the construction of the HPG the paths are discovered by using Depth-First search algorithm. Before mining process, support and confidence thresholds must be specified. 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 is obtained by the probability of the derivation of the first state of the path from the start state while confidence

threshold is obtained from the derivation probabilities of other pages on the path. The value of support and confidence threshold affects the quality of the paths discovered.

An approach similar to association rule mining, called Maximal Forward (MF) Reference, is proposed in [34]. 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. The algorithm derived, MF, converts the original log data into a set of traversal subsequences. This process also filters out the effect of backward references that are mainly made for ease of traveling. As an example, assume the path traversed by any user is as follows $\langle A, B, C, D, C, B, E, F, E, G \rangle$ would be broken into three transactions of $\langle A, B, C, D \rangle$, $\langle A, B, E, F \rangle$ and $\langle A, B, E, G \rangle$. At the end of processing MF algorithm, we get all Maximal Forward Reference sequences and these sequences are stored in a database. Two main algorithms, Full Scan (FS) and Selective Scan (SC) are derived to determine the frequent traversal patterns, termed large reference sequences from the Maximal Forward References obtained by the algorithm MF, where a large reference sequence is a reference sequence that appeared in a sufficient number of times in the database. Algorithm FS is required to scan the transaction database in each pass and utilizes key ideas to the Direct Hashing with Pruning (DHP). In contrast, by properly utilizing the candidate reference sequences, the second algorithm devised, Selective Scan, is able to avoid database scans in some passes so as to reduce the disk I/O cost. Maximal reference sequences are the subset of large reference sequences so that no maximal reference sequence is contained in the other one. If the large reference sequences are AB, AE, AGH, ABD then maximal reference sequences become AE, AGH, and ABD.

WebSift [9] project is one of the global architectures to handle the web usage mining. WebSift establishes a framework for web usage mining. The system has three main phases. Preprocessing, Pattern Discovery and Pattern Analysis. Preprocessing phase is for converting the usage information contained in web server log files into data abstractions necessary for pattern discovery. The preprocessing

algorithm includes identifying users, server sessions and inferring cached page references through the use of the Referrer field. In the second phase, well known data mining techniques are applied such as association rule mining, sequential pattern mining or clustering on the data abstraction obtained in the preprocessing phase. At the last step, the results of the various knowledge discovery tools analyzed through a simple knowledge query mechanism, a visualization tool (association rule map with confidence and support weighted edges). An information filter based on domain knowledge and the web site structure is applied to the mined patterns in search for the interesting patterns. Links between pages and the similarity between contents of pages provide evidence that pages are related. This information is used to identify interesting patterns, for example, item sets that contain pages not directly connected are declared interesting.

WUM [18] is one of the tools used for mining user navigation patterns from access log files. It employs an innovative technique for the discovery of navigation patterns over an aggregated materialized view of the access log file. This technique offers a mining language as interface to the expert, so that the generic characteristics can be given, which make a pattern interesting to the specific person. The system has two main modules. The Aggregation Service prepares the access log file for mining and the Query-Processor does the mining. In WUM, individual navigation paths called trails are combined into an aggregated tree structure. Queries can be answered by mapping them into the intermediate nodes of the tree structure. The aggregate tree is formed by merging trails with the same prefix. Each node in the tree contains a URL and these nodes is annotated with the number of visitors having reached the node across the same trail prefix, that is, the support of the node. Query processor is the module responsible for the mining on the aggregate tree formed by the Aggregation Service. Queries can be answered by mapping them into the intermediate nodes of the tree structure.

SpeedTracer [30], SpeedTracer is a web usage mining and analysis tool which tracks user browsing patterns, generating reports to help Webmaster to refine web site structure and navigation. SpeedTracer makes use of Referrer and Agent information in the preprocessing routines to identify users and server sessions in the absence of additional client side information. The application uses innovative inference algorithms to reconstruct user traversal paths and identify user sessions. Advanced mining algorithms uncover users' movement through a web site. The end result is collections of valuable browsing patterns that help Webmaster better understand user behavior. SpeedTracer generates three types of statistics: user-based, path-based and group-based. User-based statistics point reference counts by user and durations of access. Path-based statistics identify frequent traversal paths in web presentations. Group-based statistics provide information on groups of web site pages most frequently visited.

In [39], the authors propose a novel data structure and a new algorithm to mine web access patterns from log data. The web access sequences are stored in a tree like data structure, the WAP-tree, which is more compact than the initial access in the tree. However, the tree inferred from the data is not incremental since it includes only the frequent access sequences. Moreover, although the algorithm is efficient, the performance analysis should take into account the time needed to build the tree, since the input data for the tree construction is in the form used by the algorithm against which the proposed method is compared.

2.2 Business Intelligence

The information on how customers are using a web site is critical for especially e-commerce applications. Buchner and Mulvenna present a knowledge discovery process in order to discover marketing intelligence from web data [35]. They define a web access log data hypercube that consolidates web usage data along with

marketing data for e-commerce applications. Four distinct steps are identified in customer relationship life cycle that can be supported by their knowledge discovery techniques: customer attractions, customer retention, cross sales and customer departure.

There are more than 30 commercially available applications. But many of them are slow and make assumptions to reduce the size of the log file to be analyzed. These applications are all useful for generating reports about the site such as

- Summary report of hits and bytes transferred
- List of top requested URLs
- List of top referrers
- List of most common browsers
- Hits per hour/day/week/month reports
- Hits per Internet domain
- Error report
- Directory tree report, etc.

One of these tools described above, WebTrends [31], provides the most powerful e-business intelligence reporting available, enabling customers to track, manage and optimize e-business strategies. WebTrends Log Analyzer reports on all aspects of a web site's activity including how many people have visited a web site, where they come from, and what pages interest them most. But it is a fact that these tools are limited in their performance, comprehensiveness, and depth of analysis.

In [40], web server logs have been loaded into a data cube structure in order to perform data mining as well as Online Analytical Processing (OLAP) activities such as roll-up and drill-down of the data. In the WebLogMiner project, the data collected in the access log files goes through four stages. In the first stage, the data is filtered to remove irrelevant information and a relational database is created containing the meaningful remaining data. This database facilitates information extraction and data summarization based on individual attributes like user, resource,

user's locality, day, etc. In the second stage, a data cube is constructed using the available dimensions. OLAP is used in the third stage to drill-down, roll-up, slice and dice in the web access log data cube. Finally, in the fourth stage, data mining techniques are put to use with the data cube to predict, classify, and discover interesting correlations.

2.3 System Improvement

The problem of modeling and predicting of a user's access on a web site has attracted a lot of research interest. One of the aims of predicting the next page request is improving the web performance through pre-fetching. The objective of pre-fetching is the reduction of the user perceived latency. Potential sources of latency are the web servers' heavy load, network congestion, low bandwidth, bandwidth underutilization and propagation delay. There seem some obvious solutions to reduce the effects of the reasons described above. One of them may be increasing the bandwidth, but it does not seem a viable solution since the structure of the web cannot be easily changed without significant economic cost. Another solution is to cache the documents on the client's machine or on proxies. But caching solution is limited when web resources tend to change very frequently.

Performance and other service quality attributes are crucial to user satisfaction from services such as databases, networks etc. Similar qualities are expected from the users of web services. Web usage mining provides the key to understand web traffic behavior, which can in turn be used for developing policies for web caching.

Some prediction approaches utilizes path and point profiles generated from the analysis of web server access logs to predict HTTP requests as described in [27]. They used these predictions to explore latency reductions through the pre-

computation of dynamic web pages. The profiles are generated from user session. During a single session, a user interacting with web traverses some sequences, of URLs. From that single sequence, the set of all possible subsequences is extracted as paths. A method is proposed for predicting the next move of the visitor based on matching the visitor's current surfing sequence against the paths in the path profile. The ranking of matches is determined by a kind of specificity heuristic: the maximal prefixes of each path (the first $N-1$ elements of an N -length path) are compared element-wise against the same length suffixes of the user path (i.e. a size $N-1$ prefix is matched against the last $N-1$ elements of the user path) and the paths in the profile with the highest number of element-wise matches are returned. Partial matches are disallowed. In other words, if a visitor's path were $\langle A, B, C \rangle$, indicating the visitor visited URL A, then URL B, then URL C, the path would be better matched by a path in the profile of $\langle A, B, C, D \rangle$ than $\langle B, C, E \rangle$. For the paths in the profile that match, the one with the highest observed frequency is selected and used to make prediction. Using our example, if $\langle A, B, C, D \rangle$ were the best match and most frequently observed path in the profile, then it would be used to predict that the user who just visited $\langle A, B, C \rangle$ would next visit URL D.

A first order Markov model is proposed in [37] to implement a pre-fetching service aimed at reducing server load. The model is built from past usage information and the transition probabilities between pages are proportional to the number of times both pages were accessed in a predefined time window. We note that the use of a time window results in having transitions with probability greater than zero between pages that were never accessed consecutively. The results of the conducted experiments show that the method is effective in reducing the server load and the service time. A similar method is proposed in [38] wherein a dependency graph is inferred and dynamically updated as the server receives requests. There is a node for every requested page and an arc between two nodes exists if the target node was requested within n accesses after the source node; the weight of an arc is

proportional to the number of such requests. The simulations performed with log data show that a reduction in the retrieval latency can be achieved.

2.4 Site Modification

The attractiveness of a web site, in terms of both content and structure, is the main idea of many applications, especially for a product catalog for e-commerce applications. The structure and the attractiveness of the web site is crucial because web sites are the only way between the company and their visitors. Web Usage Mining provides detailed feedback on user behavior, providing the web site designer with information on which to base redesign decisions. Web usage data provides an opportunity to turn every site into an ongoing usability test. While the information is not as complete as the information that can be gathered from a formal usability analysis with videos and trained observers. Web usage data are cheap and plentiful.

Designing a good web site is not a simple task because hypertext structure can easily expand in a chaotic manner as the number of pages increases. Thus many techniques to improve the effectiveness of user navigation have been proposed. Discovering the gap between the expectations of the web site designer and the behavior of the users helps to improve the restructure of the web site [22]. The expectation of the web site designer is assessed by measuring the inter-page conceptual relevance. Measurement of conceptual relevance is done by a vector space model. All web documents are analyzed by the system to construct the vector. All HTML tags and stop words are discarded to obtain content words. Then the frequency of content words for each page is calculated. Finally the inter-page conceptual relevance (SimC) for each page pair p_i and p_j using the cosine similarity formula is measured. If the number of content words that appear in both pages is 0, the value of SimC is also 0. The measurement of access co-occurrence is done by modifying the vector space model. The number of accesses for each page is

measured by counting the IP addresses in the access log file. Then, the inter-page access co-occurrence (SimA) for each page pair, p_i and p_j , is measured. After SimC and SimA are calculated, the correlation coefficient that is the degree of linear relationship between two variables (SimC and SimA) is measured. The technique finds page pairs that should be improved. It finally shows page clusters meaning clues for web designer to improve the web site and to understand the design problem.

The major main goals of the approach proposed in [24], Adaptive Web Sites, are avoiding additional work for visitors and protecting the original design of the site from destructive changes. The system is to apply only nondestructive transformations meaning that some links can be added on the pages but cannot be removed or some index pages can be created but none of the pages can be removed. The aim is to create an index page containing collections of links to related but currently unlinked pages. An algorithm, PageGather, is proposed to find collections of pages that tend to co-occur in visits. The PageGather algorithm uses cluster mining to find collections of related pages at a web site relying on the visit-coherence assumption. The algorithm process the access log into visits and compute the co-occurrence frequencies between pages and create a similarity matrix. Then a graph corresponding to the matrix is created and cliques are found on that graph. At the end of the algorithm, for each cluster found, a web page consisting of links to the documents in the cluster is formed and recommended to the user.

2.5 Personalization

Web Personalization is the task of making web-based information system adaptive to the needs and interests of individual users or groups of users. Typically, a personalized web site recognizes its users, collects information about their preferences and adapts its services, in order to match the needs of the users. One way

to expand the personalization of the web is to automate some of the processes taking place in the adaptation of a web-based system to its users.

SiteHelper [36] is a local agent that acts as the housekeeper of a web server, in order to help a user to locate relevant information within the site. The agent makes use of the access log data to identify the pages viewed by a given user in previous visits to the site. The keywords characterizing the contents of such pages are incorporated into the user profile. When that user returns to the site, the agent is able, for example, to show the changes that took place in pages that are known to be interest and also to recommend any new pages.

WebWatcher [6], acts like a web tour guide assistant, it guides the user along an appropriate path through the collection based on the past experiences of the visitor. It accompanies users from page to page, suggests appropriate hyperlinks and learns from experience to improve its advice-giving skills. The user fills a form stating what information he is looking for and, as the user navigates the web, the agent uses the knowledge learned from previous users to recommend links to be followed; the links thought to be relevant are highlighted. At the end of the navigation the user indicates whether or not the search was successful, and the model is updated automatically.

Letizia [16] is similar to WebWatcher in the sense that the system accompanies the user while browsing. It is a user interface agent that assists a user browsing the World Wide Web. As the user operates a conventional web browser such as Netscape, the agent tracks user behavior and attempts to anticipate items of interest by exploring of links from the current position of the user. The difference from WebWatcher is that the system serves only one particular user. Letizia is located on the users' machine and learns his/her current interest. The knowledge about the user is automatically acquired and does not require any user input. By doing look ahead search, Letizia can recommend pages in the neighborhood of where the user is currently browsing.

Syskill & Webert [7] is designed to help users distinguish interesting web pages on a particular topic from uninteresting ones. It offers a more restricted way of browsing than the others. Starting from a manually constructed index page for a particular topic, the user can rate hyperlinks on this page. The system uses the ratings to learn a user specific topic profile that can be used to suggest unexplored hyperlinks on the page. Also, the system can also use search engines like LYCOS to retrieve pages by turning the topic profile into a query.

WebPersonalizer [19] system is divided into two components, offline and online component like the system we designed in the thesis. The offline module is responsible for data preparation tasks resulting in a user transaction file. It performs specific usage mining tasks to form clusters from user transactions and URL clusters from the transaction cluster. The other component, online component, provides dynamic recommendations to users. When the server accepts a request, the recommendation engine matches the active session with the URL clusters to compute a set of recommended hyperlinks. The system recommends pages from clusters that match most closely to the current session. Pages that have not been viewed and are not directly linked from the current page are recommended to the user. The recommendation set is added to the requested page as a set of links before the page is sent to the client browser.

The system proposed in [21] is based on the two main user profiles depending on the navigation strategy. The user can either return to the same objects over and over or always visit a new object. The first user, called as “net surfer”, is more interested in exploring the cyberspace than to explore what the document can offer him while the other user, called as “conservative”, is more concerned with exploring the contents of the objects in a certain site. Because user profiles perform an important role in the effectiveness of pre-fetching, two empirical user models were constructed. Random Walk User Model captures the long-term trend. The second model, Digital Signal Processing (DSP) User Model, applies to the short-term

behavior. Both models are able to track user's behaviors. The algorithm devised has two main parts. Preparation phase computes the user's profile curve. Prediction phase initially determines in the last accesses how conservative the user was. Then the prediction is made based on the user profile detected.

WebTool, an integrated system [23], is developed for mining either association rules or sequential patterns on web usage mining to provide an efficient navigation to the visitor, the organization of the server can be customized and navigational links can be dynamically added. The system has a 2-phase process. The preprocessing phase removes irrelevant data and performs a clustering of entries driven by time considerations. In the second phase, data mining techniques are applied to extract useful patterns or relationships and a visual query language is provided in order to improve the mining process. A generator of dynamic links in web pages uses the rules generated from sequential patterns or association rules. The generator is intended for recognizing a visitor according to his navigation through the pages of a server. When the navigation matches a rule, the hypertext organization of the document requested is dynamically modified. The hyperlinks of the page are dynamically updated according to the rule matched.

Another approach [25] has the idea of matching an active user's pattern with one or more of the user categories discovered from the log files. It seems under the category of user-based web personalization system. The system has two main module, Offline and Online module. In the offline module, the preprocessor extracts information from web server log files to generate records of users sessions. For every session in the log file, one record is generated. The records generated are then clustered into categories, with similar sessions put into the same category. A user session is represented by n -dimensional vector (assuming n interest items in the site) in the preprocessing phase. Each interest page in the vector has a weight depending on the number of times the page is accessed or the amount of time the user spends on the page. Such an n -dimensional vector forms a user session record mentioned

above. After all sessions are represented in a vector format, LEADER algorithm which is also a clustering algorithm is applied on these vectors formed to discover clusters of session vectors that are similar. After finding of the clusters, the median of each cluster is computed as a representative of the clusters. The other module of the approach is responsible to make dynamic recommendations to the user. The module temporarily buffers the user access log in main memory to detect the pages the user retrieved before. The active session information is maintained the same type of vectors as in the preprocessing phase. For every page request of the user, the vector is updated automatically. The system tries to match the active session vector to the existing clusters formed by the offline module. Then the pages in the vector that the user has not accessed so far and are not accessible from the URL just requested are suggested to the user at the top of the page she/he requested.

Another prediction system called WhatNext [26] is focused on path-based prediction model inspired by n -gram prediction models commonly used in speech-processing communities. The algorithm build is n -gram prediction model based on the occurrence frequency. Each sub-string of length n is an n -gram. The algorithm scans through all sub-strings exactly once, recording occurrence frequencies of the next click immediately after the sub-string in all sessions. The maximum occurred request is used as the prediction for the sub-string.

In [28], the authors proposed to use Markov chains to dynamically model the URL access patterns that are observed in navigation logs based on the previous state. Markov chain model can be defined by the tuple $\langle S, A, \Pi \rangle$ where S corresponds to the state space; A is the matrix representing transition probabilities from one state to another. Π is the initial probability distribution of the states in S . If there are n states in the Markov chain, then the matrix of transition probabilities A is of size $n \times n$. Markov chain models can be estimated statistically, adaptively and are generative. The probabilistic Link Prediction System described has five major components. In the “Markov Chain Model” component, a (sparse) matrix of state transition

probabilities is constructed. In the “Client Path Buffer”, a buffer is assigned in the main memory to hold client requests and all the sequence of client requests stored in that buffer. In the “Adaptation Module” the matrix created is updated with the user path trace information. The “Tour Generator” outputs a sequence of states for the given start URL. The last module “Path Analysis and Clustering” clusters the states into similar groups to reduce the dimensionality of the transition matrix. The system proposed is used in HTTP request prediction, in adaptive web navigation, in tour generators, in personalized hub/authority.

