An Intelligent Web Page Ranking based on User Preferences

A Graduate Project submitted to
The Faculty of Computer Science Department
Lamar University - Beaumont, TX

In Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Submitted
To
Dr. Chung-Chih Li

By
Sookanan, Parnu
Misirli, Beytullah
Alicli, Etem Emrah
Nandakumar, Nagaraj
Khan, Md. Shawkat Ali

December 2003

COMPUTER SCIENCE DEPARTMENT
LAMAR UNIVERSITY, BEAUMONT, TX

An Intelligent Web Page Ranking based on User Preferences

Sookanan, Parnu
Misirli, Beytullah
Alicli, Etem Emrah
Nandakumar, Nagaraj
Khan, Md. Shawkat Ali

Approved:

Dr. Chung-Chih Li
Supervising Professor

Dr. Lawrence Osborne
Committee Member

Dr. Dehu Qi
Committee Member

Dr. Lawrence Osborne
Chair, Department of Computer Science

ACKNOWLEDGEMENTS

We express our heartfelt gratitude to Dr. Chung Chih Li whose priceless encouragement, guidance and support allowed this project to be a success. We would like to thank him for supervising and providing suggestions and solutions to many of the problems that we faced while working on this project.

We would like to thank our other committee members, Dr. Lawrence Osborne and Dr. Dehu Qi, for their guidance and helpful suggestions. We also thank all the other faculty members of the Computer Science department for all the knowledge we acquired during our stay here at Lamar University.

We dedicate our endeavor to our parents, whose boundless love and blessings helped this project to be a success. Finally, we thank all our friends for their support and encouragement.

ABSTRACT

Currently available PageRank algorithms are based on the linkage structure of the web graph. To refine the ranking result we should depend not only on the linkage structure but also on the user's preferences. In this project, we first crawl a specific web domain to construct a web graph. Second, we implement a user interface to catch the user preferences by reading the bookmarks of the user, the history, and the list of important pages selected among the hub pages by the user. Finally, we calculate the Personalized PageRank Vector (PPV) using the graph and the collected user preference. We present experimental results demonstrating the effectiveness of the idea and of the personalized PageRank.

Table of Contents

CHAPTER 1

INTRODUCTION

The dramatic increase in the number of web pages has made the problem of information overload more severe when browsing and searching the World Wide Web. The World Wide Web also poses many new challenges for information retrieval. The current WWW has been very large, heterogeneous, and diverse. Consequently, it is very difficult to present the information in the web that is most relevant to the users. The expansion of the web has forced a greater reliance on client-side filtering and relevance analysis of the web pages, and this in turn makes the ranking of web pages based on user preference essential. Two systems, Personalization Systems and Recommendation Systems [19], are popular remedies by which the web environment can be customized based on user preference. Unlike Recommendation Systems, Personalization Systems have received little attention from the e-commerce domain; even though search engines have become indispensable tools for web surfing. By taking the user's preferences into consideration, search engines can remove a large amount of irrelevant results. With some careful analysis, irrelevant results can be filtered out or made to lose their importance in the user's personal context. In other words, user preference will take a part in the ranking process.

1.1 Purpose and Idea

The aim of this project is to rank the web pages in a given specific domain according to a specific user's preferences. Each web page in the given domain is initially assigned a rank based on the PageRank Algorithm [2]; the algorithm exploits the linkage structure of the web to compute importance scores. The rank assigned by the PageRank Algorithm reflects a democratic importance, namely, we do not assume any preference for any particular page. A new rank is then re-calculated for each web page using an approach discussed in [1]. The user preferences are obtained from three different sources:

(i). the user's bookmark (favorites) list,

(ii). n, a pre-specified number of unique URL's from the "History" folder, which provides the user's browsing history, and

(iii). a set of user preferred pages selected from a set of Hub pages, which is the list of pages that we want to rank [3].

The URLs' obtained from these three sources together provide user preferences:

(i) The static preferences are those, which do not change without user interference, such as bookmarks.

(ii) The dynamic preferences are those, which change in time without user interference, depending on user surfing behavior, such as histories.

Thereby, we are able to perform an intelligent Web Page ranking based on user preferences. The experimental results of this project show the advantages of ranking the Web Page based on User Preference.

1.2 Motivation

The sensational expansion of the World Wide Web, now almost a billion pages, is forcing existing web search engines to look beyond the superficial content of pages in providing meaningful answers to user's queries. Recent work in utilizing the link structure of the web for improving the quality of search results is promising. For example, the "Google" search engine uses "PageRank," which is an iterative algorithm that determines the "importance" of a web page based on the "importance" of its parent pages. The

PageRank algorithm measures the importance of a WebPage with respect to other WebPages by computing a global rank for every page based on the web graph. This generalized link structure, however, does not consider user preference. The principle underlying PageRank is that important pages are those referenced by many important pages like Yahoo!, Netscape, AltaVista, etc. We can formalize this recursive notion by using the random surfer model discussed in [2]. Still, the deep structure of the Web does not reflect every individual's need. Within the huge amount of web pages, users may want to personalize their search results. In this case, Google's global notion for importance can be specialized to create a personalized view of importance, where the importance score is biased according to a user specified set of initially interesting pages and the user's browsing history.

A user may have a set of preferred pages *P* that he considers more interesting. We can account for the pages in *P* in the random surfer model using a "Teleportation Probability" c, which represents the probability for a surfer to jump back to random pages in *P* and thereby calculate the Personalized Page rank Vector (PPV) [1]. The

PPV is a personalized view of the importance of pages on the web.

On one hand, experiments with the use of personalized page Rank have shown its utility and promises. On the other hand, the size of the web makes the implementation of this concept extremely difficult. This project is a scaled down version of the above idea, where the web pages in a "particular web domain" are ranked based on users' preferences.

1.3 Organization of the Report

The report is divided into five chapters followed by the Bibliography and Appendix.

1. The first chapter is an introduction to the purpose of the project, the motivation that fueled the project, and the organization of this report.
2. The second chapter discusses the research that was done prior to the present project in order to gain in-depth background knowledge of the principles and concepts involved in this project. This is further divided into three sections. The sections describe some basic concepts in Machine

Learning, Web Search, the relation between Machine Learning and Web Search, PageRank, and the role of User Preferences in biasing PageRank.

3. Chapter three presents the Methodology of this project. The chapter has been categorized into three sections. The first section presents the detailed algorithms, protocols and the flowchart of the various processes. The second section describes the programming environment used, the principle of the programming concepts, and the implementation details. The specifications of the project are discussed in the third section.
4. The fourth chapter discusses the experiments conducted and the results obtained from them.
5. Chapter five provides a conclusion of the project and suggestions for future work in this area of Web page ranking.
6. Appendix A provides partial source code for different modules. Appendix B provides Screen Shots.

CHAPTER 2

BACKGROUND RESEARCH

2.1 Machine Learning [5]

Before discussing machine learning, let us at first survey the idea of learning. Many people have defined learning in different ways. Learning is the use of experience to change the system so that the system can do the job more efficiently in the future. Fig.2.1 shows the components of a learning system; here arrows represent the direction of information flow:

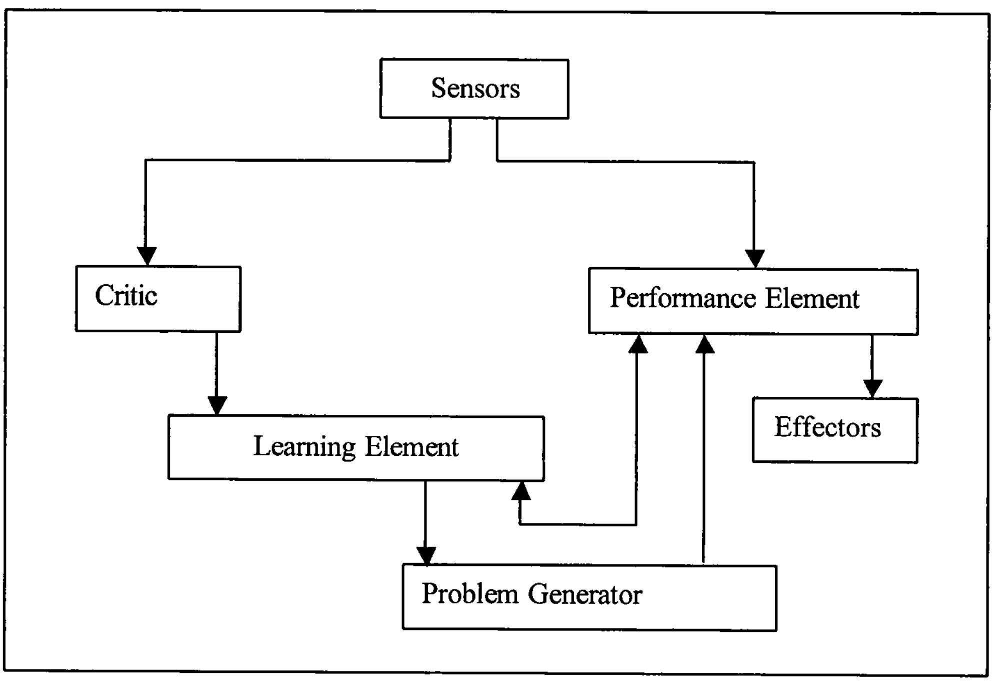


Fig. 2.1 Components of Learning System

1. The Learning element formulates modifications to the system.
2. The Performance element is the negotiator that takes action in the real world.
3. Comparing with a fixed standard of performance, the Critic informs the Learning element about its performance.
4. The Problem Generator creates "problems" or action that will develop new examples or experiences.