In [29], the authors describe a tool named WebMate, a proxy agent that monitors the user web navigation while building his profile. Each time the user finds an interesting page he points the page to the agent. The agent analyses the contents of the page and classifies it into one of a predefined set of classes. In this way, the user profile is inferred from a set of positive training examples. In off peak hours the agent browses a set of URLs the user wants to have monitored in search for new relevant pages. If the user does not specify URLs to be monitored the agent uses a set of chosen keywords to query popular search engines and assess the relevance of the returned pages.

The WebMiner system [1][32], divides the web usage mining process into three main phases. In the first phase, called as preprocessing phase, includes the domain dependent tasks of data cleaning, user identification, session identification and path completion. In the second phase, called as the knowledge discovery phase, especially association rule and sequential pattern generation algorithms applied on the data obtained in the first phase. The discovered information is then fed into various pattern analysis tools. The site filter is used to identify interesting rules and patterns by comparing the discovered knowledge with the web site designer’s view of how the site should be used. At the same time, the site filter can be applied to the data mining algorithms in order to reduce the computation time or the discovered rules and patterns.

Another prediction system proposed in [15] is based on the assumption of mining longest repeating subsequences to predict www surfing. In this approach, a longest prediction subsequence is defined as a sequence of items where subsequence means a set of consecutive items, repeated means the item occurs more than some threshold T and longest means that although a subsequence may be part of another repeated subsequence, there is at least once occurrence of this subsequence where this is the longest repeating.

Another usage based personalization system, which is slightly different than the others, is proposed in [17]. It is capable of guessing the web pages and showing these web pages that have the highest scores as a recommendation set to the visitor. The system is based on two criteria, the path followed by the visitors and the identity information. It has two major modules like many applications based on usage-based and prediction system, Offline and Online module. The off-line module mines the access log files for determining the behavioral patterns of the previous visitors of the web site considered. It has also two sub modules called as PathInfoProcessor and HostIdentityProcessor. The aim of the former is to find user navigation paths hidden in the access log file and store them in a form to be utilized by the online module whereas the aim of the latter, is to discover the relations between the identity information and navigation patterns of visitors and store the results that has been discovered. All paths discovered are maintained in a path tree and this path tree is updated with the new path information of the current day. The path tree created is then stored in such a file that the online module will spend minimum amount of time on creating and accessing it. The other major module of the system, Online Module, is triggered by a java applet embedded into the HTML page. The java applet is used for the connection between the client and the server. The java applet triggers a PERL script to acquire the identity information of the visitor and then the identity information acquired is sent to a CGI program, which is the main part of the online module. The CGI program finds two separate sets of recommendation according to the path and the identity information. The module searches the path tree whether the

path of the visitor exists or not. Then a score for each page coming after the page that includes the java applet on that path tree is evaluated based on the frequencies of the pages. Another set of recommendation is found for the identity information. The recent paths followed by the same identity are checked to find the pages to be recommended. At the end of the recommendation phase, these two sets of pages are merged to form a single set and recommended to the visitor.

The approach presented in [20] focuses on the use of the resources efficiently. The starting point of the approach is the learning and the memorization. When an object is observed or the solution to a problem is found, it is stored in memory for future use. In the light of this observation, memory can be thought of as a look up table. When a new problem is encountered, memory is searched to find if the same problem has been solved before. If an exact match for the search is required, learning is slow and consumes very large amounts of memory. However, approximate matching allows a degree of generalization that both speeds learning and saves memory. Three experiments were conducted to understand the issues better involved in learning prototypes. IBL learns to classify objects by being shown examples of objects, described by an attribute/value list, along with the class to which each example belongs. In the first experiment (IB1), to learn a concept simply required the program to store every example. When an unclassified object was presented for classification by the program, it used a simple Euclidean distance measure to determine the nearest neighbor of the object and the class given to it was the class of the neighbor. This simple scheme works well, and is tolerant to some noise in the data. Its major disadvantage is that it requires a large amount of storage capacity. The second experiment (IB2) attempted to improve the space performance of IB1. In this case, when new instances of classes were presented to the program, the program attempted to classify them. Instances that were correctly classified were ignored and only incorrectly classified instances were stored to become part of the concept. While this scheme reduced storage dramatically, it was less noise-tolerant than the first. The third experiment (IB3) used a more sophisticated method for evaluating instances to

decide if they should be kept or not. IB3 is similar to IB2 with the following additions. IB3 maintains a record of the number of correct and incorrect classification attempts for each saved instance. This record summarized an instance's classification performance. IB3 uses a significance test to determine which instances are good classifiers and which ones are believed to be noisy. The latter are discarded from the concept description. This method strengthens noise tolerance, while keeping the storage requirements down.

Chapter 3

NextPage

As described above, one of the common properties of the applications developed on web usage mining, especially under the category of personalization, is the prediction of the next pages to be accessed. This property makes the web site, especially for e-commerce companies, more attractive for the visitors. The aim of the system presented is to predict next access pages to help visitors while navigating the web site by analyzing the access log files. The system developed is designed to recognize the user visiting the site and recommend the pages based on her/his past experiences. If the system does not have any information about the visitor, that is, a new visitor for the system, then it finds the parent domain of the visitor by parsing its identity information and recommends the pages according to the interests of the visitors from the parent domain. The process continues until the number of recommendation derived satisfies the number determined by the web master.

NextPage consists of two independent modules shown as in Figure 1.2. Log Analyzer and Recommendation Engine. Log Analyzer is the main part of the system that produces the Result File containing the session information and the Index File containing the index variables of the identities used by the Recommendation Engine. These files contain the relation between the identity information and the navigation patterns of the visitors.

3.1 Log Analyzer

Log Analyzer module analyzes the access log file maintained by the web server to determine the identity information and session identification. It has mainly four phases, Data Preparation, Session and User Identification, Indexing / Storing and Forgetting (when necessary) phases. In the following sections, detail of each phase of the Log Analyzer will be explained.

Our usage mining system is designed to run on predetermined times of the day automatically to process the newly added entries of the access log file. To achieve this process, Log Analyzer module has two contingencies. One of them is the probability of being the same log file one day before whereas the other is that of being a new log file. If the log file is the same as the log file one day before, then it finds the last entry it processed and begins to process the entries from that entry. Otherwise if it is a new one, then it begins to process from the first entry of the log file. The module keeps the first entry and the size of the access log file in a file called as LogDeterminer. By comparing the entry in the LogDeterminer file and the first entry of the access log file, it determines whether the log file is the same as the log file one day before or not. If comparison is positive, that is, the same log file, then the file pointer is positioned to the entry just after the last entry processed by using the *sizeoflog* variable kept in LogDeterminer file. By storing the size of the log file processed one day before, the module avoids itself to run again on the same entries. If the size of the access log file is greater than the *sizeoflog* variable meaning that there exists newly added entries in the access log file, Log Analyzer module directly begins to process these newly added entries by skipping the entries that have been processed in prior days. The module terminates without doing anything if the first entry and the size of the access log file is the same as the ones kept in the LogDeterminer file meaning that the same log file is being tried to be processed again. The module updates the LogDeterminer file by rewriting the first entry and the size of the log file to the file at every execution of the module.

3.1.1 Data Preparation

The main source of the data preparation phase is the access log file maintained by the web server. An access log file is a text file in which every page request made to the web server is recorded. The format of the log files is related to the configuration file of the web server. Generally, there are two main log formats used. One of them Common Log Format and the other is Combined Log Format. The difference between them is that the former does not store Referrer and Agent information of the requests. The format of the log file kept by the Computer Engineering of Bilkent University web server is NCSA Combined Log Format. A single example entry of the log file is shown in Figure 3.1. An entry is stored as one long line of ASCII text, separated by tabs and spaces.

```
lab30640.bcc.bilkent.edu.tr - - [01/Nov/2001:21:56:52 +0200] "GET
/~guvenir/courses/HTTP/1.1" 200 1749
"http://www.cs.bilkent.edu.tr/guvenir" "Mozilla/4.0 (compatible;
MSIE 5.5; Windows 95)"
```

Figure 3.1: An example entry in the access log file

The details of the fields in the entry are given in the following section.

Address or DNS :

lab30640.bcc.bilkent.edu.tr

This is the address of the computer making the HTTP request. The server records the IP and then, if configured, will lookup the Domain Name Server (DNS) for its FQDN.

RFC931 (Or Identification) :

-

Rarely used, the field was designed to identify the requestor. If this information is not recorded, a hyphen (-) holds the column in the log.

Authuser :

-

List the authenticated user, if required for access. This authentication is sent via clear text, so it is not really intended for security. This field is usually filled by a hyphen -.

Time Stamp :

[01/Nov/2001:21:56:52 +0200]

The date, time, and offset from Greenwich Mean Time (GMT x 100) are recorded for each hit. The date and time format is: DD/Mon/YYYY HH:MM:SS. The example above shows that the transaction was recorded at 21:56:52 on Nov 1, 2001 at a location 2 hours forward GMT. By comparing time stamps between entries, we can also determine how long a visitor spent on a given page that is also used as a heuristic in our system.

Target :

"GET /~guvenir/courses/HTTP/1.1"

One of three types of HTTP requests is recorded in the log. GET is the standard request for a document or program. POST tells the server that data is following. HEAD is used by link checking programs, not browsers, and downloads just the information in the HEAD tag information. The specific level of HTTP protocol is also recorded.

Status Code :

200

There are four classes of codes

1. Success (200 series)
2. Redirect (300 series)
3. Failure (400 series)
4. Server Error (500 series)

A status code of 200 means the transaction was successful. Common 300-series codes are 302, for a redirect from `http://www.mydomain.com` to `http://www.mydomain.com/`, and 304 for a conditional GET. This occurs when the server checks if the version of the file or graphic already in cache is still the current version and directs the browser to use the cached version. The most common failure codes are 401 (failed authentication), 403 (forbidden request to a restricted subdirectory), and the dreaded 404 (file not found) messages. Server errors are red flags for the server administrator.

Transfer Volume :

1749

For GET HTTP transactions, the last field is the number of bytes transferred. For other commands this field will be a hyphen (-) or a zero (0).

The transfer volume statistic marks the end of the common log file. The remaining fields make up the referrer and agent logs, added to the common log format to create the “extended” log file format. Let’s look at these fields.

Referrer URL :

`http://www.cs.bilkent.edu.tr/guvenir`

The referrer URL indicates the page where the visitor was located when making the next request.

User Agent

:

Mozilla/4.0 (compatible; MSIE 5.5; Windows 95)

The user agent stores information about the browser, version, and operating system of the reader. The general format is: Browser name/ version (operating system)

3.1.1.1 Elimination of irrelevant items

Two terms will be described which are mostly used in web usage mining before going into detail. “Valid File Request” describes any type of data including graphics, scripts or HTML pages requested from the web server whereas “Valid Page Request” describes any successfully answered request for one of the actual web pages taking place in the web site in process. Different objects are embedded into the HTML pages such as text, pictures, sounds etc. Therefore, a user’s request to view a particular page often results in several log entries since graphics and sounds are downloaded in addition to the HTML file. The discovered associations or statistics are only useful if the data represented in the log files gives an accurate picture of the user accesses to the web site. In most web usage applications, only the log entries of the HTML pages are considered as relevant and the others are considered as irrelevant. This is because, in general, a user does not explicitly request all of the graphics that are on a web page, they are automatically downloaded due to HTML tags.

Also, especially index pages usually redirect all visitors automatically to a script; e.g., count.cgi, to count the number of visitors. As a result, for each redirection from these index files to the script, an entry is put into the log file. So, a technique must be applied onto the access log file to eliminate these irrelevant items for any type of analysis.

Elimination of these items considered as irrelevant can be reasonably accomplished by checking the suffix of the URL name in the “Target” field of the entry. For instance, all log entries with file extension jpg, gif, wav, class, au, cgi are removed for the accurate determination of the user and session identification.

Besides, we have one more factor to be taken into consideration. Sometimes, in case of having a problem, the web server cannot be able to give successful respond to the requests. The web server records these actions in the access log file by putting an error code into the “Status Code” field of the entry. As a result, these unsuccessful requests must be eliminated from the log file before mining. These entries can be determined only by checking the status code of the entry. For example, status code “400” or ”404” means that the page could not be found on the site by the server due to the deletion of the pages and so, in general, the entries with the status code “400” or “404” are eliminated.

3.1.1.2 Elimination of entries containing frame pages

Another data preparation process done in this phase is the detection and if necessary the elimination or modification of the entries that contain frame pages. After eliminating the irrelevant entries from access log file, the Log Analyzer module analyze each web page in the web site whether it has frame pages or not. The more detailed information about the determination of frame pages will be given in 3.1.3. Such an entry has been given as an example in Figure 3.2

As shown in Figure 3.2, the web page “/~canf/CS351/” consists of two frames, frame1.htm and frame2.htm. When a request is made for the page mentioned above, three entries are put into the access log file automatically, one entry for the page itself (Entry 1), two entries for the frame pages (Entry 2 and 3) that form the page.

```
[1] labb30640.bcc.bilkent.edu.tr - - [01/Nov/2001:22:07:13 +0200]
"GET      /~canf/CS351/ HTTP/1.1" 200 669
"http://www.cs.bilkent.edu.tr/~endemir/courses/cs35101/cs35101.html"
      "Mozilla/4.0 (compatible; MSIE 5.5; Windows 95)"

[2] labb30640.bcc.bilkent.edu.tr - - [01/Nov/2001:22:07:13 +0200]
"GET      /~canf/CS351/frame1.htm HTTP/1.1" 200 2778
"http://www.cs.bilkent.edu.tr/~canf/CS351/" "Mozilla/4.0
(compatible; MSIE 5.5; Windows 95)"

[3] labb30640.bcc.bilkent.edu.tr - - [01/Nov/2001:22:07:13 +0200]
"GET      /~canf/CS351/frame2.htm HTTP/1.1" 200 13306
"http://www.cs.bilkent.edu.tr/~canf/CS351/" "Mozilla/4.0
(compatible; MSIE 5.5; Windows 95)"

[4] labb30640.bcc.bilkent.edu.tr - - [01/Nov/2001:22:07:23 +0200]
"GET      /~canf/CS351/CS351LectureNotes/index.html HTTP/1.1" 200
1230 "http://www.cs.bilkent.edu.tr/~canf/CS351/frame1.htm"
"Mozilla/4.0 (compatible; MSIE 5.5; Windows 95)"
```

Figure 3.2: A series of entry with frame pages

Entries containing the frame pages were irrelevant items such as images or sounds embedded into the page and must be eliminated or modified before the user and session identification. At the same time, when a user requests a page from any page consisting frame pages, then the “Referrer” field of the active log entry seems as shown in the fourth entry of the figure. So, the module exchanges the “Referrer” field with the name of the main page. After modification, the fourth entry becomes like

```
labb30640.bcc.bilkent.edu.tr - - [01/Nov/2001:22:07:23 +0200] "GET
/~canf/CS351/CS351LectureNotes/index.html HTTP/1.1" 200 1230
"http://www.cs.bilkent.edu.tr/~canf/CS351" "Mozilla/4.0 (compatible;
MSIE 5.5; Windows 95)"
```

Figure 3.3: The entry after modification

In addition, “Referrer” field of some entries should be modified. If a visitor begins her/his visit to the site by the help of a search engine, an entry is put into the log file as shown in Figure 3.4. The Referrer field of this entry must be modified since we are not interested with the query search words coming from different search

engines. In these situations, we assume that the user begins her/his visit from the page written in the “Target” field of the entry and we exchange “Referrer” field with “-“ sign meaning that the full path of the page requested has been typed directly in the address field of the browsers. After modification, the entry becomes as shown in Figure 3.5.

```
client-209-158-171-2.jerseycity.k12.nj.us - - [01/Nov/2001:21:47:10
+0200] "GET /~david/derya/activities1/activity70.htm HTTP/1.0"
200 82389 http://google.yahoo.com/bin/query?p=puffy+aand+r&hc=0&hs="
Mozilla/4.0 (compatible; MSIE 5.0; Windows 95; DigExt)"
```

Figure 3.4: An example entry with search engines in Referrer field

```
client-209-158-171-2.jerseycity.k12.nj.us - - [01/Nov/2001:21:47:10
+0200] "GET /~david/derya/activities1/activity70.htm HTTP/1.0" 200
82389 "-" "Mozilla/4.0 (compatible; MSIE 5.0; Windows 95; DigExt)"
```

Figure 3.5: The same entry after modification

3.1.2 Determination of Frame Pages

It is a common way to create an HTML page with frame pages to make more attractive and more helpful for the users visiting the page. A web page may be constructed with three frames, top frame for general site navigation, left frame for more specific navigation and a main frame with some content. After eliminating the irrelevant items such as embedded pictures or sounds from the log file, the next aim of the module is to analyze all valid page requests to discover whether it contains frame pages or not. This is the most time consuming process of the preprocessing module. If we do not store these information gathered by the “Frame_Detector” algorithm shown in Figure 3.6, the module must go to all pages of the web site at every execution. So, it is acceptable to store these information gathered in a file called as “Frame File” to avoid from analyzing every pages at every execution. The module opens the Frame File updated one day before at every execution; it loads all frame page structures in the file into the main memory as a list to be used for

eliminating the entries containing frame pages from the log file. When the module executes, it checks whether the web page is in the list or not. If it exists in the list, it means that this web page was analyzed before by the module, so the module do not analyze this page again and continues with the next page and so on. At the first runs of the module, the time spent for Frame Detection is quite long, but after every execution of the module in the following days, Frame File is updated with new pages and contains more information, so the time spent for this process in later runs decreases.

```
[1]   Open web page requested
[2]   For each line in the file Do
[3]     If line contains "<frameset" tag
[4]       While (entry contains "</frameset" tag)
[5]         If line contains "src=" tag
[6]           Store the name of frame page written after the "src" argument
           into the list
[7]         End If
[8]       End While
[9]     End If
[10]  End For
```

Figure 3.6: Algorithm Frame_Detector

It can be easily detected whether an HTML page has any frame page or not only by checking the content of the file. The frame pages are inserted between "<frameset>" and "</frameset>" tags in an HTML page. (Lines 3-4) The path and full name of these frame pages comes after "src=" argument. (Lines 5-7) The algorithm opens the HTML file that is represented as a valid page request for the web server and checks all lines until the end of the file. If the algorithm encounters "<frameset>" tag, it stores all frame pages coming just after "src=" tag to the list until the line containing "</frameset>" tag or the end of the file.

Assuming that the web master may sometimes decide to make a change on the pages in the site, we decided to delete all information in the Frame File on the first day of the month. So the changes that have been made on the pages in the present month will be detected at the execution of the module on the first day of the

next month. It means that the module analyzes all pages again in the site every month to detect the changes in the page.

3.1.3 Session Identification

A user session is a sequence of all of the page references made by a user during a single visit to a site. A transaction differs from a user session in that the size of a transaction can range from a single page reference to all of the page references in a user session. The raw server access log can be thought of in two ways; either as a single transaction of many page references or as set of many transactions each consisting of a single page reference. The goal of session and transaction identification is to create meaningful clusters of references for each user. Therefore, we divide all of the log file transaction into smaller ones and then merge them into fewer larger ones for each identity.

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 FQDN
- $l.time$ is the time of the request
- $l.target$ is the requested URL
- $l.referrer$ is the referrer page used for accessing the target page
- $l.agent$ is the browser name used by the user

This information can be used to reconstruct the user navigation sessions within the site that the log data originates. In an ideal scenario, each user is allocated a unique IP address whenever (s) he accesses a given web site. Moreover, it is expected that a user visit the site more than once, each time possibly with a different goal in mind. Therefore, a user session is usually defined as a sequence of requests from the same IP address such that no two consecutive requests are separated by

more than X minutes where X is a given parameter. In [41], the authors report an experiment conducted with a web browser that was modified in order to record, among other things, the time interval between user actions on the browser's interface. One interesting result of the study revealed that 25.5 minutes corresponded to 1.5 standard deviation of the average time between user actions, meaning that the probability of a user staying more than 25.5 minutes without placing any page request is very low. As a result of the study, many authors and also we have adopted the value of 30 minutes for the time limit between requests within a session, i.e. $X=30$ minutes. In the light of this observation, a user session can be defined as shown in Definition 2.

Definition 2: User session S is defined as $S = \langle \text{ident}_S, PR_S \rangle$ where

- $PR_S = \{(R^S_1.\text{referrer}, R^S_1.\text{target}), (R^S_2.\text{referrer}, R^S_2.\text{target}) \dots (R^S_m.\text{referrer}, R^S_m.\text{target})\}$
- $R^S_k \in L$ and $R^S_k.\text{ident} = S.\text{ident}$ for $0 < k \leq m$
- $R^S_m.\text{time} - R^S_{m-1}.\text{time} < 30$

Another relevant aspect to take into account when using log data is the widespread use of cache and proxy servers on the web. As a result, not all page requests made to a server are recorded in the log file. In fact, if the browser finds a copy of a document being requested by the user in its cache, the request is not made to the server and the stored copy of the document is displayed. Therefore, although the user views the page, the request is not recorded in the server log file. A similar thing can occur at proxy level. A proxy server can be configured in such way that, a copy of a requested page is not available in the local memory, the page is requested by the proxy to the content provider on behalf of the user. In addition, the use of proxy servers that will be discussed later raises difficulties in the identification of the requests made by a given computer. A sample session can be seen in Figure 3.7.

<u>Target</u>	<u>Referrer</u>
1. /~gudukbay/home.html	--
2. /~gudukbay/cs565/index.html	/~gudukbay/home.html
3. /~gudukbay/cs565/project_list/project.html	/~gudukbay/cs565/index.html
4. /cs466/index.html	/~gudukbay/home.html

Figure 3.7: A sample user session

As shown in the figure, the user begins the session just by typing the exact path of the page “/~gudukbay/home.html” directly in the address line of the browser since the Referrer field is represented as “-“. Then he goes to the page “gudukbay/cs565/index.html, and then “/~gudukbay/cs565/project_list/project.htm” consecutively. Each of these two pages is retrieved by following the links on the previously retrieved pages. We draw this conclusion just by looking at the Referrer field of the corresponding accesses. But the Referrer field of the fourth request made by the user shows us that the user has clicked two times on BACK button of the browser and returned back to the page “/~gudukbay/home.html”. Then the user retrieves the page “/cs466/index.html”. So the navigation of the user through the web site by clicking the BACK button could not be detected by the server and recorded in the server access log. Also, we may encounter with the common problems below due to the proxy servers.

- *Single IP address/Multiple Server Sessions:* Internet Service Providers (ISPs) typically have a pool of proxy servers that users access the web through. A single proxy server may have several users accessing a web site, potentially over the same time period.
- *Multiple IP address/Single Server Session:* Some ISPs or privacy tools randomly assign each request from a user to one of several IP addresses. In this case, a single server session has multiple IP addresses.

- *Multiple IP address/Single User:* A user that accesses the web from different machines will have a different IP addresses from session to session. This makes tracking repeat visits from the same user difficult.
- *Multiple Agent/Single User:* Again, a user that uses more than one browser, even on the same machine, will appear as multiple users.

In such cases, the IP address recorded in the log file corresponds to the proxy and not to the user. Note that more than one user can be using the same proxy to browse the same site at the same time. So, the unique users must be determined before applying data mining techniques. There are some techniques available used for determining the unique users. One of these techniques is the use of cookies to track an individual user within a site. If cookies are enabled, when a new user requests a document, the response includes a unique user identifier, which the browser stores in the user's hard disk. All subsequent requests made by the browser to that same site will include the cookie information and therefore, allow the service provider to recognize the user. However, the use of cookies is only possible with the user's consent and its use has raised privacy concerns.

Another way to identify the unique users and also be used in our web usage mining system is to use "Agent" field of the entry. Even if the IP address is the same, if the Agent field of the entry shows a change in browser software or operating system, a reasonable assumption to make is that each different agent type for an IP address represents a different user.

By taking all criterions discussed above into account, we derived an algorithm shown in Figure 3.8 to determine the sessions embedded in the log file. A linked list model is used to hold all user sessions in the given log file. For the first session in the access log file, a root node is created and all remaining sessions are added to the list.

```

[1] For each entry in the given log file
[2]   If the entry is valid
[3]     Assign Identity, Target, Referrer, Time and Agent information
[4]     For each Identity and Agent pair do
[5]       Search "Session List" for the given "Identity and Agent" pair
[6]       If an open session belonging to the given "Identity and Agent" pair is
         not found
[7]         Create a new open session
[8]         Owner of the session ← (Identity and Agent)
[9]         Starting time of the session ← Time
[10]        First access of the session ← (Referrer and Target)
[11]      End If
[12]    Else
[13]      If Referrer is "-"
[14]        Close the open session
[15]        Create a new open session and load all data to the new session
[16]      End If
[17]    Else
[18]      If Referrer is in the Page List of the session
[19]        If (Time- Starting Time of the Session) > 30
[20]          Close the open session
[21]          Create a new open session and load all data to
            the new session
[22]        End If
[23]      Else
[24]        Add Target into List Page of the session
[25]        num_of_pages ++;
[26]      End Else
[27]    End If
[28]  Else
[29]    Close the open session
[30]    Create a new open session and load all data to the new
        session
[31]  End Else
[32] End For
[33] End If
[34] End For

```

Figure 3.8: Algorithm used for session identification

Each node in the list generally consists of four fields which are the owner of the session (Identity and Agent fields), the number of the pages in the session (number_of_pages), the pages accessed in the session (Page_List) and a flag representing whether the session is open or not. Also an extra field for each page in

the session (`time_spent`) which is especially used by Recommendation Engine is kept.

Firstly, the entry is processed to obtain whether it consists of irrelevant items (Line 2) or not. It is eliminated if it is not a valid page request; otherwise it is used as an input to the algorithm.

Since the users can be uniquely identified by using the “Agent” field of the entry, the owner of the session is indicated with “Identity and Agent” pair for each node in the list. The process of session identification begins with the searching “Identity and Agent” pair in the available session list whether there is an open session or not belonging to the mentioned “Identity and Agent” pair. The most important factor here is to find an *open* session belonging to the visitor. Here, open session means that it has not been finished yet and still continues. While the session list is updated with the new entries, there may be some sessions belonging to the same visitor that have been finished. If an open session belonging to the “Identity and Agent” pair is not found, then a new session node is created and linked to the last node (Lines 6-11). An example is given with a fragment of session list as depicted in Figure 3.9 to make understanding easier. The first box of the session nodes shown in the figure represents the owner of the session while the second one is for “flag” which is used for determining whether the session is open or not; “0” for open sessions, “1” for closed sessions. Assuming that a page whose owner is “A” will be inserted into session list, firstly session list is searched for a node whose owner is “A” with flag “0”. The session list has two nodes belonging to “A”, but their flag is “1” meaning that they have been finished before. In that case, a new node belonging to “A” is created and linked just after the last node whose owner is “A”.

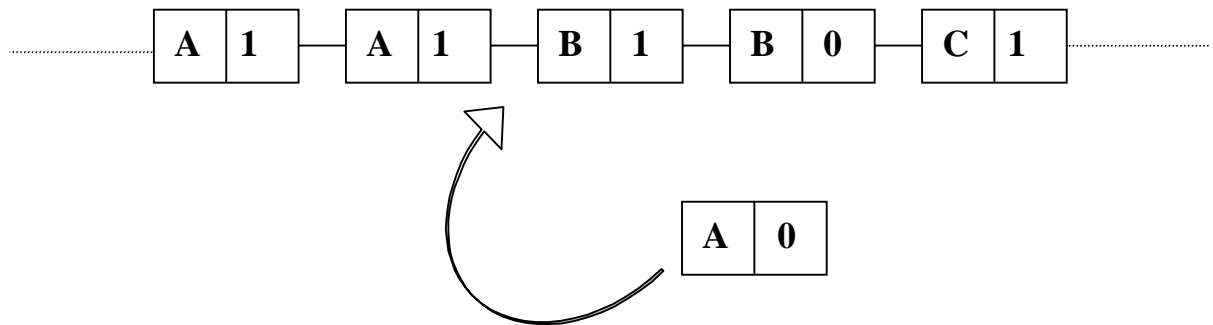


Figure 3.9: An example illustrating the creation of a new session node

At this point, there are two contingencies related to the number of pages that must be added to the Page List of the session. The Referrer field used for accessing to any page in the site may be empty or has any page. As discussed above, if the Referrer field is empty, it means that the visitor types the full path of the page directly into the address box of the browser and it is also acceptable in every step of the algorithm to finish an open session and create a new open session. In these cases, a new session node is created and the Target Page is added into the Page List of the session and the `number_of_pages`, is set to 1. As a result, the session node has its owner, one page in the Page List and the `number_of_pages`, is equal to the 1.

Otherwise, if the Target Page is accessed by using the cache copy of the page in the Referrer field at any time, a new session is created. He firstly requests the page in the Referrer field, but after retrieving a number of pages, he comes back to the page in the Referrer field. Because the pages used to get back exist in the cache, the web server cannot be aware of these requests, so this transactions do not include in the log file. This situation must be taken into consideration to eliminate the disadvantage of using cache pages, so we put the page in the Referrer field as the starting page and then the page in Target field is added into the Page List. Because we have now two pages in the Page List in the session, then the `number_of_pages` is set to 2.

Because our main idea is to recommend the pages, which have been spent more time than the others to the user, we also load the time information of the page requests into the session node. The time spent on the page is calculated by just subtracting the access time of the active page from that of the page one before and put it into the “time_spent” field of the page requested.

If any open session belonging to the “Identity and Agent” pair is found in the Session list, then the first step is to check the Referrer field of the entry whether it has any page or empty (Lines 13-16). As explained above, if the Referrer field is empty, we assume that a new user session begins. In these situations, the session found is closed by changing the flag to “1” and a new session node is created for the visitor and added to the node closed just before. After creating the new session, all work explained in case of having no open session in the session list for any visitor is done for the active session node. If the Referrer field of the entry is not empty and contains a page, then we check the page in the Referrer field whether it exists in the Page List of the session or not (Lines 23-26). That the page in the Referrer field is the same as the last page in the Page List of the session means that no backtracking is made and the page in the Target field is added to the Page List. After adding the page to Page List of the session, the time_spent of the previous page in the Page List is calculated and the number_of_ pages is incremented by 1. If the page in the Referrer field is not the same as the last page in the Page List of the session, 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. At this point, what is known is the page in the Referrer field for the current access. So, the action should be taken is to perform a backward search in the Page List of the session. Whenever a page is found, the search operation terminates. We have discussed that the time threshold for a session is approximately 30 minutes. If the difference between the time of the first page of the Page List and the time of the active page is more than 30 minutes, then that open session is closed and a new session node is created (Lines 19-22).

The last contingency in the session identification is that the page in the Referrer field is not in the Page List of the session. At this point, the session found is closed and a new session node is created (Lines 28-31).

At the end of processing all entries in the given log file by Log Analyzer module, an adjustment must be made on the sessions created before storing the session information into the Result File. One of the reasons for this adjustment is that the time spent value of the last pages in the Page List of the sessions can not be calculated because the session terminates in that page (the visitor exits the site in that page) and we can not have any chance to know how much time the visitor spent on that page before leaving the site. One assumption may be that the user has found what she/he wants on the last page and then left the site. But the other assumption may be that the visitor has tried to find what she/he wants, but he could not have achieved to reach her/his goal and he may have accepted to leave the site at the last page of the session. So, by taking into the consideration the latter assumption and because we also have no idea about the time spent on the last page, we decided to discard the last page accesses from the sessions before storing.

Another reason is that some sessions may not hold enough information about the user navigation. Some sessions sometimes hold at most one page in their Page List meaning that the user starts surfing in a particular page and leaves the site without going further pages.

The other reason is that the difference between the starting time of some open sessions and the time of the last entry in the log file processed may exceed the time threshold assigned for the session. So at the end of the execution of the session identification module, the open sessions exceeding the threshold must be closed by the adjustment procedure before storing and indexing. The elimination of the sessions is done by the `Eliminate_Session` algorithm shown in Figure 3.10

```

[1] For each session in the Session List
[2]   If Flag=1
[3]     If num_of_pages=1
[4]       control←TRUE
[5]       Discard the page in the Page List
[6]     End If
[7]     If num_of_pages=2
[8]       control←TRUE
[9]       Discard both pages in the Page List
[10]    End If
[11]    If num_of_pages > 2
[12]      Discard the last page in the Page List
[13]      Decrement the num_of_pages by 1
[14]    End If
[15]  End If
[16]  Else (If Flag=0)
[17]    If time threshold is exceeded
[18]      Process the Lines (3-10) for each session that satisfies the IF
        condition
[19]      If num_of_pages >2
[20]        Discard the last page in the Page List
[21]        Decrement the num_of_pages by 1
[22]        Flag←1
[23]      End If
[24]    End If
[25]  If control=TRUE
[26]    Delete the session node
[27]  End If
[28] End For

```

Figure 3.10: Algorithm Eliminate_Session

In this algorithm, all session nodes in the session list are passed one by one to control whether it satisfies all conditions necessary for session identification or not. We have discussed that if the flag of a session node is 0 then it means that the session is still open. If the flag of the session node processed is 1 (closed session), we check the number of pages in the Page List .If the number of pages is less than two (Lines 3-10), we discard these pages from the Page List and we assign TRUE to the variable “control” which will be used for deleting the session node if necessary. If the number

of pages is more than two pages (Lines 11-14), we discard the last page from the Page List and decrement the value of `num_of_pages` by 1.

If the session checked is still open, we control whether it exceeds the time threshold or not. In the case of exceeding the time threshold, we apply the process in the lines (3-10) for the session node. But if the number of pages in the Page List is more than two pages, we just discard the last page in the Page List of the session, decrement the value of `num_of_pages` by 1 and then we close the session by changing flag variable to 1. At the end of the algorithm if “control” variable of the session node is assigned to TRUE, we delete that session node from the session list because it does not anymore hold enough information for the session identification.

There is an important point that must be discussed here. Some sessions still continue however we have processed all entries of the given log file. We open access log file to be processed for a while to read the entries until the end of the file, but the web server still continues to put entries into the log file due to the requests. We are not aware of these requests during the execution of the module and this request information may be related to the sessions that are not completed. So we must take this fact into account. We store all finished sessions into the Result file at the end of the session identification. The remaining unfinished sessions show us that they are not completed and must be kept in somewhere to load them into the session list just before processing the entries of the access log file of the next day.

So, Log Analyzer module is designed to keep all unfinished sessions in a file called as Session File. After storing all terminated session information into the Result file, we store the remaining unfinished sessions into the file with all information about the sessions, the owner of the session, the number of pages in the session and the pages in the session. If Session File contains one or more unfinished session information belong to one day before, then these sessions are loaded into the session list just before the processing of the entries in the log file at every execution of the Log Analyzer module.

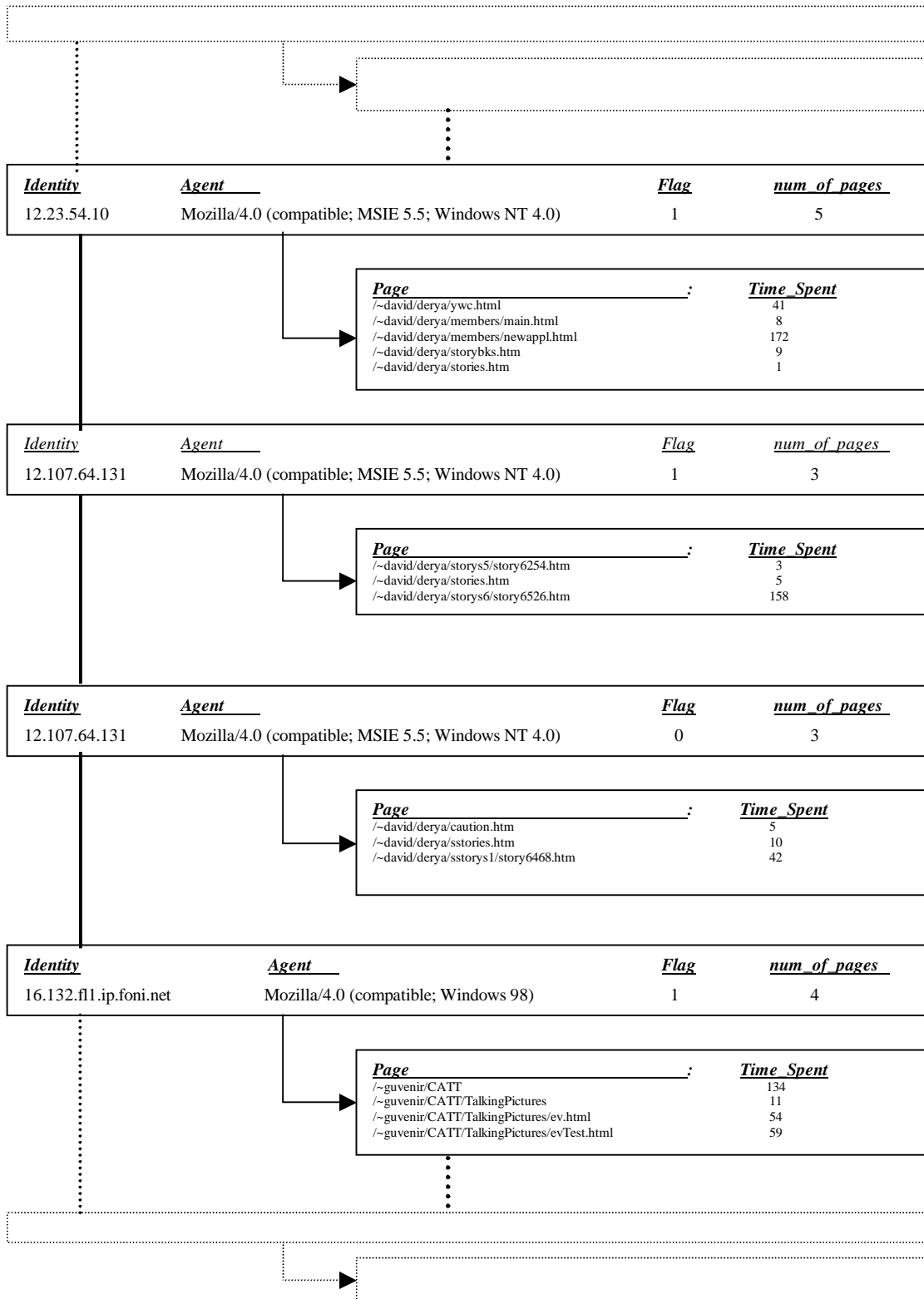


Figure 3.11: A fragment of the sessions created

The flag variable of the sessions loaded from the Session File is set to 0 meaning they are still open. In other words, the unfinished sessions are stored into the Session File at the end of every execution of the Log Analyzer module and the unfinished sessions generated by the module one day before are loaded at every start of execution of the module into the session list. A small fragment of sessions generated by the module is shown in Figure 3.11.