Machine learning refers to a system capable of the self-governing attainment and incorporation of knowledge. By learning from experience, analytical observation and other means, a system can continuously self-improve. Thereby, the system offers increased efficiency and effectiveness for future activities [11].

There are several reasons why machine learning is important. Some of these are: [7, 8]

1. For some problems or tasks, it is possible to specify input/output pairs, but it is not possible to give a concise relationship between the input and expected outputs. In this case, a machine can

draw suitable constraints relating to the input/output functions, in order to approximate the relationship implied in the example, which has the ability to adjust its internal structure to produce correct outputs for a large number of sample inputs.

2. Machine learning can discover the important relationships and correlations that are hidden among large piles of data.
3. Traditional machine design methods can be replaced by Machine learning methods that can be used for on-the-job improvement of existing machines.
4. Explicit encoding of large amounts of knowledge by humans is almost impossible, but machines might be able to acquire most of it by gradual learning.
5. Redesigning AI (Artificial Intelligence) systems with the rapidly changing situation in the world is impractical, but machine-learning methods can adopt this kind of situation easily.

To make computer systems adjust and learn from their previous knowledge is the primary goal of machine learning. The machine-learning paradigm offers an alternative to the concept of traditional software engineering, which uses human-intensive hand engineering with data-intensive software construction and testing.

Machine learning and other disciplines [8]

Several disciplines have contributed to the development of machine learning, and it is now converging from several sources. Some of the most important disciplines that contributed to machine learning are:

1. **Artificial Intelligence:** From the beginning, AI research has been associated with machine learning. The role of analogies in learning and how future actions and decisions can be based on previous exemplary cases have also been explored by AI researchers.
2. **Evolutionary models:** In nature, all individual animal species learn to perform better and evolve to be better fit in their individual niches. Genetic algorithms and genetic programming are the most famous computational techniques for evolution.

Machine Learning in Web Search [8]

Web search engines have become essential tools to locate information on the ever-growing World Wide Web. Due to the dynamic nature of surfing and contents of the World Wide Web, Machine-learning algorithms can dynamically improve retrieval quality in web search. Particularly, these algorithms emphasize and record the links that a user clicks on, in ranking the web pages for each query. Utilizing this information and learning an improved ranking function is the goal. When machine learning is used in search engines, it can adapt to the preferences of user groups, particular users, and the dynamic properties of a particular collection without expert parameter tuning. To quickly find a document in the World Wide Web, we should have the ability to construct queries that can find a wide range of document language and filter out a large proportion of the unrelated documents. Automated systems for learning from the web have primarily been installed in crawlers or spiders. Using machine-learning techniques, we can adopt a new generation of algorithms to augment the set of search capabilities, and by combining other kinds of topic or target-directed searches; we can make a search engine more efficient and usable.

2.2 Web Search

The World Wide Web is a vast collection of completely uncontrolled heterogeneous documents. Documents on the web have extreme variations internally and also in the external Meta information that might be available. Searching for information of interest, in practice, is impossible if its source is unknown in advance. This growing WWW creates many new challenges for information retrieval. The number of new inexperienced users is also increasing with the growth of information on the web. Web search engines allow the user to enter keywords as a query to retrieve those web pages that match the query term. Unfortunately, these search engines tend to retrieve many irrelevant web pages in their lists. Consequently, a user needs to select useful pages from the list and fetch the web pages to investigate their relevance. The key issue in web search is how to efficiently and precisely retrieve the most relevant pages for its results.

Algorithms used in Web Search Engine [9, 10]

There are many algorithms used in web search engines. Some of the most efficient algorithms are:

I. **Companion Algorithm:** [10] The companion algorithm takes as input a starting URL 'u'. The algorithm consists of four steps:

   I. Build a vicinity graph for 'u' [10]. Vicinity graph is a directed graph (V, E) where V is a set of nodes representing separate web pages and E is the set of edges representing the links between these nodes.

II. Combine the similar nodes [10], which have at least 95% of their links in common.

III. Compute edge weight based on host-to-host connections.

IV. Compute the number of out going links from any node (hub score) and the number of in coming links from the other nodes to this node (authority score) for each node in the graph and return the top ranked authority nodes [10].

II. **Cocitation Algorithm:** [10] An alternative approach for finding related pages is to examine the siblings of a starting node 'u' in the web graph. Two nodes are co-cited if they have a common parent. The number of common parents of two nodes is their degree of co-citation. There are few steps in this algorithm:

I. First, choose a pre-specified number 'B' of arbitrary parents of 'u' as shown in Fig. 2.2.

II. For each parent 'p', it adds to a set 'S' up to a specified number 'BF' of children of p that surrounds the link from p to u. The elements of S are siblings of u.

III. For each node 's' in S, it determines the degree of co-citation of s with u.

IV. Finally, the algorithm returns the 10 most frequently co-cited nodes in S as related pages.

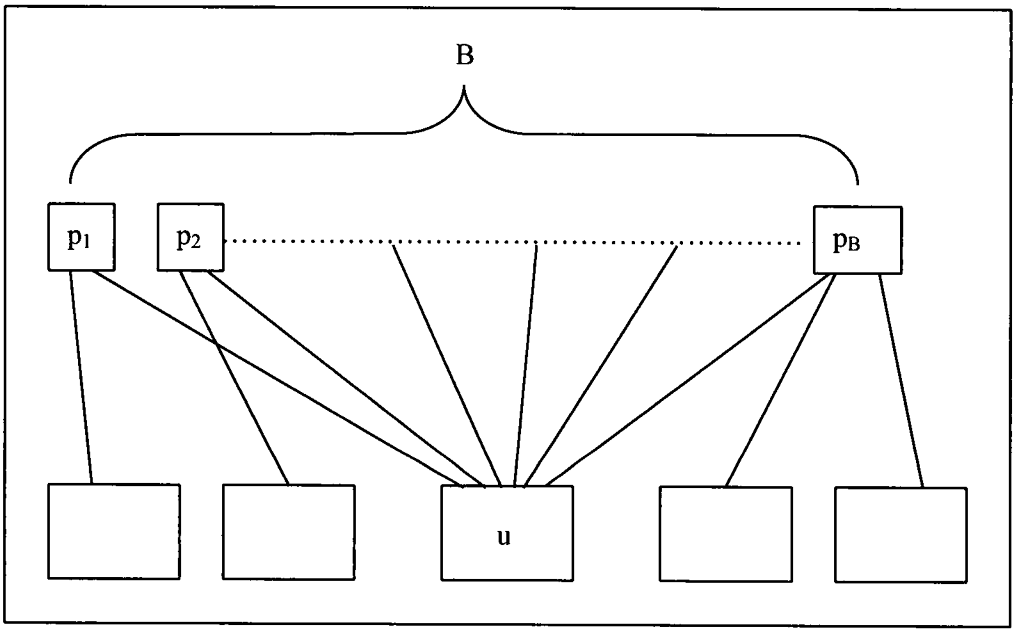


Fig. 2.2 Web model used Cocitation algorithm

3. **Google's Page Rank Algorithm:** [2] The Link structure based algorithm has proven to be the most efficient one at present time. The page rank algorithm greatly improves the results of a Web search by taking into

account the link structure of the web. Each web page is assigned a score proportional to the number of times a random surfer would visit that page if he/she surfed indefinitely from page-to-page, following all out links from a page with equal probability. Before discussing the computation of page rank, we first consider the link structure of the web.

I. Link Structure:

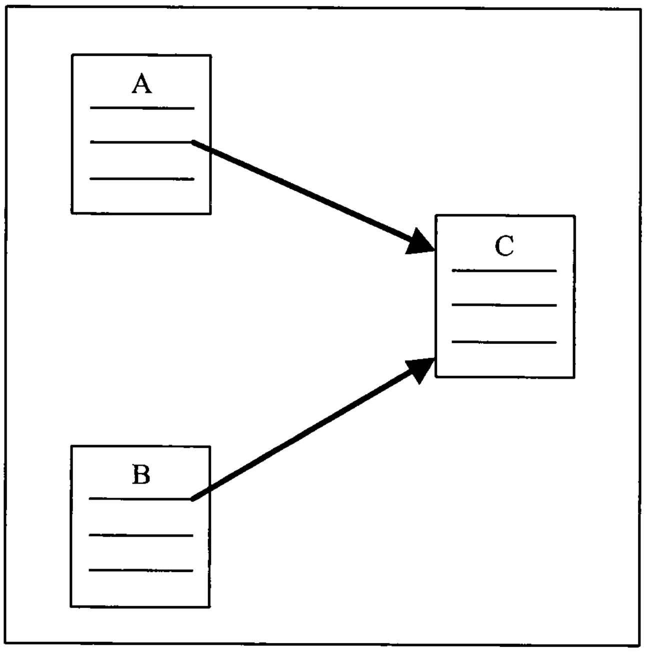


Fig. 2.3 Example of Link Structure

Every page on the graph of a crawl-able web has some number of forward links also known as out-links of the page p, represented as O(p), and backward links, also known as in-links of the page p, represented as I(p). For example, in Fig. 2.3, node C has two backward links, which come from node A and B. Also, node A and B each have a forward link to node C. It is not possible to know all the backward links of a particular page, but after downloading, we may know its entire forward links. Different web pages have different numbers of backward links. In general, pages with more backward links are considered more important than pages with fewer backward links.

II. PageRank: In short, PageRank is a "vote", by all other pages on the web, about how important a page is. A backward link of a page count as a vote of support. If there is no backward link, there's no support. Let u be a web page, F_u be the set of pages u points to and B_u be the set of pages that point to u. Let $N_u=|F_u|$ be the number of F_u, and let c be a constant less than one. Then R(u), PageRank of u is defined as:

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v} \quad \text{................(2.1)}$$

The above equation is recursive, starting with any set of ranks and iterating the computation until it converges. The Fig.2.4 shows a simplified PageRank calculation according to equation $R(u) = \sum R(v)/N_v$, which is a simplified version of the equation (2.1). Each box in the Fig. 2.4 represents a web page. The number shown in the box represents the page rank of that web page according to the simplified version of the equation.

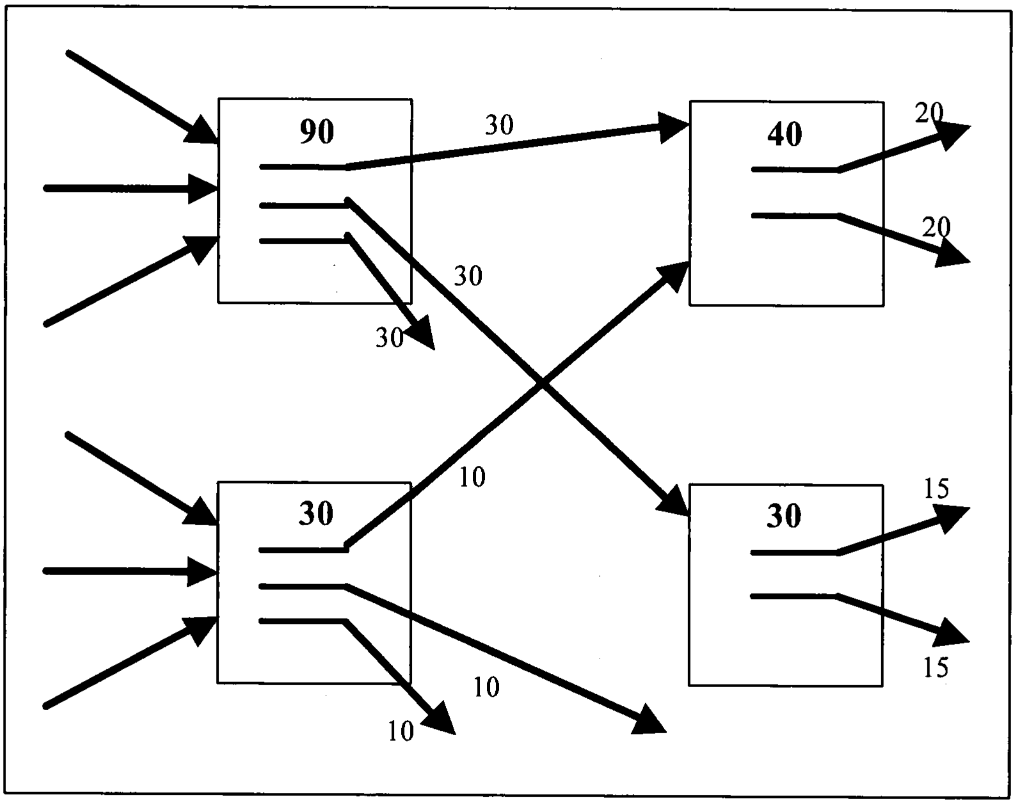


Fig. 2.4 Simplified PageRank Computations

Google [www.google.com] has implemented the idea of PageRank with the architecture overview shown in Fig. 2.5: [9]

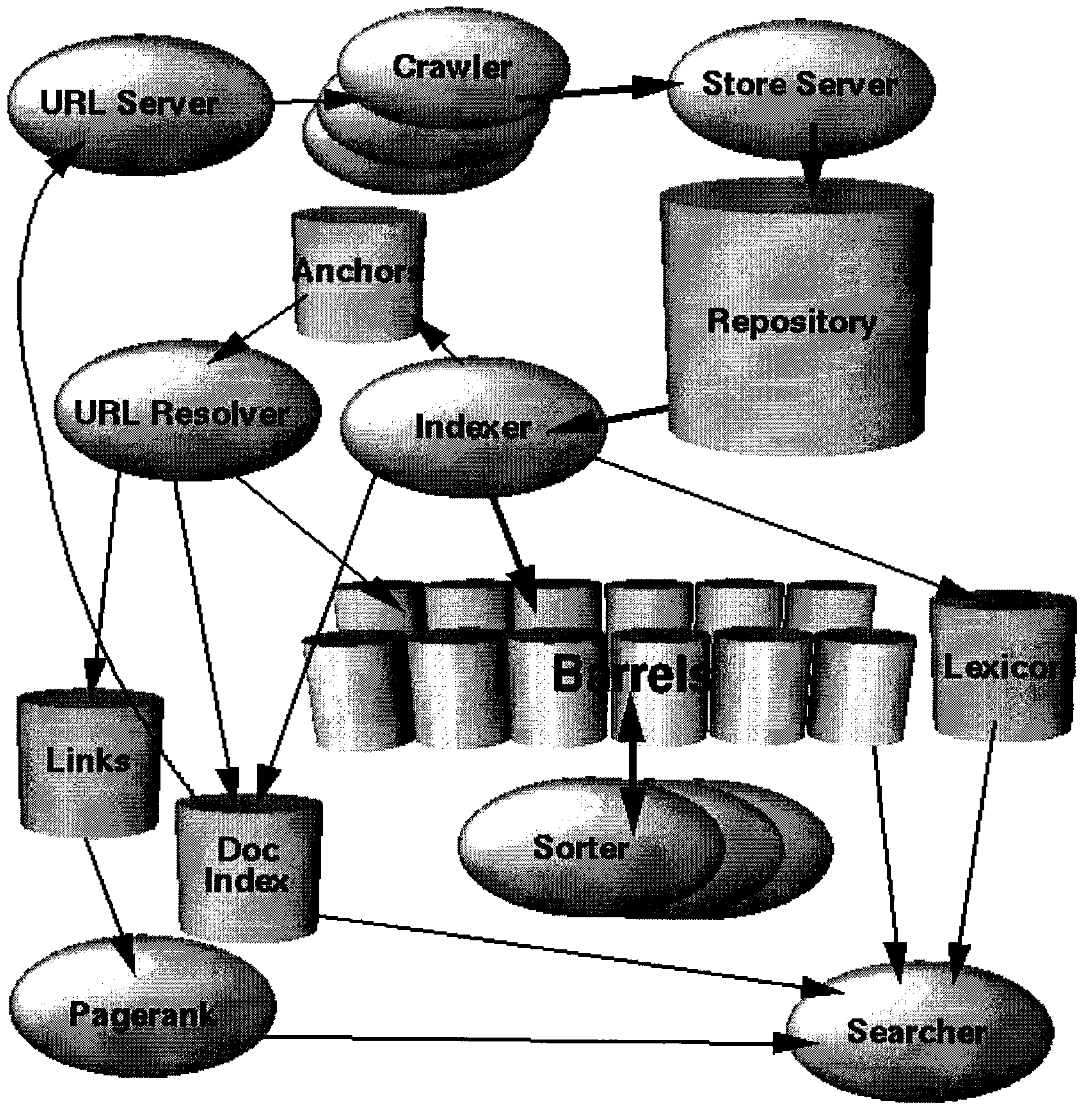


Fig. 2.5 Google architecture

1. **Crawler:** Several distributed crawlers crawl the web and are responsible for downloading web pages.
2. **URL Server:** An URL server sends a list of URLs to the crawlers, and then crawlers fetch the web pages on that list.