3.1.4 Classifying Identity Information

As described above, the first field of the entry of the access log file is either an IP address or a FQDN of the visitor who requests a page from the web server. The details of the identity information will be discussed in the next section.

3.1.4.1 IP Addresses

In an IP network, each computer is allocated a unique IP address. In the current version of IP protocol, IP version 4, an IP address is 4 bytes. The addresses are usually written as $x1.x2.x3.x4$, with $x1$, $x2$, $x3$ and $x4$ each describing one byte of the address. For example, address 16843009 (hex 1010101) is written as 1.1.1.1, since each byte of this address has a value of 1.

Since an address is 4 bytes, the total number of available addresses is $2^{32} = 4,294,967,296$. This represents the TOTAL theoretical number of computers that can be directly connected to the Internet. In practice, the real limit is much smaller for several reasons.

Each physical network has to have a unique Network Number, comprising some of the bits of the IP address. The rest of the bits are used as a Host Number to uniquely identify each computer on that network. The number of unique Network Numbers that can be assigned in the Internet is therefore much smaller than 4 billion,

and it is very unlikely that all of the possible Host Numbers in each Network Number are fully assigned.

An address is divided into two parts: a network number and a host number. The idea is that all computers on one physical network will have the same network number - a bit like the street name, the rest of the address defines an individual computer - a bit like house numbers within a street. The size of the network and host parts depends on the class of the address, and is determined by address' network mask. The network mask is a binary mask with 1s in the network part of the address, and 0 in the host part.

To allow for a range from big networks, with a lot of computers, to small networks, with a few hosts, the IP address space is divided into 4 classes, called class A, B, C and D. The first byte of the address determines which class an address belongs to:

- Network addresses with first byte between 1 and 126 are class A, and can have about 17 million hosts each.
- Network addresses with first byte between 128 and 191 are class B, and can have about 65000 hosts each.
- Network addresses with first byte between 192 and 223 are class C, and can have 256 hosts.
- All other networks are class D, used for special functions or class E that is reserved.

Most class A and B addresses have already been allocated, leaving only class C available. This means that total number of available addresses on the Internet is 2,147,483,774. Each major world region has an authority that is given a share of the addresses and is responsible for allocating them to Internet Service Providers (ISPs)

and other large customers. Because of routing requirements, a whole class C network (256 addresses) has to be assigned to a client at a time; the clients (e.g., ISPs) are then responsible for distributing these addresses to their customers.

While the number of available addresses seems large, the Internet is growing at such a pace that it will soon be exhausted. While the next generation IP protocol, IP version 6, allows for larger addresses, it will take years before the existing network infrastructure migrates to the new protocol.

Because IP addresses are a scarce resource, most Internet Service Providers (ISPs) will only allocate one address to a single customer. In majority of cases this address is assigned dynamically, so every time a client connects to the ISP a different address will be provided dynamically. Big companies can buy more addresses, but for small businesses and home users the cost of doing so is prohibitive. Because such users are given only one IP address, they can have only one computer connected to the Internet at one time. With a Network Address Translator (NAT) gateway running on this single computer, it is possible to share that single address between multiple local computers and connect them all at the same time. The outside world is unaware of this division and thinks that only one computer is connected.

3.1.4.2. Fully Qualified Domain Names

A FQDN is that portion of an Internet Uniform Resource Locator (**URL**) that fully identifies the **server** program that an Internet request is addressed to. The FQDN includes the second-level **domain name** (such as "cs.bilkent.edu.tr") and any other levels (for example, "www.cs.bilkent.edu.tr" or "www.bilkent.edu.tr"). The prefix "http://" added to the Fully Qualified Domain Name completes the URL.

DNS (Domain Name Server) is an Internet service that translates the name into the corresponding IP address. For example, the domain name

“*patara.cs.bilkent.edu.tr*” might be translated to *139.179.21.122*. The DNS system is, in fact, its own network.

When a page is requested from web server, web server asks the DNS server belong to the network whether it has information about the client’s IP address. If DNS has information about the IP address, it translates the IP address to its FQDN and returns it to the server. If the DNS server does not know how to translate a particular domain name, it asks it to the upper DNS, until the correct IP address is returned. The web server puts the FQDN returned from DNS to the “Address or DNS” field of the entry. If any information returns from the DNS servers belong to the network, then the web server puts just its IP address to the entry.

FQDNs should have at least two fields. Different from IP addresses, the 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”. In that case, each computer is associated with a domain name according to the predefined hierarchy built on the domain names called as Domain Name Hierarchy.

Topmost level of the Domain Name Hierarchy is the root domain and the nodes below the root contain the top-level domains, which are listed below. All computers connected to the Internet belongs one of these top-level domains.

- com: companies (<http://www.amazon.com>)
- edu: universities (<http://bilkent.edu.tr>)
- mil: military organizations (<http://www.tsk.mil.tr>)
- gov: government organizations (<http://www.turkey.gov.tr>)
- net: Internet Service Providers (<http://www.php.net>)
- org: nonprofit organizations (<http://www.apache.org>)
- also an extra field is used to differentiate the countries such as tr, au, uk, de etc.

Each top-level domain in the Domain Name Hierarchy has lower domains. Domain Name Hierarchy can also be represented by a hierarchical tree structure where each node of the tree represents a domain name. The nodes except the leaves of the tree contain the name of general domain. All computers connected to the Internet reside at the leaves in that tree structure. The FQDN can be acquired by following the path belong to the computer from bottom to top and by concatenating the values of the nodes on the path. The number of computers connected to a general domain is more than the number of the computers connected to the lower domain of itself. A small portion of this tree structure can be depicted in Figure 3.12.

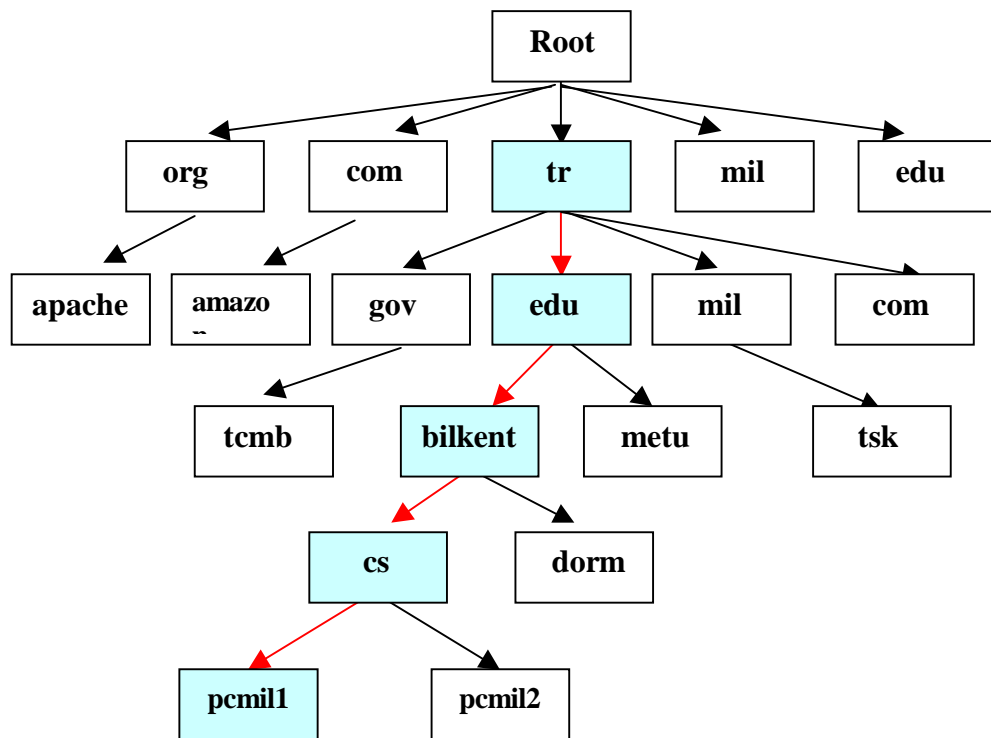


Figure 3.12: Domain Name Hierarchy

The top-level domains are assigned by the Internet Corporation for Assigned Names and Numbers (ICANN). Their administrators manage the domains below the top-level.

That the lowest domain containing the values “pcmil1” or “pcmil2” seems at the first domain identifier of the FQDN (pcmil2.cs.bilkent.edu.tr) is important for our system and will be used as an indicator of determining the parent domain.

3.1.5 Inserting Identity Information into the tree

One of the aim of the thesis is to recognize the visitor by using its IP address or its FQDN and recommend them the pages they are really interested based on their past experiences. We always want to get the identity information of the visitors in FQDN to classify them more specifically, but it is nearly impossible to get the identity information always in their FQDN

A tree structure is constructed to hold the information about the visitors. The aim of holding the identity information on a tree structure is to make indexing and create an index file which will be used by the Recommendation-Engine to find the start and the end position of the sessions belong to the visitor in the Result file. At the same time, if the system has no idea about the visitor, then tree structure finds the parent domain of the visitor to make appropriate recommendations without doing any extra process. Some identities either in IP addresses or in FQDN are given in Figure 3.13, to be used as an example to show how the tree is constructed.

As depicted in the figure, four of the identities (a, g, h, i) are FQDN whereas the others (b, c, d, e, f, j, k, l) are IP addresses. Each part of the identity will be represented by “domain identifier”. For example, when the identity “12.151.162.61” is parsed, four domain identifiers are generated such as 12, 151, 162 and 61. Because IP addresses are formed by four octets, they always have four domain identifiers. Domain identifier “12” is in the first domain level and also will be used for determining the class of the identity, domain identifier “151” is in the second level,

domain identifier “162” is in the third domain level and domain identifier “61” is in the fourth domain level.

- | | | | |
|----|-----------------------------|----|-------------------|
| a. | gregory.excite.com | b. | 12.151.162.61 |
| c. | 217.131.128.73 | d. | 217.131.133.71 |
| e. | 190.175.140.228 | f. | 190.175.136.238 |
| g. | b204d3.dorm.bilkent.edu.tr | h. | j3016.inktomi.com |
| i. | lab30640.bcc.bilkent.edu.tr | j. | 190.178.112.51 |
| k. | 217.131.143.21 | l. | 12.151.162.91 |

Figure 3.13 A series of identities

The first octet of the identity with IP addresses represents the topmost domain, but the topmost domain is the last domain identifier of the FQDNs. The insertion of the domain identifiers into the tree is made from the topmost domain to lowest level domain. So, the parsing of FQDNs is done backward. The most important factor affecting the height of the tree is the number of domain identifiers in FQDNs. For instance, when the identity “gregory.excite.com” is parsed, three domain identifiers are generated such as com, excite and gregory respectively. But, when the identity “b204d3.dorm.bilkent.edu.tr” is parsed, five domain identifiers are generated such as “b204d3”, “dorm”, “bilkent”, “edu” and “tr” respectively. As discussed above, the topmost domain of the first example is “com” whereas that of the other is “tr”.

Firstly, the root and five nodes linked to the root node are created as shown in Figure 3.14. As described in 3.1.4.1, IP addresses can be categorized as Class A, Class B, Class C, Class D, Class E. Since Class E is assigned for future use, we eliminated Class E. To avoid a conflict in future, we designed our system to classify the computers belong to Class E into Class D. We used four nodes (Node 1, Node 2, Node 3, and Node 4) for identities in IP addresses and one node for the identities in FQDN (Node 5).

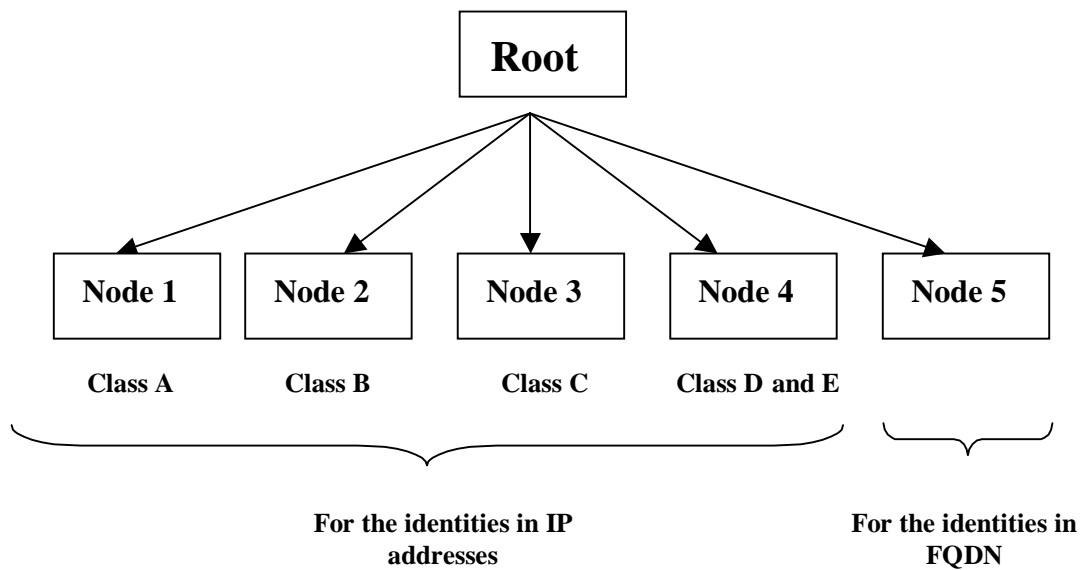


Figure 3.14 The root structure of the tree

All of the nodes in the tree have five fields.

Field 1: for the address of the node containing the next domain identifier of the identity

Field 2: for the address of the node containing the domain identifier of the other identities in the same level.

Field 3: to hold the start index that is an indicator of the beginning of sessions belong to computer/computers in the node.

Field 4: to hold the end index that is an indicator of the end of sessions belong to computer/computers in the node.

Field 5: the value of the domain identifier generated by parsing of the identity.

The algorithm, `Insert_Tree`, which is used to insert these identities into the tree, is shown in Figure 3.15. The first step is to determine whether the identity is an IP address (Line 2) or not. It can easily be detected only by checking each character in the identity information. IP addresses are composed of sequence of characters where each character is a digit between “0” and “9” whereas FQDNs may be composed of sequence of characters where each character is either a digit between “0” and “9” or a letter between “a” and “z”. If identity information has at least one alphabetic character, we understand that the identity information is a FQDN.

```

[1] For each unique identity in the given log file
[2]   Determine the identity whether it is FQDN or IP address
[3]   If identity is IP address
[4]     Parse the IP address forward to get domain identifiers
[5]     Determine the class of the identity by checking the first domain
        identifier of the IP address
[6]     Find the node at the root assigned for the class of the identity and
        make this node active
[7]   End If
[8]   Else
[9]     Parse the FQDN backward to get domain identifiers
[10]    Find the node at the root assigned for the identities with FQDN, that
        is, Node 5
[11]  End Else
[12]  For each domain identifier of the identity
[13]    Search domain identifier at the nodes attached to the active node
[14]    If node is not found
[15]      Create a new node for the domain identifier
[16]      Put the node in appropriate place satisfies the decreasing order
        of node values and make it active node
[17]    End If
[18]    Else
[19]      Make it active node
[20]    End Else
[21]  End For
[22] End For

```

Figure 3.15: Algorithm Insert_Tree

Then, we parse the identity information. The difference between the parsing IP address and FQDN is the order of the domain identifiers. As discussed above, the topmost general domain of the IP addresses seems at the first octet of the identity information whereas it seems at the last domain identifier of the FQDN. An extra work is not done for the IP addresses, but we reverse the identity information with FQDN before parsing and inserting them into the tree. For instance, identity information “190.175.140.228” is inserted into the tree as 190→ 175→ 140→ 228, but the identity information “b204d3.dorm.bilkent.edu.tr” is inserted into the tree as tr → edu → bilkent → dorm → b204d3, respectively. The domain identifiers found are then inserted to an array to be used in the insertion of the identifiers into the tree

If identity information is an IP address, the class it belongs is found by checking the value of first octet as shown in Figure 3.16. After detecting the class of the identity, the node where the domain identifiers of the identity will be inserted is found. If the identity information as a FQDN, then Node 5 is searched because Node 5 is assigned for the identities with FQDN.