3. **Store Server:** The Store Server is the repository used for storing the web pages fetched by the crawler.
4. **Indexer:** Indexers perform the indexing function. The Indexer also reads the repository, un-compresses the documents and parses them. Each document is then converted to hits, a list of word occurrences including their position in the document, font etc. After creating the hits, the indexer distributes these hits into a set of barrels; it parses out all the links in every web page and stores important information about them in a file, which contains enough information to determine where each link points from and to.
5. **URL Resolver:** The URL Resolver reads the anchor file and converts a relative URL to an absolute URL. It also creates the database of links that are used for computing the PageRank of all documents.
6. **Sorter:** The Sorter takes the barrels, which are sorted by DocID, and re-sorts them by WordID and generates the inverted index. It also produces a list of WordIDs and offsets into the inverted index.

7. **DumpLexicon:** The DumpLexicon is a program that takes the list of WordIDs and lexicon from the indexer and generates a new lexicon for the searcher.
8. **Searcher:** The Searcher is run by the web server and uses the lexicon with the inverted index and the PageRank to answer the queries.

The "PageRank" algorithm employed by "Google" has been proposed and implemented in order to compute the linkage structure of the World Wide Web for years. This algorithm will exploit the structure of the web graph as global importance scores, which will be considered as the ranking weight for the pages.

2.3 User Preferences in Page Ranking

Since there is an enormous amount of information available on the WWW, users using general search engines to search for information on the web with specific categories may have a difficult time getting a specific piece of information. [12]

A user may be given a huge number of pages related to his search topic but none of the desired category. For example, if the user wants to search for a photograph of the golf player Tiger Woods, the user may give "Tiger Woods" to the search engine. Imagine how many pages related to "Tiger" and "Woods" will pop up as a result of the search.

Considering the user preference is a reasonable approach to decrease the worthless pages, because different users will have unlike desires and perspectives of the searching document. In fact, user interests and preferences are actually more important than the static web structure in this situation. In the previous example, if some pages related to the photographs of Tiger Wood are stored in the user preference set, then retrieved pages related to the preference will be considered more important to the user. Such decisions, however, are not always easy to make. For example, a user looking for current events such as the news might prefer newer or updated documents rather than older documents, while another user who is looking for the history or origin of something may prefer the older or original versions.

This inclusive idea of importance can be specialized to create personalized views of importance. There are two approaches that can be used to adapt the idea of user preference.

1. Query modification: The Concept of query modification used for years is an obvious approach to solve the problem of locating the best documents. In query modification, the user modifies the query with a help of logical conjunction and little keywords to polish the search result. Changing terms, removing terms, or adding extra terms will be processed to the original query with logical operation. The goal is to adjust the query closely considering the user's wish. [12]
2. The ranking policy for the results will be modified based on the important scores biased according to a user specified set of initially interesting pages. For example, bookmarks or search history may be considered as a set of preferred pages, so any results that are important with respect to the preferred pages would be ranked higher.

"Personalized PageRank" is introduced in order to include users' preferences to the original "PageRank" algorithm. The Personalized PageRank will put more emphasis on the users' preference sets as the bias to the scores than the global importance scores.

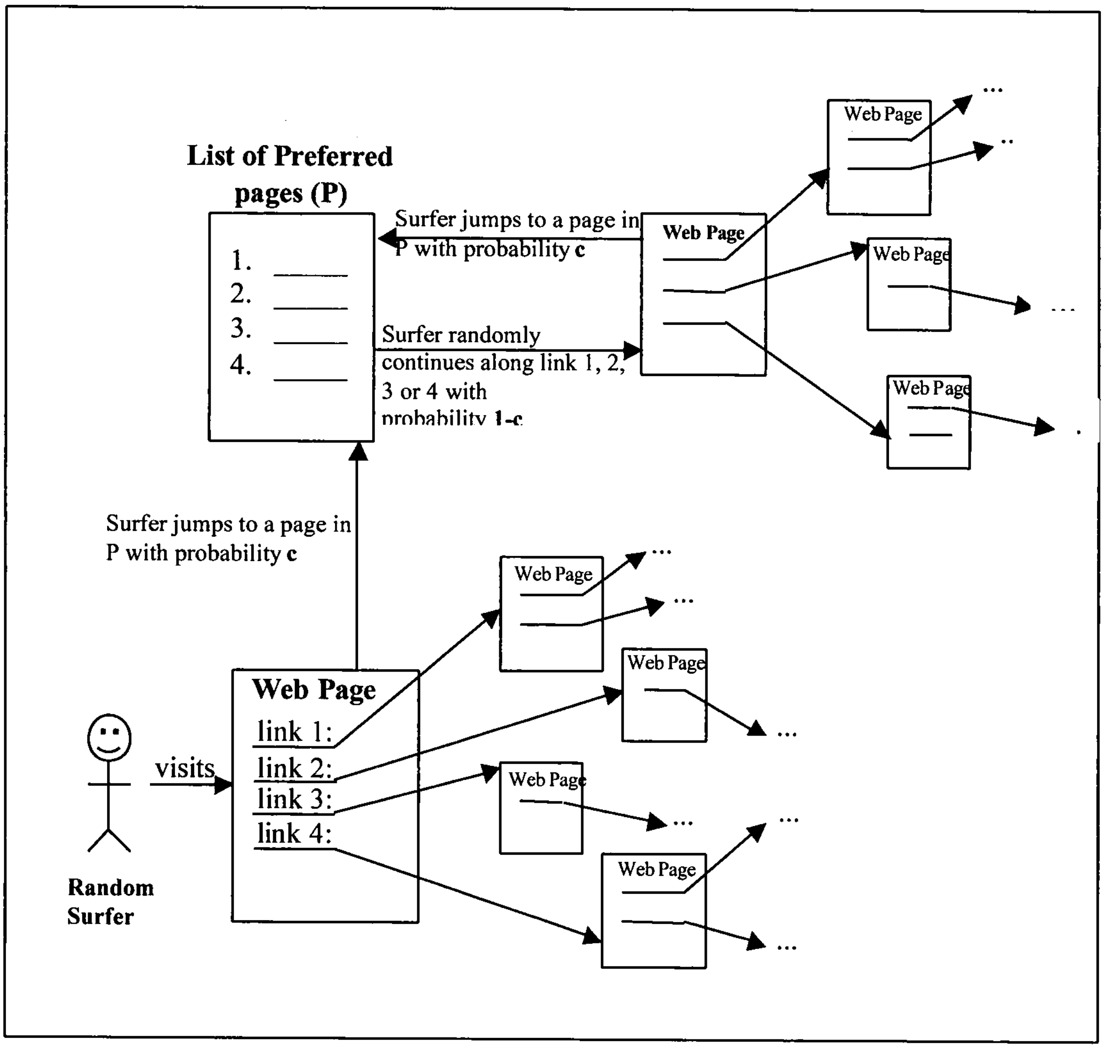


Fig. 2.6 Random Surfer Model

As described in section 1.2, we can account for the preferred pages in the random surfer model using the "Teleportation Probability" c. Consider Fig. 2.6, the distribution of surfers in this model favors pages in P, pages linked-to by P, pages linked-to in turn.

The Fig.2.6 shows a random surfer visiting a Webpage with four links. The surfer could either follow the links to other web pages (those that do not contain information related to his/her preference) or jump to a random page in the set of preferred pages (P). This jump, to a random page in 'P' is based on the probability 'c'. The random surfer could continue surfing on the links in the preferred page (with probability '1-c') or jump back to a page in 'P' (with probability 'c'), if the visited page is not of relevance. The figure tries to show how WebPages linked by other preferred pages are favored and thereby get more visitors. We can also see that 'c' increases the chances of a random surfer visiting a preferred page and, in turn, the pages it links to.

User preferences may be categorized as:

1. Static
   I. User's bookmarks.
   II. User selected pages.
   III. User homepage set as the default page.

2. Dynamic
   I. User's browsing history.
   II. The time spent on a particular page by the user visiting it.
   III. Frequency of visiting the page.

However, some of the dynamic preferences in the above list pose difficulties in calculating and managing when implemented along with the PageRank Algorithm because measurement must be dynamically updated, and the implementation is sometimes complicated. Therefore, the idea of collecting the static preference such as the bookmarks and the dynamic preference such as browsing history seem to be more feasible to achieve in biasing the results of searching.

CHAPTER 3

METHODOLOGY

The project has been divided into four logically different modules. Each part takes an input from the other modules and gives an output to others.

1. The first part is the Web Crawler (WC). It is used for collecting real world data and constructing the web graph. Web crawlers, also known as robots, spiders, worms, walkers or wanderers – are almost as old as the web itself [14]. The first crawler, named Wanderer, was written by Matthew Gray in 1993, about the same time when the first NCSA Mosaic was released. Several papers about web crawling were presented at the first two World Wide Web conferences [15, 16, 17]. People did want to explore the entire web, no matter what the size was. Today, there are many web crawlers out there trying to catch the entire web. Some have been developed for fun and some for commercial purposes. Due to the competition in this sector, the designs of these commercial crawlers have not been publicly described. There are two notable

exceptions: the Google crawler and the Internet Archive crawler.

2. The second part is the End User Module (EUM) and it collects information about the user by looking at his Favorites folder, History folder, and a list of important pages selected by the user from a predefined list of pages namely Hub Pages.
3. The third part is the Computation Module, which is the module that calculates the required page ranks based on a user's preferences.
4. Finally, the fourth part is the Module Dock, (MD) which combines all modules.

Fig.3.1 shows the relationships among the modules. We will discuss these modules in the following section. In the Algorithms and Protocols section, we explain the algorithms and protocols used in these modules and analyze their complexity. The Programming Environment section gives brief information about the development tools used in this project. Finally, in the specification section we discuss the limitations, hardware, and software specifications of the project.

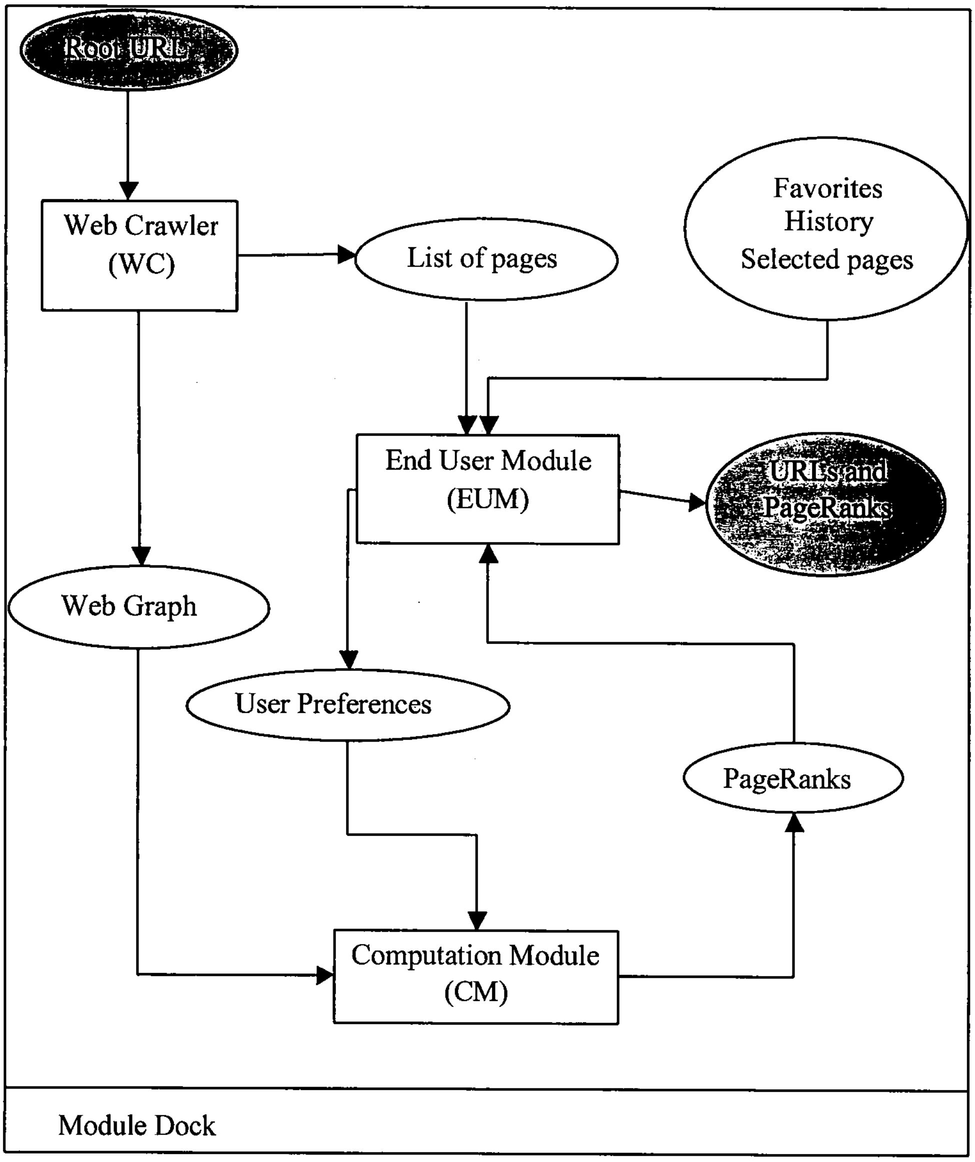
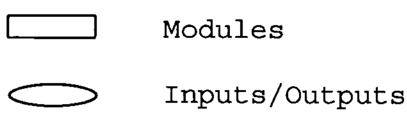


Fig.3.1 The interrelationships between the modules



3.1 Algorithms and Protocols

3.1.1 The WebCrawler

The web crawler consists of five different components.

1. The main window (TForm): The main window is the window the user sees and uses to interact with the module.
2. The thread component (TWCThread): The thread component is responsible for downloading the pages and fetching the out links in the pages.
3. The link component (TLink): The link component is responsible for generating the full URLs for the links, using the link string and the page where the link is located.
4. The URL component (TURL): The URL component analyzes the URL string.
5. Explored pages component (TmyPages): The explored pages component is responsible for storing the links that are found, storing the linkage structure and generating the output. (The Matrix A)

The TWCThread component is the main one. Each TWCThread will be running until a specified number of links (N) has been reached. The only protocol allowed is HTTP GET, and we also filter out all pages except those with htm/html extensions. Crawler does not support framed pages.

The algorithm for this component has two tasks: downloading and searching. We can omit the discussion for the downloading part of the algorithm, which is a typical component that is almost common to all the crawlers, and the performance is highly dependent on the network traffic. We can assume that on an average there are about 11 links [2] per page. The TWCThread component downloads N pages and fetches eleven links from each of the downloaded pages.

3.1.2 The End-User Module (EUM)

The End-user module takes the data from the input file created by the Web Crawler, which is a list of the links that have been explored. The file always contains 'n' links. It also reads the user's Favorite web pages, and his History folder. The user selects some pages from the list of pre-defined important pages, in our case the departments' main pages. The first output generated is the User preference vector 'u', which contains 'n' elements and it is written to a file.

After the 'u' vector is generated, the Computation Module starts executing, and it gives the ranking as an output file. EUM then reads this file and sorts it using QuickSort. Though the time complexity of QuickSort is O(nlogn), it is insignificant here because of the time complexity involved in reading the file. Therefore, time complexity of EUM depends on:

1. Downloading the WebPages
2. Matrix Calculation
3. Reading the file

3.1.3 The Computation Module (CM)

This section discusses the computational idea and the processes involved in the CM.

3.1.3.1 Global PageRank Algorithm:

The square matrix, which represents the web-graph, is created and indexed by the WebCrawler. It is necessary to rank the edges of this graph, gathering from web-pages, by using some sort of ranking algorithms in order to find the best matching results for the given query. Since the graph is extremely huge, it is essential to have a very effective algorithm to rank the results.

As explained in the previous section, the global PageRank algorithm is one of the most effective ranking algorithms today. Each page has a page-rank value that is calculated according to this algorithm, and this value takes advantage of the World-Wide-Web by using its huge global link structure as a prime indicator of a single page's value. In essence, the algorithm interprets a link from page U to page V as a vote by page U for page V. Pages that receive more votes from important pages are therefore considered more important pages with higher page-rank value. That definition in fact is a recursive definition.

As it can be seen from the equation (2.1) in section 2.2, for each page u in the web, the R(u) value is calculated by accounting for the R(v) value of each page v from the set of pages that point to u ($\forall v \in B_u$). For each such page v, the rank value of that page is divided evenly among its forward links to affect the rank values of the pages which are referenced to by this page.