<u>Value = First octet</u>	<u>Class</u>	<u>Node to be searched</u>
1 <= value <= 127	Class A	Node 1
128 < value <= 191	Class B	Node 2
192 < value <= 223	Class C	Node 3
224 < value <= 255	Class D	Node 4

Figure 3.16: Classification of the IP addresses

After the class node of the identity is found, then each domain identifier in the array is inserted into the tree. The value of first domain identifier is searched in the first level of the sub tree linked to one of the class node. If class node has no node, a new node is created by assigning the value of domain identifier to the node and is linked to the class node. Then the node created is assigned as “active”. If there exists some nodes, a search operation is made on these nodes to obtain whether the value of the domain identifier exists or not. If search operation is negative, then a new node is created by assigning the value of domain identifier to the node and is linked to appropriate node that satisfies the increasing order of the domain identifiers in the same level. Then the node created is made active. If search operation is positive, the node found is made active. The insertion of the remaining domain identifier is made by searching the value of domain identifier in the nodes linked to the active node respectively.

Insertion of each domain identifier into the tree is made in such a way that the search is completed in the least time. For the IP addresses, since each domain

identifier of the IP addresses is between 0-255, each node at the levels has 255 lower nodes connected, so the search operation on a level is made on 255 nodes. Insertion of all domain identifiers of the identity into the tree is done at most 4 (searching to find the Class Node) + 255 (searching to find the place in the first Level) + 255 (searching to find the place in the second Level) + 255 (searching to find the place in the third Level) + 255 (searching to find the place in the fourth Level) = 1024 in the worst case.

The number of nodes on each level changes dynamically due to the number of domain identifiers of the identities found at the end of parsing the identity. One of the identities in FQDN may have 3 domain identifiers as in “gregory.excite.com” or may have 5 domain identifiers in “b204d3.dorm.bilkent.edu.tr”. Generally, the time spent for the insertion of domain identifiers into the tree increases from top to bottom of the tree because the number of the domain identifiers at upper levels is less than the number of the domain identifiers at lower levels. As shown in Figure 3.11, the upper level of the sub tree assigned for identities with FQDN consist of the topmost domain values such as com, edu, gov etc.

A small portion of the tree constructed after the insertion of all identity information given in Figure 3.13 into the tree is shown in Figure 3.17. All we explained above can be seen in the figure. Node 1, Node2, Node 3, and Node 4 has each four level whereas the Node 5 has five levels. The depth of from the nodes belonging to the IP addresses is static, 4; the depth may be less or more than that from the node 5. It is fact that the depth of the whole tree depends on any identity information with FQDN having the maximum number of domain identifier.

3.1.6 Storing Session Information and Indexing

The next phase of Log Analyzer module is the process of storing the generated sessions and the tree structure holding the identity information in such a way to make the runtime of Recommendation Engine as fast as possible to find the pages which will be recommended to the user. Because our aim is to find the pages to be recommended in the least time without increasing the time of loading page into the client's machine. At the end of determination of the sessions and constructing the tree, we have two main structure used for storing and indexing. One of them is tree structure as shown in Figure 3.17, and the other is session list as shown in Figure 3.11.

There comes an important issue related to creating the Result File and updating the tree. We always have a Result File containing the sessions and an Index File containing the start and end index of the sessions in the Result File belonging to the visitors that have been created one day before. At every execution of the module, the tree is constructed just before the processing of the new entries in the access log file to have an ability to know the start and end positions of the sessions in the Result File belonging to the identities.

Because an identity may have no session in the day of the last execution of the module, but all sessions of the identity belonging to prior days in the Result file must be rewritten to the Result file. If the tree is not constructed each time, then some sessions belonging to an identity that has some sessions before but not on the day of the execution of the module may be loosed. We constructed the tree by the help of the Index File created one day before. The algorithm, Construct_Tree, used for this process is shown in Figure 3.18.

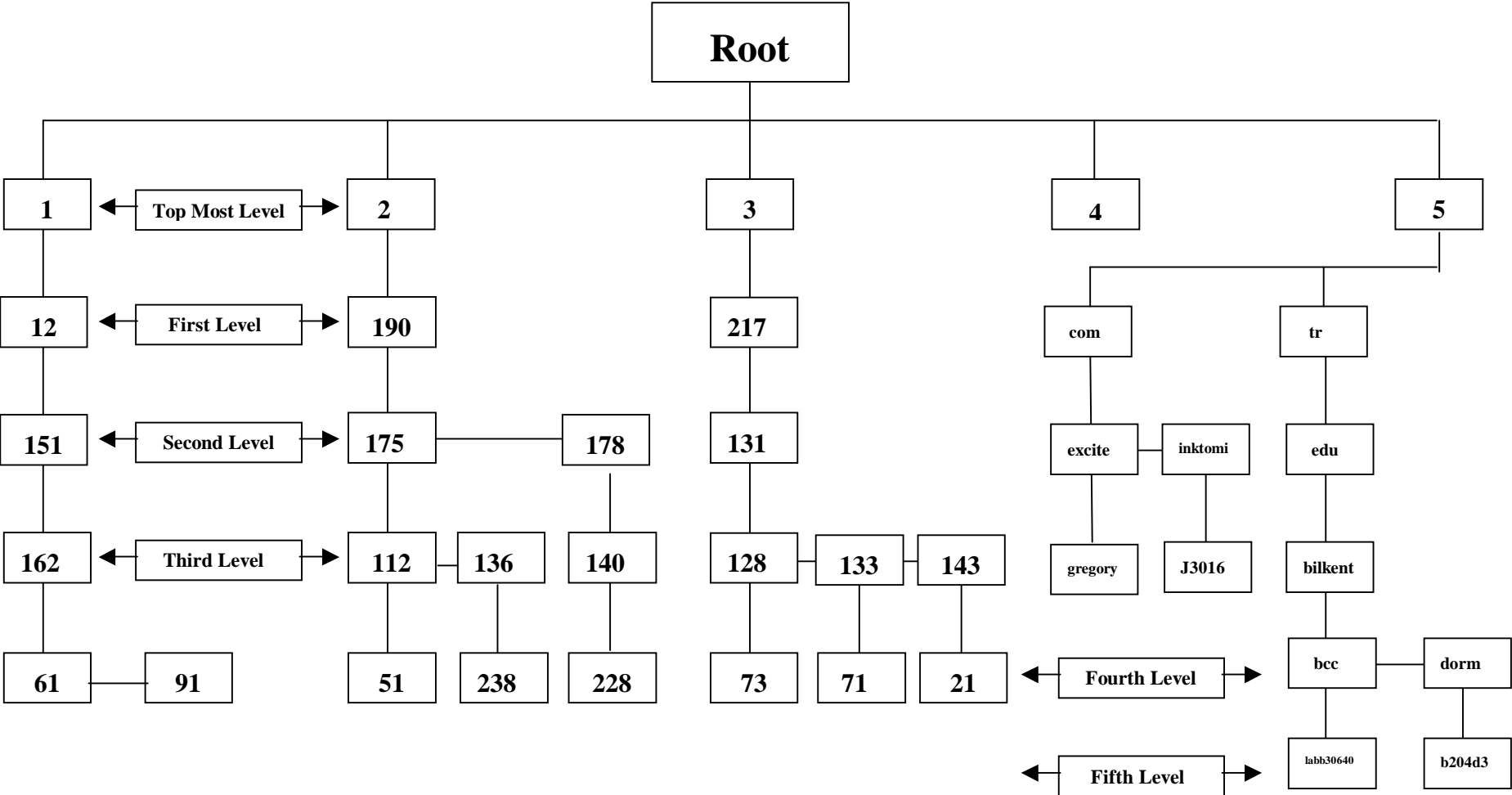


Figure 3.17: Tree structure holding the identity information

- [1] Open Index File
- [2] FOR each entry (line) of the file DO
 - [3] Parse the entry to obtain identity, start and end index
 - [4] Apply the algorithm in Figure 3.15 for the identity obtained
 - [5] Put the start and end index variables into the node holding the last domain identifier of the identity.
- [6] End FOR

Figure 3.18: Algorithm Construct_Tree

Firstly, Index File created one day before is opened for constructing the tree with the identities in the file. Each line of the file is processed to obtain the identity, the start index and the end index. After processing the entry, the identity is loaded into the tree as described in 3.1.5. For instance, assume that the first three lines of the file is as the same as in Table 3.1

Identity	Start Index	End Index
64.221.22.123	0	123321
66.223.142.27	123321	125623
66.21.22.121	125623	201237

Table 3.1: Example entries in the Index file

As shown in the table, the first entries in the file belong to the Class A that is assigned for the identities with the IP addresses due to the result of creating the index file, which will be explained later. After executing of the algorithm described above just for the first three entries in the file, the tree shown in Figure 3.19 is constructed.

As shown in the figure, the start and end index values of the identities representing the position of the sessions in the Result File belonging to the identity are placed in the start and end index field of the nodes holding the last domain identifiers of the identities which are also the leaf nodes of the tree.

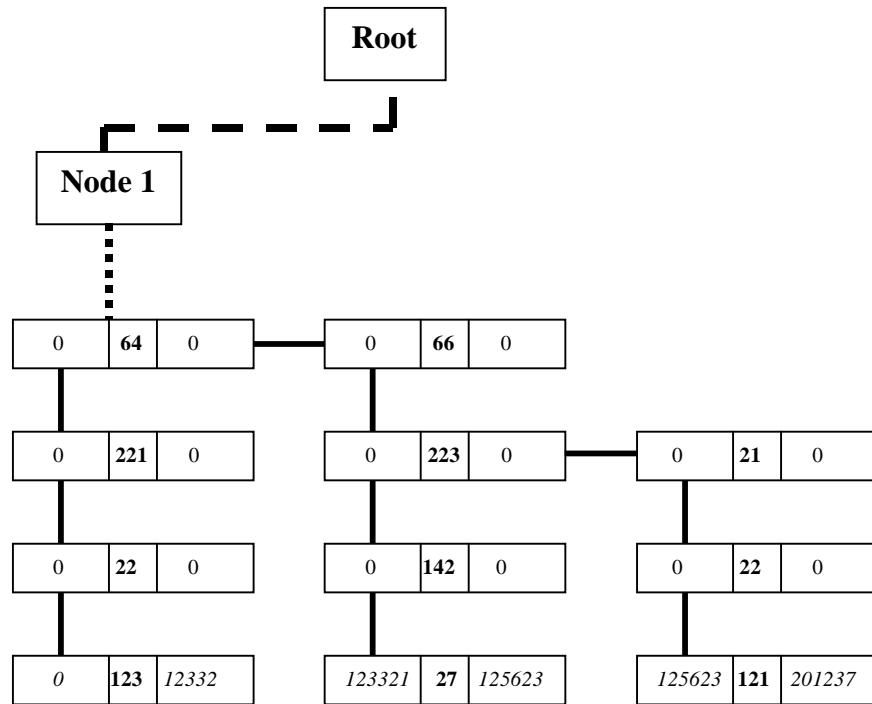


Figure 3.19: The tree constructed before the execution of the module

The “start index” and “end index” field of the intermediate nodes will be used by the Recommendation Engine of the system to find the start and end index values of the parent domain of the identity in the case of having no pages to recommend related to the past experiences of the visitor, so at that point the value of the start and end index fields is set to 0.

At the end of the execution of the algorithm, Construct_Tree, a tree is formed before analyzing of the access log file. The other identity information that comes across during the processing of the entries in the log file is loaded into the tree constructed and the tree is updated with these newly added identities.

After the determination of the sessions embedded in the access log file, the sessions belonging to each unique identity in the session list is stored in a file called as Result File. Also, at the same time, the process of updating the tree constructed before the execution of the module is done while storing the sessions into the Result file. The algorithm, Create_Result_File, shown in Figure 3.20 is derived to store all

information in the main memory to a permanent storage at the end of the execution of the module.

Before storing the sessions into the file, we firstly pass the session list once to determine the unique identities and update the tree constructed by the help of algorithm shown in Figure 3.14 (Lines 1-3). If the identity information is found on the tree meaning that it has some session information belonging to prior days, no update operation is required on the tree. Otherwise, if the user visits the web site for the first time that means she/he is new for the system, then the tree is updated with the new identity information. The important point here is that the start and index fields of the node holding the last domain identifier is set to 0. This property will be used in the process of storing the sessions to differentiate whether the identity is newly added or not.

After updating of the tree with the identities in the session list, the sessions both in the session list and in the Result file are stored in a temporary file and the contents of Result file are deleted. At the end, the temporary file created is then renamed as Result File and is used for the execution of the modules in the following days.

Traversing of all nodes in the tree is essential for the creating of Result File and the Index File. (Lines 4-11) The traversing of all nodes in the tree can be associated with the property of preorder traverse of the binary tree. Each sub tree below the class nodes is taken as a tree for the traversing of all nodes, that is, firstly the sessions belong to the identities under Class A category is stored and then Class B, Class C, Class D and at last FQDN Category, respectively.

As we discussed before, the domain identifiers of the identity with IP addresses are inserted to the tree from top to bottom. As shown in Figure 3.16 and Figure 3.18, the first domain identifier resides at the first level and the last domain identifier resides at the fourth level which is also the level holding the leave nodes.

```

[1] FOR each unique identity
[2]   Update the tree constructed
[3] End FOR
[4] FOR each class node in the tree DO
[5]   If the class node is between 1-4
[6]     Determine the identity by concatenating the domain identifiers from
           top to bottom
[7]   End If
[8]   Else If the class node is 5
[9]     Determine the identity by concatenating the domain identifiers from
           top to bottom
[10]    Reverse the identity variable acquired
[11]  End Else
[12] FOR each identity in the tree Do
[13]   NEW_start_index ← start point of the temporary file
[14]   If the node holding the last domain identifier of the identity has a start and
           end index value different than 0
[15]     Print all sessions between the start and end index in the Result file into
           a temporary file
[16]   End If
[17]   Search the session list belonging to the identity
[18]   If any session exists
[19]     Print the sessions just after the sessions written into the same
           temporary file
[20]     Delete session nodes belonging to the identity
[21]   End If
[22]   ELSE if the node holding the last domain identifier of the identity has a
           start and end index value with the value of 0
[23]     Print the sessions in the session list belong to the identity
[24]   End ELSE
[25]   NEW_end_index ← end of the temporary file
[26]   Update the start and end index variables in the node holding the last domain
           identifier with the NEW_start_index and NEW_end_index
[27]   Print identity, the start and end index values to the Index File
[28] End FOR

```

Figure 3.20: Algorithm Create_Result_File

The identities below the Node 1,2,3,4 are acquired by concatenating the domain identifiers from top to bottom. But, the identities acquired by the same way below the Node 5 meaning that the identities are in FQDN are then reversed because the domain identifiers of the identities with FQDN are inserted into the tree after reversing the identity information.

After acquiring a unique identity, the start and end index field of the node holding the last domain identifier of the identity is checked whether they have 0 or a value different from 0. 0 in the index field means that the identity is newly added to the tree, so it has no session belong to the prior days. Otherwise, it has some sessions in the Result File and these sessions must be rewritten to the file before storing the new session information to the file.

The `NEW_start_index` variable is set to the start position of the sessions in the temporary file belong to the identity. It is 0 for the first identity to be processed whereas it is the end position of the sessions belong to the antecedent identity processed (Line 13). Each session is written to the file with the owner of the session, the number of the pages in the session and the pages accessed in the session. If the variable in the start and end index fields of the nodes holding the last domain identifier of the identity has a value different from 0, then the sessions in the Result file between the start and end index values are written to the temporary file. (Lines 14-16) After writing the sessions to the temporary file, the session list is searched whether the identity has new session or not. If the identity has new sessions, then these sessions are added to the file just after the last session belong to the same identity. The session nodes written to the temporary file in the session list are then deleted to make the next searches in a smaller session list. (Lines 17-21)

If the variable in the start and end index fields of the nodes holding the last domain identifier of the identity is "0" meaning the identity is a new visitor for the system and has no session information in the Result file, so only the sessions in the session list is written to the temporary file. (Lines 22-24) After all sessions belong to the identity are written to the temporary file, the `NEW_end_index` variable is set to the end position of the sessions in the temporary file.

At the end of the line 24 of the algorithm we have the new start and end positions of the sessions belong to the each unique identity. We store these new variables into the fields of the node holding the last domain identifier of the identities

by exchanging them with the new indexes. Then, we write these index information into the Index File to be used by Recommendation Engine.

3.2 Recommendation Engine

In this section, we present the Recommendation Engine designed and implemented. Main goal of Recommendation Engine is to recommend next access pages to the visitor based on their past experiences as fast as possible as a list that can be reached by just clicking on the link. In the case of having no pages to recommend to the visitor based on her/his past experiences meaning that he/she visits the web site for the first time and the system has no session information in the Result file, pages that have been visited and spent more time by the identities belong to the parent domain of the visitor are recommended because the behavior of visitors in the same parent domain may also show the same trends with the visitor. As a result, the success of the pages recommended depends on the level that the pages are discovered. In other words, the success of the recommendation is high if the visitor has some session information in the system.

The Recommendation Engine works together with two parts. One of them is a code written in PHP scripting language and the other one is a CGI program running at the background. PHP is a scripting language similar to java script or visual basic script language. The important difference is that the code written in PHP is processed by the server, not by the client. The code is embedded into an HTML page and the web server processes this code before sending the page to the client's machine. The aim of PHP code embedded into the files is to get the IP address or FQDN of the visitor, its agent and the name of the page which will be used by the online module to produce appropriate recommendation to the visitor.

The code written in PHP scripting language embedded in the HTML page acquires the IP address or FQDN of the visitor and agent who made the request for the given page by parsing the environment variables of the connection between the

client and the web server. The other information acquired by the PHP code is the name and path of the HTML page. The name and path of the document requested is important because the system is designed for general usage. If this input were a static variable used in the program, there should have been a unique CGI program for each web page having the mentioned PHP code. Assuming that it is impossible and not acceptable in data mining concept, the name and the path of the page is given to the program dynamically by the help of the PHP code. An example of HTML page with PHP code is shown in Figure 3.21

```
<html>
<body>
{the other text and embedded objects are written here}
<? $filename = getenv ("PATH_INFO");
  $computer=getenv ("REMOTE_USER");
  $agent = getenv ("BROWSER");
  passthru (cgi.exe $filename $computer $agent)
? >
</html>
</body>
```

Figure 3.21: PHP script embedded into an HTML page

Client related information is assigned to three variables in the HTML page and are then sent to the CGI program called as FindPage running in the server as an input. FindPage runs after getting a signal from PHP code and produces the recommendation set. At the end of the execution of the FindPage, the pages to be recommended to the visitor are placed at the end of the page in a table with a link to the physical position of the pages.

3.2.1 Discovery of the pages to be recommended

The main goal of the FindPage which is triggered by PHP code embedded in the HTML page is to find the pages that the visitors have been spent the most time

just after the page requested. This process is performed by searching the sessions in the Result File by the help of the start and end indexes kept in the Index File.