Since each page's ranking value depends on the global ranking value of all other pages pointing to that page in the graph, it cannot be calculated simply by applying the values to the equation. This is a recursive equation and it requires several iterations to calculate the PageRank values of a page. After each iteration, entry values of the solution vector are calculated by using the results of the previous calculation. Therefore, each iteration generates a better solution set then the previous one. Whenever those values begin to not change significantly, it is considered that the calculation has converged to the ultimate solution, and it stops. An alternative way to calculate these values is by using linear matrix equations, which will be explained later.

3.1.3.2 Computation of Personalized PageRank

The Global PageRank algorithm never reflects any user preference. In other words, unless the linkage structure of the graph changes or some new pages, with links, are added to the graph, the PageRank values will remain constant. Therefore, capturing the browsing behavior of a user and taking this effect into account when calculating a WebPages's PageRank value could lead us to more appropriate, intelligent, and dynamically ranked results of a web search.

For example, let us say a user A mostly spends his time on the internet browsing automotive retail related websites, and another user B visits zoology oriented websites more frequently. The search results of the word "jaguar" should not be the same because, most probably, user A is looking for accessories or automotive parts for his/her vehicle, and the user B is looking for information about the animal. Therefore, in order to have a relatively smarter ranking algorithm, which will combine the effect of user behaviors and habits in browsing with the existing linkage structure of the Web; it is necessary to have a better organized PageRank algorithm for personalization.

Let G = (V, E) denote the *web graph* where V is the set of all web pages and E is the set of directed edges among the web pages. For a given page p, we will use I(p) for the set of pages which have a link to page p and O(p) for the set of pages which are referenced by page p.

We will use "n" as the number of all web pages, so each page in the graph will be labeled from 1 to n for the sake of convenience.

Instead of using the global PageRank equation 2.1, we can use a slightly different equation to compute the global PageRank values and also the personalized PageRank values of the pages. The latter value will actually be a modified version of the earlier value.

Let **v** denote the n-dimensional vector for personalized PageRank values and also preference values with the n-dimensional vector **u**. The values of these vectors will always be real numbers between 0 and 1, and the magnitude value of a vector **u** is simply the sum.

$$|u| = \sum_{i=1}^{n} u(i) \quad \ldots\ldots \quad (3.1)$$

The user defined preference set will be denoted by P where u(p) denotes the amount of preference for page p. Then, matrix-vector equations will be used to formalize personalized PageRank values.

For a given **u** preference vector, the personalized PageRank equation will be

$$v = (1-c)Av + c.u \quad \ldots \quad (3.2)$$

where v is the Personalized PageRank Vector, A is the $n \times n$-square matrix corresponding to the linkage structure of the graph G, and O(j) is the number of outgoing links of page j, where

$$A_{ij} = \begin{cases} \dfrac{1}{O(j)} & \text{, if page j links to page i} \\ 0 & \text{, else.} \end{cases} \qquad \ldots (3.3)$$

The constant value 'c' that is used in the equation is the "teleportation" constant, which was explained in the earlier sections. It is generally accepted as 0.15 in the computation of the PageRank values, but experiments show that the choice of 'c' doesn't really affect the computations in practice as long as c $\in [0, 1]$. We will try different values of 'c' in the experiments in order to demonstrate this effect.

Therefore, the personalized PageRank equation can be interpreted as:

$$\begin{bmatrix} v_1 \\ v_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ v_n \end{bmatrix} = (1\text{-}c) \cdot \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ v_n \end{bmatrix} + c \cdot \begin{bmatrix} u_1 \\ u_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ u_n \end{bmatrix} \qquad \ldots (3.4)$$

Vector u is the preference set P where,

$$|u| = 1$$

and u(p) denotes the amount of preference for that page p. In the vector u,

$$u(p) = \begin{cases} \dfrac{1}{|P|}, & \text{if page } p \in P \\ 0, & \text{if page } p \notin P \end{cases} \quad \text{.......................................(3.5)}$$

By Markov Theory [1], there is always a unique solution for vector **v**. The solution is the *personalized PageRank vector (PPV)* for preference vector **u**. The process for retrieving user preferences and building vector **u** will be explained in detail in the next section.

Equation 3.4 is equivalent to the following simultaneous equations:

$$v_1 = (1\text{-}c).a_{11}.v_1 + (1\text{-}c).a_{12}.v_2 + \dots\dots\dots\dots + (1\text{-}c).a_{1n}.v_n + c.u_1$$

$$v_2 = (1\text{-}c).a_{21}.v_1 + (1\text{-}c).a_{22}.v_2 + \dots\dots\dots\dots + (1\text{-}c).a_{2n}.v_n + c.u_2$$

$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$$

$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$$

$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$$

$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$$

$$v_n = (1\text{-}c).a_{n1}.v_1 + (1\text{-}c).a_{n2}.v_2 + \dots\dots\dots\dots + (1\text{-}c).a_{nn}.v_n + c.u_n$$

Then, after combining all of the entries of the vector **v** on the same side of the equations and removing all v entries, we will have the following (n x n+1) dimensional linear equation:

$$\mathbf{PPV} = \begin{bmatrix} (1-c).a_{11} - 1 + (1-c).a_{12} + \dots\dots\dots + (1-c).a_{1n} = -c.u_1 \\ (1-c).a_{21} + (1-c).a_{i2} - 1 + \dots\dots\dots + (1-c).a_{2n} = -c.u_2 \\ \dots\dots\dots\dots\dots\dots \\ \dots\dots\dots\dots\dots\dots \\ \dots\dots\dots\dots\dots\dots \\ \dots\dots\dots\dots\dots\dots \\ (1-c).a_{n1} + (1-c).a_{n2} + \dots\dots\dots + (1-c).a_{nn} - 1 = -c.u_n \end{bmatrix} \quad \dots (3.6)$$

Therefore, the next step would be to solve this equation and find the appropriate entry values of the vector v.

3.1.3.3 Gauss Jordan Elimination

The Gauss Jordan Elimination method is one of the most efficient and effective algorithms to solve linear equations. It follows a fairly simple process:

1. Input PPV matrix

2. Get the reduced echelon form for R using elementary row operations. First create leading 1's, then make the columns containing leading 1's have only 0's above and below the leading 1's for each column.
3. The matrix in reduced echelon [20] form will either give the solution or demonstrate that there is no solution. In our case, however, there will always be a solution, and it will be unique.

Then, each row of the solution of this linear equation will give the personalized PageRank value (PPV) of the corresponding page, or the URL in our case, which was retrieved and downloaded by the WebCrawler.

Gauss-Jordan Elimination [18]:

```
i_l = 0 (number of leading 1s created)
for j = 1 ... n+1 do:
    if ∃ i > i_l such that a_ij ? 0 then:
        {
          i_l = i_l + 1
          interchange row i and row i_l
          divide row i by a_ij creating a leading 1
          reduce all other entries in the column to 0
        }
```

Complexity:

Since our linear equation is always going to be n x n+1 dimensional, the time complexity can be measured by counting the +, -, * and / operations carried out during the computation.

Since the main computation loop will run n+1 times:

1. The interchange operation takes 2(n+1) operations.
2. Dividing row i by a_{ij} takes n+1-j operations .
3. Reducing other entries takes (n x n+1) operations.

So the time complexity of the algorithm is O(n+1 x n x n+1) or simply $O(n^3)$. We have managed to make this calculation up to size n=800 in our computer for the experimentation.

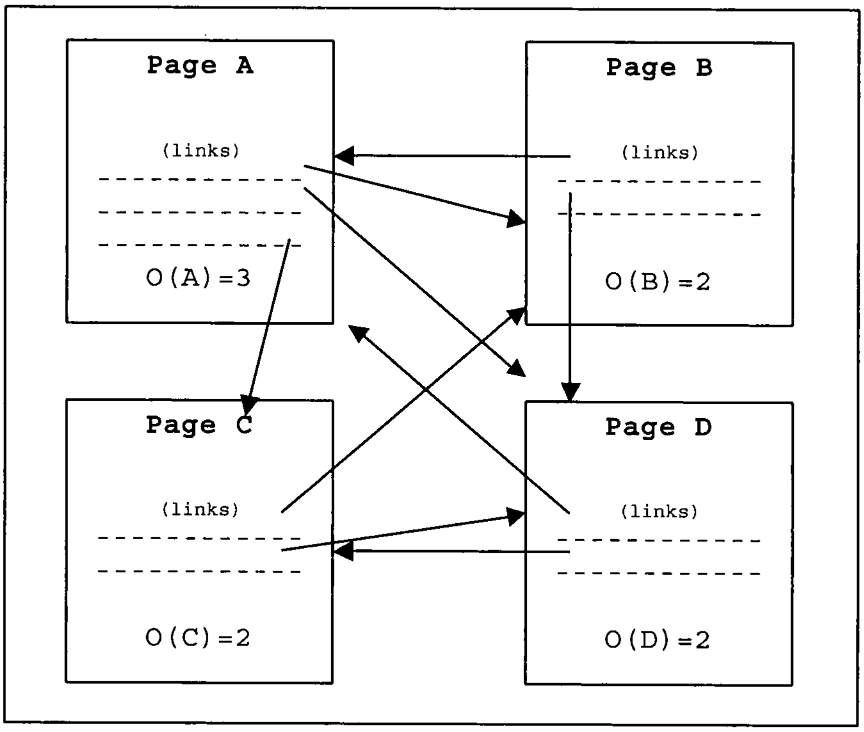


Fig. 3.2: Example of linkage structure

Example:

Let us build a very simple web-like graph shown in Fig. 3.2 and compute the personalized PageRank values of all nodes in this graph.