The number of the pages to be recommended to the visitor can also be determined by the web master. The number of recommendation may be 1 or more according to the web master's decision. After FindPage is triggered, the program produces the same amount of pages with the number of pages determined by the web master. The FindPage implemented uses Index and Result File produced by the Log Analyzer module. The algorithm is shown in Figure 3.22.

The input variables of the FindPage are the identity information either in IP Address or in FQDN, the Agent and the name of the document requested by the visitor that were sent by PHP code in the HTML page. FindPage is triggered at every access to the page having PHP code.

- [1] Construct the tree
- [2] Update the start and index fields of the intermediate nodes
- [3] Search the identity whether it exists in the tree or not and load the Index_Table
- [4] FOR each index existing in the Index_Table
- [5] Find the pages in the Result File based on the page requested
- [6] Store them in Recommendation_Table
- [7] Sort the pages by time_spent
- [8] If the num_of_found_pages > num_of_recommendation **break;**
- [9] End FOR
- [10] If (num_of_found_pages < num_of_recommendation)
- [11] Recommend the most time spent pages to the visitor to complement the number of pages to be recommend
- [12] End If

Figure 3.22: Algorithm used in FindPage

First task done in FindPage is to construct the tree because all information about the identity is stored on that tree. We have discussed that the Log Analyzer module outputs two main files, Index and Result File. FindPage uses the Index File to create the tree and the Result File to discover the pages to be recommended. The

algorithm `Construct_Tree` shown in Figure 3.17, constructs the tree. The important point at this step is that the start and end index values of the identities are loaded into the nodes holding the last domain identifier of the identities. The start and end index fields of the intermediate nodes are set to 0 as shown in Figure 3.19. Then the start and index fields of intermediate nodes are updated according to the start and end index values of the nodes at one low level connected to the node.

<i>Identity</i>	<i>Start index</i>	<i>End Index</i>
pcmil1.cs.bilkent.edu.tr	45321	56784
pcmil2.cs.bilkent.edu.tr	56784	63298
pc501b.cs.bilkent.edu.tr	63298	75192
ppp140.bcc.bilkent.edu.tr	75192	84567
ppp145.bcc.bilkent.edu.tr	84567	92133
tahir.ef.bilkent.edu.tr	92133	98739
403a.ef.bilkent.edu.tr	98739	99657

Table 3.2: Example identities with their start and end indexes

We will clarify this issue with an example by using the identity information shown in Table 3.2. After inserting of these example entries into the tree, they will be placed at somewhere in the tree shown in Figure 3.23. (Line 1)

As shown in the figure, all start and end index values are placed at the leaf nodes. For instance, the sessions between the position 45321 and 56784 of the Result File belong to the identity “pcmil1.cs.bilkent.edu.tr” whereas the sessions between the position 92133 and 98734 of the Result File belong to the identity “tahir.ef.bilkent.edu.tr”. Then, the value of the start index field of any intermediate node is set to the start index value of the first node connected to the mentioned node. At the same time, the value of the end index field is set to the end index value of the last node connected to the mentioned node.

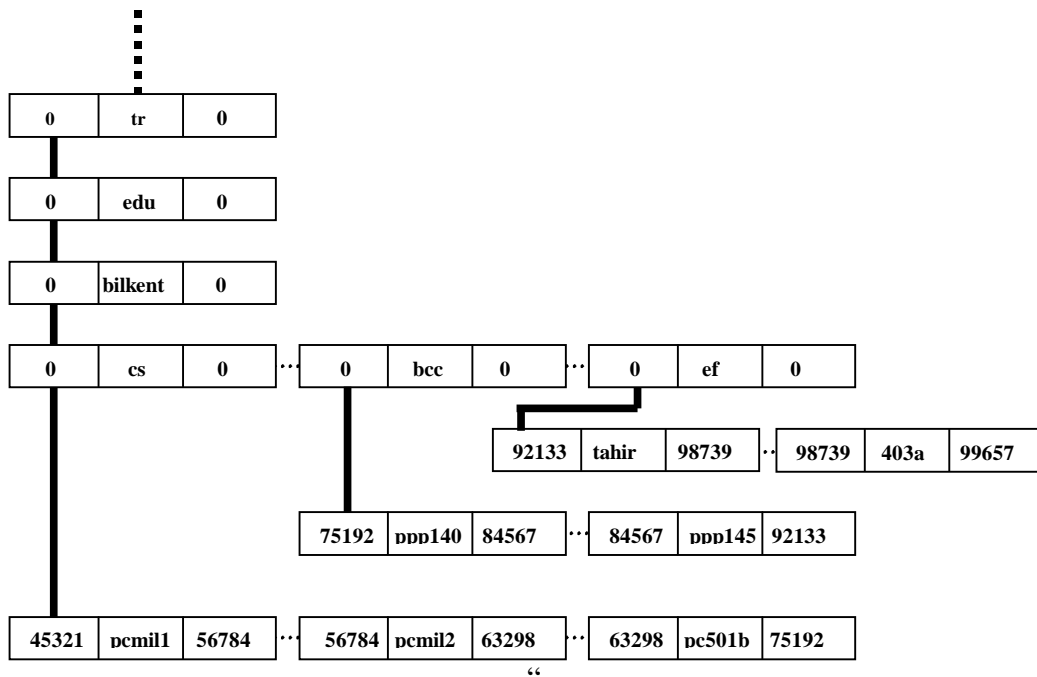


Figure 3.23: A part of the tree with the example entries

After updating the indexes, the tree looks like in the Figure 3.24. As shown in the figure, the start and end index fields of all nodes in the tree has a value minimum 0 or maximum the size of the Result File due to the start and end index values of the child nodes connected to the parent nodes. We can draw some conclusion by using the tree above. For instance, the sessions of the users belong to “cs.bilkent.edu.tr” in the Result file exist between the position of indexes 45321 and 75192 whereas the sessions of the users belong to “ef.bilkent.edu.tr” exist between the position of indexes 92133 and 99657.

After constructing the whole tree, the identity is searched on the tree (Line 3). But before the search operation, a table with two fields in each row, one field for start index and one field for end index value, is allocated and the size of the table allocated is dependent to the number of domain identifier of the identity. The search operation on the tree is made from top to bottom.

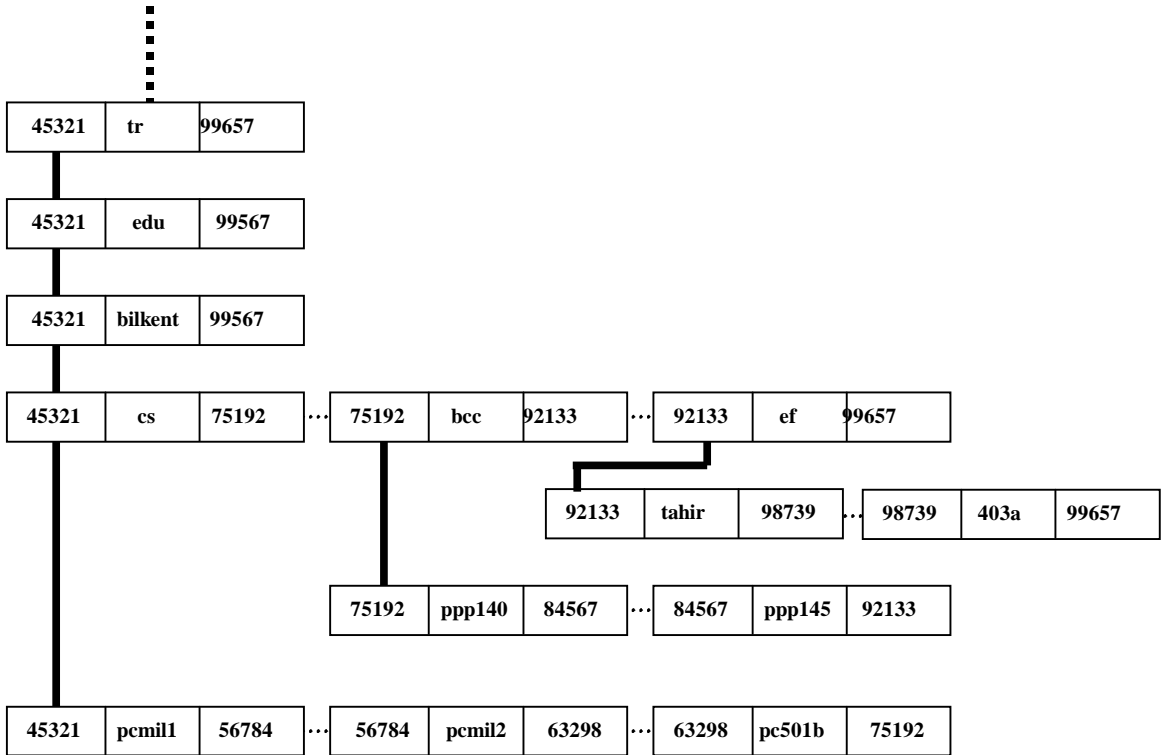


Figure 3.24: The same part of the tree after updating the index fields

If the identity is IP address, then the class node of the identity is found by checking the first octet of the identity as described before. Otherwise, if it is a FQDN, the class of the identity is clear, that is, Node 5. After determining of the class node, the identity is searched in the nodes connected to the class node of the identity. The start and end index values of the nodes on the path until last domain identifier of the identity is found are loaded into the allocated Index_Table to be used in case of having no pages to recommended to the visitor based on his/her past experience. If the identity making the request is “pc501b.cs.bilkent.edu.tr”, the search operation is made on the path tr → edu → bilkent → cs → pc501b and the start and end index values of these nodes is loaded into the Index_Table like in Figure 3.25

Node “tr” →	45321	99657
Node “edu” →	45321	99657
Node “bilkent” →	45321	99657
Node “cs” →	45321	75192
Node “pc501b” →	63298	75192

Figure 3.25: Index_Table

The search operation in the Result File begins from indexes in the last row to the indexes in the first row of the table. The start index with the value of 63298 and the end index with value of 75192 shows the position of sessions belong to the identity “pc501b.cs.bilkent.edu.tr” and the first aim is to find the probable pages to be recommended in the sessions between these indexes. If no page is found to recommend for the page requested, then the probable pages are searched between the position of 45321 and the position of 75192 holding the sessions belong to the identities connected to “cs.bilkent.tedu.tr” with discarding the section searched before.

- [1] Open Result File
- [2] Position the cursor to the start position of the sessions
- [3] While the end position of the sessions
 - [4] Read the entry and parse it
 - [5] If the entry represent a new session
 - [6] Read the line and parse it to obtain the name of page and the time
 - [7] If the page is the same as the page requested
 - [8] While the end of the active session
 - [9] Load the page information into Recommendation Table
 - [10] End While
 - [11] End If
- [12] End While

Figure 3.26: Algorithm Discover_Pages

This process continues until the number of the pages found satisfies the number of recommendation determined by the web master. The search operation in the Result File is done by the algorithm shown in Figure 3.26.

After the start and end index values are determined, the Result File is opened and the cursor is positioned at the start index value. (Lines 1-2) Then each entry until the end index is read and checked whether it has an indicator flag representing a new session or not because the Result File is stored in such a way that each session is separated with an indicator flag. (Line 5) If the entry contains that flag, all entries between the entry containing the flag and the beginning of the next session are read and parsed to obtain the name of the page (Line 6) to compare the page name of the entry with the page requested. If comparison is positive then all pages and their `time_spent` values until the beginning of the next session are stored in the Recommendation Table (Lines 7-10).

Recommendation Table is designed as a linked list because the number of pages found cannot be determined before. A memory allocation is made for every page found and a new node is added to the list. These nodes hold the name of the page, the `number_of_hits` and the `time_spent` value. If any page found for recommendation exists in Recommendation Table meaning that it occurs twice or more, the `number_of_hits` of the node holding the mentioned page is incremented by 1 and the time spent value is updated by calculating the average. The pages in the Recommendation Table is sorted by their `time_spent` variable increasingly because we are interested in the time spent of the pages for the recommendation,

If the number of the pages in Recommendation Table does not satisfies the number of recommendation determined by the web master, then the search operation continues for the indexes in the one upper row of the `Index_Table`. But before going into this process, the pages in the Recommendation Table are stored in another list that is used as a result set by `FindPage` and the pages in Recommendation Table are deleted. Assuming that the number of pages to be recommended in the result set is 5

and the number of recommendation determined by webmaster is 10, we need extra 5 pages. These five pages will be acquired by the process of the indexes belonging to the parent domain of the identity. Of course, there may be found more than five pages for the parent domain of the identity, but after sorting the pages found, we take the first five pages in the Recommendation Table and add them to the result set to complete the necessary number of pages for the recommendation. If the number of pages found for the identity is adequate for recommendation, we do not do any work for the indexes of the parent domain of the identity.

It is clear that the algorithm executes in the least time if the number of pages satisfies the number of recommendation determined by the Webmaster at the first iteration of the algorithm shown in Figure 3.22. In case of having not enough pages for recommendation, the same process will continue for the parent domain. Because the indexes of the parent domain will cover wider segment of the Result File than the segment of the identity, the process time of the algorithm increases.

Node “tr”	→	45321	99657		
Node “edu”	→	45321	99657	Node “tr”	→
Node “bilkent”	→	45321	99657	Node “edu”	→
Node “cs”	→	45321	99657	Node “bilkent”	→
Node “pc503”	→	0	0	Node “fen”	→
				Node “dorm3042”	→
				0	0
				0	0

(a)
(b)

Figure 3.27: Index_Table of the given identities

If an identity is not available on the tree meaning that the system has no session information about her/him, the rows assigned for the domain identifiers that

could not be found in `Index_Table` are set to 0. For instance, for the identity “pc503.cs.bilkent.edu.tr”, the `Index_Table` will be as in Figure 3.27(a) and for the identity “dorm3042.fen.bilkent.edu.tr”, the `Index_Table` will be as in Figure 3.27(b).

The search operation is done only for the indexes having a value different from 0. For the identity “pc503.cs.bilkent.edu.tr”, the first search is done for the parent domain of the identity (cs.bilkent.edu.tr) whereas for the identity “dorm3042.fen.bilkent.edu.tr”, the first search is done for the domain of the identity (bilkent.edu.tr). In the first case, the sessions belong to the identities in “cs.bilkent.edu.tr” domain, which is at one-level up, are searched for the probable pages to be recommended. In the second case, the sessions belong to the “bilkent.edu.tr” domain, which is at two-level up, are searched for the probable pages to be recommended.

Chapter 4

Efficient Use of Resources

The most important task in all system is to use resources efficiently because there is no resource that is not scarce. If the limit of the resources could not be taken into account, the system designed becomes unusable in one day in future. So, all resources to be used by the system must be considered as the most important factor for the existence of the system

The system must use some resources efficiently such as the main memory, the disk capacity and the time. As we discussed earlier, the size of log files grows in an enormous rate. Sometimes the number of entries added to the access log file of the web site becomes two or more times more than a normal day, especially on the days the web site has an announcement for the visitors of the web site, for instance, the announcement of the grades in a university.

4.1 Efficient use of the main memory

The efficient use of resources concept is related to Log Analyzer module of the system because the most and dense work is done in that module. The Log Analyzer module is generally designed to execute everyday to update the session information kept in the Result File. The module can hold all information in the main memory for

the entries belonging to one day. Assuming the system has been designed for general usage, the module is configured to process all entries even in case of having more entries that cannot be processed in one hop. The main steps of the use of main memory can be depicted as in Figure 4.1.

```

[1] For each entry in access log file Do
[2]   Check whether it is a valid request or not
[3]   TOTAL_MEMORY_OCCUPATION = 0 and MEMORY_LIMIT =XX
[4]   If the entry is valid
[5]     Insert the information to be used into the main memory
[6]     TOTAL_MEMORY_OCCUPATION←
      TOTAL_MEMORY_OCCUPATION + the size of the objects inserted
      into the main memory
[7]     If (TOTAL_MEMORY_OCCUPATION > MEMORY_LIMIT)
[8]       Position← the position of the access log file
[9]       Eliminate_Session_List
[10]      Update the Result File with the new finished sessions
[11]      Delete the stored session nodes
[12]      Update the Index file
[13]    End If
[14]    If there exists remaining entries in the access log file
[15]      Process the entries after the Position variable by going on from
      Line 2
[16]    End If
[17]  Else
[18]    Delete all objects in the main memory
[19]    Exit the program
[20]  End Else
[21] End If
[22] End For

```

Figure 4.1: Algorithm Use_Memory_Efficient

The space requirement in the memory for the objects used in our system can be calculated by summing the object sizes at each allocation because most of them such as a session object are created by dynamic memory allocation. So, TOTAL_MEMORY_OCCUPATION is assigned for calculating the memory

occupation. (Line 3) The variable assigned is set to 0 at the beginning of the module and for every object that is loaded into the main memory during the execution of the module; it is incremented by the size of the object. (Line 6) The system operator assigns an upper limit to another variable, `MEMORY_LIMIT`. The module configures itself in the case of exceeding the `MEMORY_LIMIT`. This configuration is made by transferring the objects loaded in the main memory to the permanent storage and release the occupied space of the main memory. (Lines 7-12) The module knows where it has stopped and finds the position of the access log file where the last entry processed. If there still exists remaining entries, the module processes them as if it starts for the first time.

4.2 Efficient use of the disk capacity

Because Log Analyzer module executes and filters all session information everyday from the new entries in the access log file and updates the session information kept in Result File, the size of the Result File grows everyday. If this fact is not taken into consideration, the size of the Result file may reach an unacceptable size that cannot be hold in the permanent storage. The other important factor that must be taken into account to limit the size of the Result File is the speed of the execution of the Recommendation Engine. This effects the loading time of a page to the client's machine because the search operation for the pages to be recommended is mainly done in Result File. In other words, the speed of Recommendation Engine decreases while the size of the Result File increases.

The loading time of a page requested to the client's browser is very important for all dynamic web applications. If the loading time increases due to the speed of the dynamic content in the HTML pages, the visitor may be bored while the page is being loaded and may leave the page before the page is loaded into the browser. So, the system operator must choose the speed of the dynamic applications embedded in the HTML pages reasonable. There exist two important factors affecting the speed of

the Recommendation Engine designed. The possible cases of having no pages to recommend to the visitor based on her/his past experience and the size of the Result File. The former factor cannot be controlled, but the latter can be controlled by the system operator.