First, matrix A is constructed by using O(p) values for each $p \in P$ according to the algorithm and let us take c = 0.15.

$$A = \begin{bmatrix} 0 & 1/2 & 0 & 1/2 \\ 1/3 & 0 & 1/2 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 1/2 & 0 \end{bmatrix}$$

Then,

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = 0.85 . \begin{bmatrix} 0 & 1/2 & 0 & 1/2 \\ 1/3 & 0 & 1/2 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} + 0.15 . \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

Let us also assume that pages C and D are in the reference set for that particular user. Then, the **u** vector will be,

$$u = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1/2 \\ 1/2 \end{bmatrix}$$

After combining the values and variables in equation 3.4, we have the following equation:

$$\begin{aligned}
v[1] &= 0.v_1 + 0.425.v_2 + 0.v_3 + 0.425 . v_4 + 0 \\
v[2] &= 0.283.v_1 + 0.v_2 + 0.425.v_3 + 0 . v_4 + 0 \\
v[3] &= 0.283.v_1 + 0.v_2 + 0.v_3 + 0.425 . v_4 + 0.075 \\
v[4] &= 0.283.v_1 + 0.425.v_2 + 0.425 .v_3 + 0 . v_4 + 0.075
\end{aligned}$$

And finally, we solve the following linear equation:

$$PPV = \begin{bmatrix} -1.000 & 0.425 & 0.000 & 0.425 & : & 0.000 \\ 0.283 & -1.000 & 0.425 & 0.000 & : & 0.000 \\ 0.283 & 0.000 & -1.000 & 0.425 & : & -0.075 \\ 0.283 & 0.425 & 0.2425 & -1.000 & : & -0.075 \end{bmatrix}$$

We get the following solutions for vector v:

$$v(1) = 0.215557$$

$$v(2) = 0.178224$$

$$v(3) = 0.275814$$

$$v(4) = 0.328969$$

The solution of the above equation suggests that page D has the highest ranking among all others. In other words, page D is the most important page in the graph, receiving more links from most of the important pages. Also, if we look at the linkage structure, we can clearly expect this because page D has the highest number of links from other pages. Additionally, page D is also one of the two preferred pages of the graph for the user. Hence, it certainly deserves to have the highest ranking.

In the next section, we perform some experiments on some more realistic graphs and user preferences with a larger size of data samples.

3.1.4 The Module Dock (MD)

Module Dock is just the user interface, so we need not analyze the complexity.

3.2 Programming Environment

Module	Operating System	Development Tool
The Web Crawler	Windows XP	Borland Delphi
End User Module	Windows XP	Borland Delphi
Comp. Module	Windows 2000	Borland C++ Builder
Module Dock	Windows XP	Borland Delphi

Table 3.1 Programming Environments

3.3 Specifications

Even though the project can be used for the entire web graph, it has not been tested in this way. Due to time and resource restrictions we have just tested it using Lamar University's web domain. To scale the project for the entire web, advanced calculation techniques need to be deployed.

Since the number of WebPages is changing every moment; it is not possible to determine the size of the matrix and the time to compute the whole web graph. Even though, we have downloaded first five thousand pages of the Lamar University's web domain with our WebCrawler, we will use the only first one thousand pages for our experiments.

CHAPTER 4

EXPERIMENTS

Experiment 1:

In our first experiment, we will run our program in http://www.lamar.edu domain to retrieve the first 100 pages with our WebCrawler starting from the university's home page. We will measure and compare the ranking results of these pages for three different situations:

1) We will first assume that there is no user preference accounted in the ranking; basically it will be the global page rank computation.

2) Then, we will assume that there is a user A, who is interested in the subject of "athletics." Of course, his bookmark pages, history and selected hub pages are mostly going to be about athletics related WebPages in the Lamar University domain.

3) And finally, we will assume another user B who is interested in the subject of "arts and sciences." Again, this

user's preferred pages are mostly going be pages related to his subject in the same domain.

According to the idea we have tried to establish in our project, the algorithm should learn from users' browsing behaviors and habits by retrieving the information from his/her computers for calculating the page ranking. Again, we are not really interested in the searching aspect of the subject in our project. Instead, we will show that our algorithm can alter the searching results by considering users' preferences.

Global Rankings

Table 4.1 shows ten websites from Lamar University's domain with their global PageRank values calculated by using a uniform u vector.

The first column shows the order of the pages retrieved from the domain by our WebCrawler. For example, http://www.lamar.edu, which is the root for the domain, is the first page downloaded and has the order 1 but it is not shown in Table 4.1 because we mainly show the pages related to user preference.

Order	URL	PageRank
.		
.		
25	http://dept.lamar.edu/athletics/Baseball/baseball.htm	0.00206948
26	http://dept.lamar.edu/athletics/MBB/MBB.htm	0.00206948
27	http://dept.lamar.edu/athletics/WBB/WBB.htm	0.00206948
28	http://dept.lamar.edu/athletics/Golf/Golf.htm	0.00206948
29	http://dept.lamar.edu/athletics/Tennis/Tennis.htm	0.00206948
30	http://dept.lamar.edu/athletics/Track/Track.htm	0.00206948
31	http://dept.lamar.edu/athletics/Volleyball/Vball.htm	0.00206948
32	http://dept.lamar.edu/athletics/Summer_Camps.htm	0.00206948
33	http://dept.lamar.edu/athletics/Tubbs.htm	0.00206948
34	http://dept.lamar.edu/athletics/Directory.html	0.00265578
.		
.		
.		
62	http://dept.lamar.edu/artssciences/biology/index.html	0.00219303
64	http://dept.lamar.edu/artssciences/chemistry/index.htm	0.00157174
66	http://dept.lamar.edu/artssciences/english/index.html	0.00273345
67	http://dept.lamar.edu/history/	0.00157174
68	http://dept.lamar.edu/nursing/	0.00157174
69	http://dept.lamar.edu/artssciences/political/index.htm	0.00157174
70	http://dept.lamar.edu/artssciences/psychology/index.html	0.00273345
71	http://dept.lamar.edu/artssciences/crim/index.html	0.00157174
72	http://dept.lamar.edu/languageinstitute/	0.00157174
78	http://dept.lamar.edu/education	0.00157174
.		
.		

Table 4.1

Scenario-1:

User A:

Let us assume that user A has his pages book marked as shown in Table 4.2. We will ignore history and hub pages in this experiment for the sake of simplicity.