We have discussed the first factor affecting the speed of the Recommendation Engine above. If an identity requesting a page from the web server is new for the system meaning the system has no session information about the identity, the system tries to find the pages that are accessed by the visitors coming from the parent domain of the visitor and recommends the pages found to the visitor. For instance, assuming that the identity “pcmil234.cs.bilkent.edu.tr” requests a page from the server and is a new for the system, the Recommendation Engine will find the pages which are accessed by the visitors belonging to its parent domain, which are in “cs.bilkent.edu.tr” domain to make a recommendation to the visitor by searching the session section in the Result File belonging “cs.bilkent.edu.tr” domain. It is fact that the size of the session section belonging to the domain “cs.bilkent.edu.tr” is greater than that belonging to the identity “pcmil234.cs.bilkent.edu.tr”, so the time spent to find the pages in the sessions belonging to the visitors coming from the parent domain is more than that belonging to the visitor itself. The system encounters with these circumstances frequently because there will be always new identities for the system.

When the size of the Result file reaches an unacceptable size, there come some questions that must be answered such as how the size of the file can be reduced. Another question may be “Do we delete whole session information or some pages from the session information to reduce the size of the file?” It was discussed that the main aspect of the thesis is to recommend the pages that the visitors have been spent the most time to the visitor. For instance, after retrieving the page A, if the visitor spends 24 seconds on page B in one session and spends 45 seconds on page C in another session, we firstly recommend the page C first and then the page B

to the visitor. We can draw the answer of the questions above from our main goal as far as we discussed. The least interesting thing in the Result File is the page that has been accessed by the visitor for a very small time period.

The main problem in this phase is to detect how many and which pages should be eliminated from the sessions. So, there must be a component in the system to hold some information about all pages in the web site in case of “*forgetting*” the pages. The word “*forgetting*” means the elimination of some input which has no effect on the access of the system or have become useless for the system.

So, the system is designed in such a way that it has an ability to configure itself and to start the process of the elimination of the pages in the sessions automatically that are discovered by the *forgetting* algorithm in the case of exceeding the threshold predetermined for the size of the Result File. The forgetting is done by the elimination of the pages, but there are two circumstances that should be lightened. Only two pages may form some sessions in the Result File. That is, the visitor may have been requested a page and another page after retrieving the first page. At that time, she/he may have been decided to exit. So, only two pages retrieved by the visitor form the session. Assuming that one of the pages in that session is the same as one of the pages discovered by the forgetting algorithm and must be eliminated to reduce the size of the Result File, the number of pages in that session is decremented by 1 and becomes 1 after eliminating that page. The sessions having only one page cannot be accepted as a session anymore and must also be eliminated. Sometimes, the visitor who has such a session explained above may have not have one more session information in the Result File. After eliminating the session of the visitor, there is no available information about her/his surfing in the Result File anymore meaning that the index information about the visitor must also be eliminated. In other words, the elimination may begin with a single page, but it may cause the elimination of the session it belongs and even the index information of the visitor.

The system operator determines an upper and lower bound for the size of the Result File by taking some factors into account. When the size of the Result file updated by the Log Analyzer module exceeds the upper bound, the forgetting process takes into effect to reduce the size of the Result file to the lower bound. These bounds are generally determined parallel to the speed of the Recommendation Engine. The upper bound must be an acceptable limit that the speed of the Recommendation Engine that uses the Result File to discover the pages to be recommended to the visitor is as fast as possible. If that bound is determined too small, the speed of Recommendation Engine may be fast because it searches on a small file to find the pages to be recommended but the success of the pages recommended decreases because less session information can be kept in such a small size file. The determination of the lower bound is related to the frequency of the execution of the forgetting algorithm. If the lower bound is set to very close to the upper bound, the probability of execution of the forgetting algorithm increases because the gap between the lower and upper bound may be filled in a few days. Of course, the elimination of the pages or sessions is not a desirable action because the elimination of them is the deletion of some information about the visitor. So the lower bound must be determined small enough to hold enough information about the visitors and big enough which makes the speed of the Recommendation Engine reasonable.

A Page_Information_List is maintained to hold the information about the pages in the web site in the execution of Log Analyzer module that will be used by the Forgetting Algorithm. The name of the page, the number of hits and the time spent on that page are dynamically stored into this list. The name of the pages in the list is unique. The changing variables are the number of hits and the time spent .

There is an important point here to be clarified. However we are interested in the time spent on the pages, some of the pages may be visited for a short period of time which are really important for the users. For instance, a page containing a grade

list for the students of a university may be commonly visited for a short period of time. They visit that page just for looking at their grades and then leave it. According to the assumption used in the thesis, that page must be classified as a navigational page. But, that page is more important for the students than the other pages especially on the days after an exam or homework. So, there comes a problem with that page about making it a *content* page.

In the light of these circumstances, the calculation of the spent time value of such pages must be correlated with the number of hits on these pages. For instance, assume that the page A containing a story has been visited just for once and the time spent for that page is 50 seconds whereas the page B containing a grade list for the students has been visited four times and the time spent for that page is 15 seconds for each visit. It seems as that the content page must be page A, not page B. But, page B is more important than page A for the students because it has been visited much more than page A on the special days. As a result, by making the calculation of spent time value of the pages as shown in Figure 4.2, the viewing time of such pages is increased due to the number of hits and as a result, its class changes from navigational to content.

<u>Number of hits (i)</u>	<u>Duration</u>	<u>Time Spent</u>
1	10	0+10 =10
2	20	10+20=30
3	15	30+15=45
4	55	45+55=100

Figure 4.2: Calculation of the time spent

Page_Information_List is stored permanently in a separate file at every end of the execution of the Log Analyzer module and loaded into main memory at every beginning of the execution of the module. Besides, the list is updated with the next new entry information in the access log file.

The “*forgetting*” process is done by the algorithm shown in Figure 4.3. The first work done by the algorithm is to check the size of the Result File whether it exceeds the threshold determined. If it is positive, the algorithm starts the forgetting process (Lines 1-2). The next step is to detect the pages to be forgotten and eliminate them from the Result File to reduce the size of the file to lower bound. The page information list is constructed and sorted by the time spent field decreasing because we try to forget the pages that the visitors have spent the least time. We have also the number of the occurrence of the pages in the Result File. (Lines 3-5) We start with the first page in the list to determine the difference value, which is used to reduce the size of the Result File. (Lines 6-10) So, by multiplying the number of occurrence with the length of the name of the page, a difference value is calculated. If this difference value is enough to reduce the size of the Result File to the lower bound of the Result File, the process of forgetting is started. In the case of having not enough difference value, then the difference value is calculated again by incrementing with the next page’s difference value in the list until the enough difference value is acquired to reduce the size of the Result File to the predetermined lower bound. As a result, we have a list of pages that have been visited for less time than that of the other pages in the site.

After determining the pages to be forgotten, the process of elimination of these pages from the Result File starts. Main components of this process are the list of pages to be eliminated and the Result File. This process is done for all unique identity in the Result File. Because the new session information must be stored in the Result File after forgetting the pages from the sessions belong to a unique identity that means the index variables must be updated for each unique identity. Besides, sometimes we may have not enough session information about the identity after forgetting which have been discussed in the elimination of the session section. In these cases, the index entry for that identity must also be eliminated from the Index File because it has no session information in the Result File.

```

[1] Filesize  $\leftarrow$  size of the Result File
[2]   If Filesize > Upper Limit
[3]     Load the page information into the memory
[4]     Sort the Page Information List by time_spent decreasingly
[5]     diff  $\leftarrow$  0
[6]     For each page in the page list
[7]       diff  $\leftarrow$  diff + (num_of_hit *strlen (page))
[8]       Add the page to the Forget list.
[9]       If (Filesize – diff) <= Lower Limit break
[10]    End For
[11]    Open Result File
[12]    For each unique identity in the file
[13]      Load Session information into session list
[14]      For Each Page in the Forget List
[15]        If it exists in session, delete from the session
[16]      End For
[17]      Store the new session information and make a new indexing
[18]    End For
[19]  End If

```

Figure 4.3: Algorithm Forget

The important thing here is that the tree holding the identity and index information is reconstructed and updated after each forgetting process. So, the root of the tree is constructed before the beginning of the forgetting process and the tree is updated with the identities that have enough session information after forgetting process.

The Result file is opened after constructing the Forget List. (Lines 11-18) By starting the first identities in the file, the session information belong to the identity is loaded into the session list in the memory. So, we have session information in the session list just for a unique identity, not for all identities. The session list may have one or more sessions belong to the identity. After constructing the session list for the identity, we check all pages in the sessions whether it exists in Forget List or not. That the page checked exists in Forget List means that the page must be eliminated from the session. This process is done for every unique identity in the Result file until the end of the file.

After eliminating of the pages from the sessions, the identity information is stored in the tree and the session information is stored in a new Result file. The indexing process is done according to the new Result File created after forgetting process. The size of the Result file automatically reduces to the lower limit determined by the system operator at the end of the forgetting algorithm terminates.

After the forgetting process is terminated, note that all pages found to be forgotten have been deleted from the Result File. As a result, the occurrence of the pages and the `time_spent` variable in the Forget List are set to 0 and the file holding the information about the pages in the web site is updated with these changed values.

Chapter 5

Evaluation

In this chapter, the results of the experiments conducted for determining the applicability of the system will be demonstrated. Although, the system will work on the web site of the CS department of Bilkent University, we close to run experiments on a standalone computer because of the changing load of the UNIX systems to obtain more accurate results.

The first experiment has been done to determine the effect of the preprocessing phase. Remember that if a web page consisting of three images is requested, the web server puts four separate entry into the access log file, one for the page itself and the others for the images containing it. The system focuses only on the entries containing HTML pages. So, as discussed before, a preprocessing algorithm is applied onto the access log file before session identification to eliminate these entries containing images.

We performed our experiments by using the access log file maintained by the web server of the CS Department at Bilkent University. We had an opportunity to evaluate the success of the results of the experiments by using real data, not generated data. We have chosen a 10-day period fragment of the access log file as the resource for our experiments.

Day	Size of Log File	Total Number of the Entries	Number of the Relevant Entry
1	1873976	9229	3769
2	5206384	26100	10929
3	4717906	24600	13255
4	3421910	16324	6525
5	5063804	26319	13547
6	5330442	24501	11802
7	5112256	23721	12821
8	3557033	17038	7247
9	4246646	20212	8136
10	4038749	19503	8139

Table 5.1 Test results of the Preprocessing Algorithm for 10-day period. Size values are in byte.

Log Analyzer module runs automatically everyday at a predetermined time. (especially when the load of the system is the minimum). At every execution, the last entry processed is obtained and the module processes the newly added entries.

Table 5.1 shows the results for the execution of the preprocessing algorithm. In this table, the second column shows the size of the access log files belong to days shown in the first column meaning that 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. Third column gives the total number of the entries in the access log files while the fourth one gives the number of entries that are relevant for mining process.

It can be said that at least the half of the total entries contain irrelevant items (e.g. pictures, sounds etc.) and sometimes this number may reach to $2/3$ of the access log file. Some of the existing web servers have an ability to eliminate irrelevant entries before recording them in the access log file. For instance, the web server can be configured as not to put entries including images or sounds or whatever. But, unless such a configuration is available, a preprocessing algorithm like the one used in our system must be applied to eliminate these irrelevant items.

The second experiment is conducted to determine the properties of the sessions created. Table 5.2 shows the results of the experiment conducted for Session Identification Algorithm. As discussed in 3.1.3, an elimination process is applied onto the sessions created at the end of the execution of the Log Analyzer module just before the storing the session information into the file.

We have discussed that the last access of the sessions are eliminated because we cannot calculate the time spent on the last page. After discarding the last page from the sessions, some sessions may not have enough information about the visitor and must be eliminated.

Day	Total Session	Eliminated	Finished	Unfinished
1	1103	791	225	87
2	2775	2004	633	138
3	2290	1717	394	179
4	2205	1603	367	235
5	2663	1951	408	304
6	2740	1978	407	355
7	2817	1975	420	422
8	2792	1939	386	467
9	2847	1869	456	522
10	2829	1806	472	551

Table 5.2 Test results of the Session Identification Algorithm

The second column represents the number of total sessions created at the end of the execution of the Algorithm `Session_Identification`. The number of the eliminated sessions is shown in the third column of the table. The reason that the number of eliminated sessions is high is that we are not interested in the path completion or the number of pages on that path. It means that a session consisting of only two pages may be important for a system based on the path completion, but it must be eliminated for our system. In real life, most of the visitors begin surfing a web site with a page in the site and then retrieves another page and at last they leave the web site. So the sessions belonging to these kinds of visitors have only two pages and do not have enough information for our system. Besides, some of the users begin

to visit the site in a page and leave the site without going any further pages. We do not accept the sessions consisting only one page as a valid session, so these kinds of sessions are eliminated. By taking this fact into account, it can be said that the high number of the eliminated sessions can be accepted as reasonable.

Also, we have discussed that some sessions still continue at the end of the execution of the module and these sessions are kept in the Session File and then loaded into the memory as “*open*” sessions just before the execution of the module on the next day. These unfinished sessions are updated with the new entries and then closed. The unfinished sessions in the i^{th} row of the table are loaded into the memory on the $(i+1)^{\text{th}}$ day and can be thought that they are included in the number of finished sessions on the $(i+1)^{\text{th}}$ row if and only if all of them are accepted as closed or not eliminated sessions.

Because the identity information is the most important factor for the presented model, an experiment is conducted to clarify the user identification process. The results of the experiment are given in Table 5.3.

Day	Num of identities	Size of IndexFile	Size of Result_File
1	147	5300	49630
2	512	19724	186316
3	739	28659	278560
4	944	37445	375926
5	1159	46043	485616
6	1376	54510	583341
7	1600	63415	680885
8	1807	71647	776893
9	2004	79536	913301
10	2224	88848	1024632

Table 5.3 Test results of the identity information and the size of the Index and Result_File. Size values are in byte.

The number in the $(i+1)^{\text{th}}$ row of the second column of the table represents the total number of unique identities on the i^{th} day meaning that the total number of unique identity at the end of the 10^{th} day is 2224. The third column represents the

size of the IndexFile consisting of the identity information and their start and end indexes indicating the position of the sessions in the Result_File. And the last column represents the size of the Result_File consisting of all sessions belong to the identities. The size of the Index and the Result_File are important for Recommendation Engine because it uses these two files to generate a recommendation set based on the session information of the visitors. The size of the Result_File is more important than that of the IndexFile, so the size of the Result_File must be chosen reasonable by the web master as discussed in Chapter 4.

The main phases of Log Analyzer module can be divided into four main steps as shown in Figure 5.1

Step 1: Preprocessing Phase

Step 2: Constructing the tree at every beginning of the execution of the module

Step 3: Creating the sessions

Step 4: Storing the identity and session information

Figure 5.1 Main Phases of the Log Analyzer Module

Day	Step 1	Step2	Step3	Step4
1	0,66	0	0,21	0,05
2	1,61	0,005	1,45	0,55
3	1,43	0,011	1,87	0,94
4	0,97	0,017	0,69	1,43
5	1,64	0,019	1,92	2,41
6	2,05	0,028	1,31	2,31
7	1,68	0,031	1,42	3,95
8	1,22	0,043	1,12	4,25
9	1,81	0,045	1,43	5,17
10	0,98	0,048	1,44	5,22

Table 5.4 Test results of each phase of the Log Analyzer module. Time values are in seconds

The second column of the table shows the running time needed to perform to determine whether the entries are valid or not as shown in the third and the fourth column of the Table 5.1. By comparing the number of the “Total Entry” column in Table 5.1 to the running time of preprocessing of these entries, it is clear that the running times of the entries are proportional to the number of total entry.

The third column of the table represents the time needed for constructing the tree at every beginning of the execution of the Log Analyzer module. As discussed above, the identity information is kept in a file called as IndexFile and this identity information is loaded into main memory at every execution of the module just before the processing the newly added entries. In the first day, the time is 0 because no identity information is available for the previous days. As shown in Table 5.3, the module processed the entries for 147 identities and stored this information in the IndexFile with a size of 5300 bytes. The value in the third column of the second row, 0.05 sec, gives us the time needed to construct the tree with these 147 identities. The time needed for constructing the tree with the identity information gathered in one day before increases because the total number of the identities discovered increases everyday. The time values for preprocessing are proportional to the size of the IndexFile and the number of the identities in the IndexFile.

The fourth column of the table represents the time needed for creating the sessions. The sessions are created with the entries in the access log file after eliminating the irrelevant items as shown in the fourth column in Table 5.1. The time values are proportional to the number of the relevant entries in the access log file in the same day.

The results indicate that the most significant time consumption occurs during the execution of the fourth step. In the fourth step, all sessions created into the Result_File and the time needed for this operation is given in the last column of the table. As discussed in 3.1.6, before storing the identity and session information into the file on the i^{th} day belong to the identity X, all session information belong to the

same identity in the Result_File created on prior days is firstly rewritten in the Result_File and then the sessions created on the i^{th} day are added to the session information. So, excessive numbers of I/O operations performed in this step make the time needed for storing the identity and session information higher.

A sample fragment of IndexFile is given in Figure 5.2. A single entry in the file represents the identity information, start index value and end index value. As discussed in Section 3.1.6, IndexFile is created by searching all nodes of the tree. Firstly, the start and end index values of the identities belonging to Class A are written to the IndexFile, and then Class B and at last FQDNs. At the end of the updating of IndexFile, the first section of the IndexFile is for the identities belonging to Class A whereas the last section is for the identities belonging to the FQDNs. The identities shown in the sample fragment belongs to the Class A residing at the first section of the IndexFile.

```
.  
. 62.60.74.123 22049 22225  
62.85.56.131 22225 22886  
62.98.243.73 22886 24062  
62.108.64.1 24062 24561  
62.114.36.165 24561 24706  
62.140.2.8 24706 25052  
62.235.20.22 25052 25492  
62.248.0.161 25492 26140  
62.248.17.152 26140 26390  
62.248.17.194 26390 26484  
62.248.25.2 26484 26642  
62.248.105.102 26642 27293  
62.251.172.26 27293 27421  
63.69.85.2 27421 27674  
63.70.129.194 27674 27804  
63.83.144.159 27804 27987  
63.88.160.101 27987 28462  
63.97.144.96 28462 28772  
63.115.16.66 28772 28977  
.  .
```

Figure 5.2 A fragment of IndexFile

The session information belonging to some identities (written in *italic*) in Figure 5.2 is given in Figure 5.3. The first entry of the sessions indicates the Identity Information/Agent pair and the number of the pages in the session. The remaining entries in the sessions indicate the pages in the sessions and the spent time value of the page.

```
*62.248.105.102 "Mozilla/4.0 (compatible; MSIE 6.0; Windows 98; Win 9x 4.90)" 18
/
  4
/courses.html 3
/~guvenir/courses/CS315 8
/~guvenir/courses/CS315/p1.html 15
/~guvenir/courses/CS315/p2.html 4
/~guvenir/courses/CS315/hw1.html 8
/~guvenir/courses/CS315/hw2.html 7
/~guvenir/courses/CS315/hw3.html 7
/~guvenir/courses/CS315/quizzes.html 79
/~ugur/teaching/cs319 15
/~ugur/teaching/cs319/outline.html 6
/~ugur/teaching/cs319/assignment.html 29
/~endemir/courses/cs35101/cs35101.html 21
/~endemir/courses/cs35101/hw1.html 13
/~endemir/courses/cs35101/hw2.html 9
/~endemir/courses/cs35101/hw3.html 14
/~endemir/courses/cs35101/grades.html 23
/CS299 14
*62.251.172.26 "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)" 2
/~erayo/wml/cv.html 141
/~erayo/wml/personal-data.html 7
*63.69.85.2 "Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0)" 5
/~akman/jour-papers/air/air.html 7
/~akman/jour-papers/air/node1.html 10
/~akman/jour-papers/air/node2.html 4
/~akman/jour-papers/air/node3.html 16
/~akman/jour-papers/air/node4.html 4
*63.70.129.194 "Mozilla/4.77 [en] (Win98; U)" 3
/~david/derya/ywc.html 127
/~david/tywc/games/wgame2 14
/~david/derya/about.htm 21
*63.83.144.159 "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; isp1057; Q312461)3
/~david/derya/ywc.html187
/~david/derya/keypals.htm 7
/~david/derya/members/newappl.html 331
```

Figure 5.3 A Fragment of Result_File

Another experiment is conducted to determine the effect of the Forgetting algorithm. Forgetting in our system has been described as to eliminate the pages that were visited for a short time period in the case of exceeding the threshold value predetermined for the size of the Result_File to use disk capacity efficiently. Test results are shown in Table 5.5.

Size of Log	Upper Limit	Lower Limit	Time	Size of the Result_File	Size of the IndexFile	Number of Identity
42569943	1200000	1000000	647	1024632	88848	2218
42569943	1000000	800000	651	721949	74464	1857
42569943	1000000	600000	656	492366	54238	1345

Table 5.5 Test Results of Forgetting Algorithm. Time values are in seconds while the sizes are in bytes.


We have decided to utilize a larger log file for this experiment, so we have chosen the same 10-day period access log file as a whole that has been also used in other experiments. The size of the access log file is quite huge because it contains all requests in a long time period. The first row of the table shows the results without exceeding the threshold shown in the second column. The reader is reminded that the Forgetting Algorithm starts if and only if the size of the Result_File exceeds the threshold predetermined. Because the size of the Result_File does not exceed the threshold in the experiment shown in the first row, the module did not start the Forgetting algorithm. In this experiment, 2218 unique identity information is stored in the IndexFile meaning that the Result_File holds all session information belonging to these 2218 identities.

Then, the Upper Limit has been lowered to make Forgetting Algorithm run and it is set to the value shown in the second column of the third row. Two separate experiments were conducted to see the effect of the algorithm by setting the Lower Limit to different values. The difference between the time needed to process all entries in the access log file without exceeding the threshold and that in case of exceeding of the threshold is used to eliminate the pages discovered by the Forgetting Algorithm from the Result_File.

Another important point here to be clarified is the decreasing number of the identities. As discussed in section 4.2, if an identity has no more session information after eliminating the pages that are found by the Forgetting Algorithm to reduce the size of the Result_File from the sessions belong to the identity considered, then this

identity is also discarded from the IndexFile. So the number of the identities in the IndexFile decreases due to the threshold determined for the Lower Limit. This situation also affects the size of the Result_File after processing the Forgetting Algorithm.

A sample output of the Recommendation Engine is given in Figure 5.4. As shown in the figure, the Recommendation Engine has been executed for the page “/~guvenir” for the identity “pcmil2.cs.bilkent.edu.tr” making the page request. The recommended pages have been found according to their spent time values. Also, the sample output shows the effect of BACK button. For instance, assuming that the page “/~guvenir” has been reached after the page “/csfaculty”, the visitor has decided to go to the page “/~aykanat” by clicking BACK button to return the page “/csfaculty” and then clicking on the link of the page “/~aykanat”



H. Altay Güvenir

Professor, Chairman
 Department of Computer Engineering
 Bilkent University
 Ankara, 06533 TURKEY

Email: guvenir@cs.bilkent.edu.tr
Phone: +90 (312) 290 1252
Fax: +90 (312) 266 4047

Research interests: Artificial Intelligence, Machine Learning and Data Mining.

Research: Publications, Projects, FunApp Repository, Software, ML Group
Teaching: Courses, Schedule, Theses Supervised
Professional: Activities, Vita

1 .. http://www.cs.bilkent.edu.tr/~guvenir/publications.html
2 .. http://www.cs.bilkent.edu.tr/~guvenir/projects.html
3 .. http://www.cs.bilkent.edu.tr/~guvenir/software.html
4 .. http://www.cs.bilkent.edu.tr/~akman/talks/cil
5 .. http://www.cs.bilkent.edu.tr/
6 .. http://www.cs.bilkent.edu.tr/~guvenir/courses/CS315
7 .. http://www.cs.bilkent.edu.tr/~guvenir/courses
8 .. http://www.cs.bilkent.edu.tr/~guvenir/courses/CS558
9 .. http://www.cs.bilkent.edu.tr/~guvenir/courses/CS550
10 . http://www.cs.bilkent.edu.tr/~aykanat

Figure 5.4 A Sample Output of the Recommendation Engine

Chapter 6

Conclusions and Future Work

In this thesis, we have presented a new usage based personalization system. The system developed has two main modules, Log Analyzer and Recommendation Engine. Log Analyzer runs off-line to mine the access log file to determine the user and session information. The other module, Recommendation Engine, uses the Result and IndexFile updated everyday by the Log Analyzer module to make dynamic recommendation to the visitor by getting its identity information by the help of the PHP scripting language and sending this information to a CGI program.

The main idea of the system is to guess the next access page of the visitors based on their past experiences and recommend these pages dynamically to the visitor. In static web sites, the visitors are limited to access to the pages they are interested in only by retrieving a number of pages consecutively. But, recommending of these pages to the visitor dynamically makes retrieving of the desired information easier by just clicking on the recommended link. Assuming that each visitor has different interests, the system processes the identity information of the visitor before recommendation. So, the recommendation set is found according to the past experiences of the user visiting the site.

One of the properties of the system is the ability to hold all session and identity information in a manageable way. Increasing the amount of the session

information belonging to the identities can increase the success of the recommended pages. But, the solution is not easy because the time needed for the execution of the Recommendation Engine to find the pages to be recommended to the visitor is critical. The size of the Result_File holding the session information must be chosen reasonable so that the Recommendation Engine produces the set of recommendations in the least time without boring the visitor. Because, most of the visitors behave impatiently that they are dissatisfied with too much loading time of the page into the client's machine. But, as the number of the visitors gets larger, it makes keeping all session information in such a limited size of file more difficult. In other words, we have to minimize the amount of session information without decreasing the success of the recommendations.

One of the problems we encountered is to determine the unique users. As discussed in 3.1.3, especially proxy servers which can be thought as a gateway to World Wide Web for many users make the determination of unique users more difficult even sometimes impossible. All visitors coming from the same proxy server are seen as if they all have the same identity information. To reduce the effect of this problem, we have used the "Agent" field of the entries. So, the identity information has been accepted as a pair of "IP Address and Agent". As a result, we had an ability to distinguish unique users who use the same proxy server with different browsers.

The other problem was the behavior of the visitors who visit the site without any goal. These visitors retrieve the pages in the site randomly without any idea in their mind. As a result, the session information of these kinds of visitors reduces the success of the recommendation. This problem is also related to a given coffee break. That is, the visitor gives himself/herself a coffee break while surfing the web site at the time looking at a page that may not be interesting for her/him. Because our main idea is to recommend the pages that have been spent much more time than the others to the visitor, that given coffee break increases the score of that page however it may not a content page and consists of any related information.

The experiments conducted indicate that the most time consumption occurs in the execution of the Log Analyzer module. But this drawback may not be considered as a big problem because the Log Analyzer module runs off-line and does not have any influence on the success of the Recommendation Engine. The factor affecting the success of the Recommendation Engine is the size of the Result_File created by the Log Analyzer module. But, the size of the Result_File is under the control of the Forgetting Algorithm. It checks the size of the Result_File at every end of the execution of the Log Analyzer module and if the size of the Result_File exceeds the threshold that has been chosen reasonable for the success of the online module, the Forgetting Algorithm starts and eliminates the pages which the visitors retrieved them for a short time period. At every end of execution of the Forgetting Algorithm, the size of the Result_File is reduced to a reasonable limit determined by the web master.

We foresee that if a particular visitor shows some trends on his/her navigational behavior, the visitors belong to the parent domain of the visitor will also be inclined to show the same trends. So, the system firstly tries to guess the behavior of the visitor by looking at the sessions belonging to the visitor considered. But, if the visitor is new for the system, then the identity information of the visitor is parsed to obtain its parent domain and the pages to be recommended are searched in the sessions belonging to its parent domain. It is clear that the success of the recommendation based on the past experiences of a particular visitor is higher than that of the parent domain of the visitor considered.

The experiments conducted show that the success of the pages recommended to the visitors is satisfactory. But it is nearly impossible to say that the success is 100% in the real life. But, observing the number of times in which the visitors utilize our recommendation can measure the success of the recommendation. If the recommended pages are really interesting for the visitors, they will follow the recommended links instead of the static links on the page.

The experiments also showed the importance of preprocessing of the entries before inserting them to the session information. As shown in Table 5.1, approximately the half of the total entries in the access log file seems as irrelevant items that cannot be an input to the Log Analyzer module. All irrelevant entries (including image, sound etc) are detected by the Log Analyzer module and eliminated just before the session identification phase. In the future, if the configuration of the web server permits, the web server can be configured in such a way that it does not record the entries including irrelevant items in the access log file. In that case, approximately all entries in the access log file will be relevant items, so the cost of the preprocessing phase can be automatically reduced.

In the future, the system may be extended by the concept of “Information Retrieval System”. In the thesis, we have assumed that a page in which the visitors spent much more time than the others is a *content* page, otherwise a *navigational* page. We think that the success of the recommendation will be higher if a procedure, that analyzes the content of a web page and produces a score for each page to categorize it as a *content* page or *navigational* page by evaluating the frequencies of the terms in the page, is added to the module. In that case, Recommendation Engine will show the pages in which the visitors spent much more time than the others and also be categorized as *content* page by the procedure that analyze the content of the page. We believe that the success of the recommendation set will increase after such a procedure is added to the module.

Also, assuming that the same module detects all static links on the pages while analyzing the content of the page, we have an ability to understand the link structure of the site. By using the link structure of the web site, Recommendation Engine shows a recommended page if the active page has no static link to the recommended page. Otherwise, if a link exists on the page to the recommended page, existing link may be lightened to attract attention to the importance of the page.

At the same time, the system may be modified in such a way that it may use an extra index structure formed for the pages in the site. In the existing system, only one index structure is used to find the position of the sessions belonging to the visitors. But this index structure may be extended to find the position of the active page in the sessions. As a result, the system will have an ability to know the position of the active page in the sessions beside the start and the end positions of the session section belonging to the visitor. If an extra index for the active page were available, then the Recommendation Engine does not need to find the active page in the sessions, it automatically redirects itself to the position of the active page in the sessions.

As a conclusion, a system is developed as a starting point and improved due to the interests and needs of human being. However, the success of the system we designed and implemented is satisfactory, it is open to be improved by future works and we believe that our system will inspire the people who are desirable to develop applications on web usage mining.

Bibliography

- [1] R. Cooley, B. Mobasher 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
- [2] C. M. Brown, P. B. Danzig, D. R. Hardy, U. Manber and M. F. Schwartz. The Harvest Information Discovery and Access System. In *the 2nd International WWW Conference, (Chicago, IL, 1994)*, 763-771
- [3] E. Spertus. Parasite: Mining Structural Information on the Web. In *Computer Networks and ISDN Systems*, 29: 1205-1215,1997
- [4] R. B. Doorenbos, O. Etzioni and D. S. Weld. A scalable comparison shopping agent for the World Wide Web. In *Proceedings of the 1st International Conference on Autonomous Agents*, pp 39-48. New York, NY.1997, ACM Press
- [5] W. B. Fakes and R. Baeza-Yates. Information Retrieval Data Structures and Algorithms. In *Prentice Hall Englewood Cliffs, NJ*, 1992
- [6] T. Joachims, D. Freitag and T. Mitchell. WebWatcher: A Tour Guide for the World Wide Web. In *Proceedings of 15th International Joint Conference on Artificial Intelligence*, pages 770-775. Morgan Kaufmann, Aug.1997.
- [7] M. Pazzani, J. Muramatsu and D. Billsus. Syskill&Webert: Identifying interesting web sites. In *Proceedings of the 13th National Conference on Artificial Intelligence, Portland, OR*, 1996
- [8] D. Konopnicki and O. Shmueli. W3QS: A Query system for the World Wide Web. In *Proceedings of the 21st International Conference on Avery Large Databases*,
pages 54-65,1995

- [9] J. Srivasta, R. Cooley, M. Deshpande and P. Tan. Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data. In *SIGKDD Explorations*, (1) 2, 2000
- [10] S. Chakrabarti, B. Dom and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *SIGMOD Conference, ACM, 1998*
- [11] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *the 7th International WWW Conference, (WWW7)*, pp. 107-117, Brisbane, Australia
- [12] J. Borges and M. Levene. Data Mining of User Navigation Patterns. In *Web Analysis and User Profiling*, Volume 1836, pages 92-111. *Lecture Notes in Computer Science*, 1999
- [13] Web Site Traffic Analysis Tool. Online at <http://www.softseek.com/Internet/>
- [14] Web Statistics Software. Online at <http://www.openwebscope.com/>
- [15] J. Pitkow and P. Pirolli. Mining Longest Repeating Subsequences to predict World Wide Web Surfing. In *Proceedings of the 1999 USENIX Technical Conference*, April 1999
- [16] H. Lieberman. Letizia: An Agent That Assists Web Browsing. In *Proceedings of the 1995 Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.
- [17] Esra Satiroğlu. Mining User Access Patterns and Identity Information From Web Logs For Effective Personalization. *Master of Science Thesis submitted to Bilkent University*, September 2001.
- [18] M. Spiliopoulou, L.C. Faultich. WUM: A Web Utilization Miner. In *Proceedings of EDBT Workshop WebDB98*, Valencia, LNCS 1590, Springer Verlag, 1999
- [19] B. Mobasher. WebPersonalizer : A Server Side Recommender System Based on Web Usage Mining. In *Technical Report TR-01-004*, March 1991.
- [20] D. Aha and D. Kibler. Instance-Based Learning Algorithms. *Machine Learning*, 6, 37-66, 1991
- [21] C. R. Cunha, C. F. B. Jaccoud. Determining WWW User's Next Access and Its Application to Pre-fetching. In *Proceedings of the International Symposium on Computers and Communications'97, Alexandria, Egypt*, July 1997.

- [22] T. Nakayama, H. Kato and Y. Yamane. Discovering the Gap Between Web Site Designers' Expectations and Users' Behavior. In *the 9th International WWW Conference, Amsterdam*, May 2000.
- [23] F. Masegla, P. Poncelet and M. Teisseire. Using Data Mining Techniques on Web Access Logs to Dynamically Improve Hypertext Structure. In *ACM SigWeb Letters*, 8(3): 13-19, October 1999.
- [24] M. Perkowitz, O. Etzioni. Adaptive Web Sites: Automatically Synthesizing Web Pages. In *Communications of the ACM*, 43(8): 152-158, 2000
- [25] T. W. Yan, M. Jacobsen, H. Garcia-Molina, U. Dayal. From User Access Patterns to Dynamic Hypertext Linking. In *Computer Networks*, 28(7): 1007-1014, May, 1996
- [26] Z. Su, Q. Yang, H. Zhang. WhatNext: A Prediction System for Web Requests using N-Gram Sequence Models. In *First International Conference on Web Information Systems and Engineering Conference*. Hong Kong, June 2000.
- [27] S. Schechter, M. Krishnan, M. D. Smith. Using path profiles to predict HTTP requests. In *Proceedings of the 7th International WWW Conference*, 1998
- [28] R. R. Sarukkai. Link Prediction and Path Analysis Using Markov Chains. In *the 9th International WWW Conference*, 2000.
- [29] L. Chen, K. Syra. WebMate: A Personal Agent for Browsing and Searching. In *Proceedings of the 2nd International Conference on Autonomous Agents*, 132-139.
- [30] K. Wu, P. S. Yu and A. Ballman. SpeedTracer: A Web Usage Mining and Analysis Tool. *Internet Computing*, 37(1): 89, 1997.
- [31] WebTrends Log Analyzer. Online at <http://www.webtrends.com>
- [32] R. Cooley, B. Mobasher, J. Srivastava. Data Preparation for Mining World Wide Web Browsing Patterns. In *Knowledge and Information Systems*, 1(1): 5-32, 1999
- [33] 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
- [34] M. S. Chen, J. S. Park and P. S. Yu. Efficient Data Mining for Path Traversal Patterns. In *Knowledge and Data Engineering*, 10(2): 209-221, 1998

- [35] A.G. Büchner and M. D. Mulvenna. Discovering Internet Marketing Intelligence through Online Analytical Web Usage Mining. In *ACM SIGMOD Record*, 27(4): 54-61, 1998
- [36] D. S. W. Ngu and X. Wu. SiteHelper: A Localized Agent that Helps Incremental Exploration of the World Wide Web. In *the International Journal of Computer and Telecommunications Networking*, 30, 1997
- [37] A. Bestavros. Using Speculation to Reduce Server Load and Service Time on the WWW. In *Technical Report TR-95-006, Computer Science Department, Boston University*, page 15, February 1995.
- [38] V. N. Padmanabhan and J. C. Mogul. Using Predictive Prefetching to Improve World Wide Web Latency. In *Proceedings of ACM SIGComm, 1996*, pp 22-36, 1996.
- [39] J. Pei, J.Han, H. Zhu and B. Mortazavi-asl. Mining Access Patterns Efficiently from Web Logs. In *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'00)*, pages 396-407, Kyoto, Japan, April 2000.
- [40] O. R. 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
- [41] L. D. Catledge, J. E. Pitkow. Characterizing Browsing Strategies in the WWW. In *Proceedings of the International Conference on the WWW, Darmstadt 1995*