Lamar University Baseball	http://dept.lamar.edu/athletics/Baseball/baseball.htm
Lamar University Basketball	http://dept.lamar.edu/athletics/MBB/MBB.htm
Lady Cardinal Basketball	http://dept.lamar.edu/athletics/WBB/WBB.htm
Lamar Golf	http://dept.lamar.edu/athletics/Golf/Golf.htm
Lamar University Tennis	http://dept.lamar.edu/athletics/Tennis/Tennis.htm
Lamar University Track	http://dept.lamar.edu/athletics/Track/Track.htm
Lamar University Volleyball	http://dept.lamar.edu/athletics/Volleyball/Vball.htm
2003 Lamar Athletics Summer Camps	http://dept.lamar.edu/athletics/Summer_Camps.htm
Billy Tubbs-Director of Athletics	http://dept.lamar.edu/athletics/Tubbs.htm
Department Directory	http://dept.lamar.edu/athletics/Directory.html

Table 4.2

The bookmark structure will provide the following **u** vector for the computation according to equation 3.5,

u = [0, 0, . . ., 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0, 0, . . ., 0, 0]

After running our program with this new preference vector, the new personalized PageRank values for the same pages are as shown in the Table 4.3:

Order	URL	PPV
.		
.		
25	http://dept.lamar.edu/athletics/Baseball/baseball.htm	0.0174167
26	http://dept.lamar.edu/athletics/MBB/MBB.htm	0.0174167
27	http://dept.lamar.edu/athletics/WBB/WBB.htm	0.0174167
28	http://dept.lamar.edu/athletics/Golf/Golf.htm	0.0174167
29	http://dept.lamar.edu/athletics/Tennis/Tennis.htm	0.0174167
30	http://dept.lamar.edu/athletics/Track/Track.htm	0.0174167
31	http://dept.lamar.edu/athletics/Vo lleyball/Vball.htm	0.0174167
32	http://dept.lamar.edu/athletics/Summer_Camps.htm	0.0174167
33	http://dept.lamar.edu/athletics/Tubbs.htm	0.0174167
34	http://dept.lamar.edu/athletics/Directory.html	0.0181014
.		
.		
.		
62	http://dept.lamar.edu/artssciences/biology/index.html	0.000045958
64	http://dept.lamar.edu/artssciences/chemistry/index.htm	0.000000329
66	http://dept.lamar.edu/artssciences/english/index.html	0.000000572
67	http://dept.lamar.edu/history/	0.000000329
68	http://dept.lamar.edu/nursing/	0.000000329
69	http://dept.lamar.edu/artssciences/political/index.htm	0.000000329
70	http://dept.lamar.edu/artssciences/psychology/index.html	0.000000572
71	http://dept.lamar.edu/artssciences/crim/index.html	0.000000329
72	http://dept.lamar.edu/languageinstitute/	0.000000329
78	http://dept.lamar.edu/education	0.000000329
.		
.		

Table 4.3

User B:

Table 4.4 shows the bookmark pages of user B:

Welcome to Lamar University	http://dept.lamar.edu/artssciences/biology/index.html
Lamar University's Department of Physics and Chemistry	http://dept.lamar.edu/artssciences/chemistry/index.htm
English and Foreign Languages	http://dept.lamar.edu/artssciences/english/index.html
Welcome to Department of	http://dept.lamar.edu/history/
Welcome to Lamar University	http://dept.lamar.edu/nursing/
Political Science	http://dept.lamar.edu/artssciences/political/index.htm
Department of Psychology	http://dept.lamar.edu/artssciences/psychology/index.html
Criminal Justice	http://dept.lamar.edu/artssciences/crim/index.html
Lamar University Language	http://dept.lamar.edu/languageinstitute/
College of Education and Human Development	http://dept.lamar.edu/education

Table 4.4

This bookmark structure will provide a new preference vector **u** for this computation. After running our program with this new vector, the new Personalized Pagerank values for the same pages are as shown in Table 4.5:

Order	URL	PPV
.		
.		
25	http://dept.lamar.edu/athletics/Baseball/baseball.htm	0. 000512866
26	http://dept.lamar.edu/athletics/MBB/MBB.htm	0.000512866
27	http://dept.lamar.edu/athletics/WBB/WBB.htm	0.000512866
28	http://dept.lamar.edu/athletics/Golf/Golf.htm	0.000512866
29	http://dept.lamar.edu/athletics/Tennis/Tennis.htm	0.000512866
30	http://dept.lamar.edu/athletics/Track/Track.htm	0.000512866
31	http://dept.lamar.edu/athletics/Volleyball/Vball.htm	0.000512866
32	http://dept.lamar.edu/athletics/Summer_Camps.htm	0.000512866
33	http://dept.lamar.edu/athletics/Tubbs.htm	0.000512866
34	http://dept.lamar.edu/athletics/Directory.html	0.000658163
.		
.		
.		
.		
62	http://dept.lamar.edu/artssciences/biology/index.html	0.0210928
64	http://dept.lamar.edu/artssciences/chemistry/index.htm	0.0151171
66	http://dept.lamar.edu/artssciences/english/index.html	0.0262907
67	http://dept.lamar.edu/history/	0.0151171
68	http://dept.lamar.edu/nursing/	0.0151171
69	http://dept.lamar.edu/artssciences/political/index.htm	0.0151171
70	http://dept.lamar.edu/artssciences/psychology/index.html	0.0151171
71	http://dept.lamar.edu/artssciences/crim/index.html	0.0151171
72	http://dept.lamar.edu/languageinstitute/	0.0151171
78	http://dept.lamar.edu/education	0.0151171
.		
.		

Table 4.5

Results:

It is possible to see the following results by studying this experiment in the Lamar university domain:

1. Without taking into account user preferences, the linkage structure of the graph is the only factor that affects the PageRank values of the nodes.
2. It is clear that by taking into account the preference pages of user A, the algorithm increases the ranking values of the pages that are related to "athletics." We can also see that the rankings of the pages related to "arts & sciences" are lower than the global rankings in the Table 4.1 because user A has no preference for this subject.
3. In Table 4.5, since user B has no interest in the area of "athletics," web pages related to this subject received much lower rankings than the previous figure. On the other hand, the pages about "arts & sciences" received significantly higher rankings than the previous one, even higher than the global ranking values.

Experiment 2:

In this experiment, we ran our program in http://www.stanford.edu domain to retrieve the first 500 pages with our WebCrawler starting from the university's home page. Different from the first experiment, we searched for some actual query in the university's WebPages. Then, we ranked the results before and after user's preferences were taken into account.

Step 1:

We searched for the query "student" in the http://www.stanford.edu domain, and Table 4.6 shows some of the links that were returned by the result page.

http://www.stanford.edu/home/students/
http://www.stanford.edu/home/welcome/students/
http://vaden.stanford.edu/
http://www.stanford.edu/dept/vpsa/
http://www.stanford.edu/group/psa/
http://www.stanford.edu/group/sutsa/
http://www.stanford.edu/group/DRC/
http://www.stanford.edu/group/wingchun/

Table 4.6 Part of the search results

Step 2:

We ran our WebCrawler module (WC) to fetch the first 500 pages in the domain starting from the Stanford homepage. That process generated the linkage structure matrix of the domain, which was used as an input for the computational module (CM). Afterwards, we ran the CM without any user preferences to compute the global page rank values for the domain. Table 4.7 shows the global PageRank values of the selected links in order. Highlighted items in the Table 4.7 are the links, which will be included in the set of preference pages in the next step.

Link	Global Page Rank
http://www.stanford.edu/home/students/	0.000505911
http://www.stanford.edu/group/DRC/	0.000473848
http://vaden.stanford.edu/	0.000358133
http://www.stanford.edu/dept/vpsa/	**0.000320641**
http://www.stanford.edu/home/welcome/students/	0.000313438
http://www.stanford.edu/group/wingchun/	**0.000312469**
http://www.stanford.edu/group/psa/	0.000311941
http://www.stanford.edu/group/sutsa/	**0.000310868**

Table 4.7 Ranks sorted by global page rank

Step 3:

After computing the global PageRank values of these links, we retrieved the user-preferred pages, shown in Table 4.8, as identified through the user's history, bookmark and hub pages, by using the end-user module (EUM) to build the u vector. Then, we ran the CM again with these user preferences to compute the personalized PageRank values for the domain. Table 4.9 shows the global PageRank values and the personalized PageRank values (PPV) of the selected links sorted according to the PPV.

Link	**Source**
http://www.stanford.edu/	Favorites
http://www.stanford.edu/group/wingchun/	**Favorites**
http://givingtostanford.stanford.edu/	Favorites
http://summer.stanford.edu/	Favorites
http://transportation.stanford.edu/	History
http://www.stanford.edu/dept/vpsa/	**History**
http://www.stanford.edu/group/SURE/	History
http://www.stanford.edu/faculty/positions.html	History
http://www.med.stanford.edu/alumni/	Hubs
http://www.stanford.edu/group/sutsa/	**Hubs**

Table 4.8 User Preferences

Link	Global Page Rank	Personalized Page Rank
http://www.stanford.edu/dept/vpsa/	**0.000320641**	**0.0150151**
http://www.stanford.edu/group/wingchun/	**0.000312469**	**0.0150091**
http://www.stanford.edu/group/sutsa/	**0.000310868**	**0.0150079**
http://www.stanford.edu/home/students/	0.000505911	0.000375289
http://www.stanford.edu/group/DRC/	0.000473848	0.000032474
http://vaden.stanford.edu/	0.000358133	0.000023934
http://www.stanford.edu/home/welcome/students/	0.000313438	0.000009968
http://www.stanford.edu/group/psa/	0.000311941	0.000008677

Table 4.9 Ranks sorted by personalized page rank

Results:

1. Without user preferences taken into account, the linkage structure of the graph is the only factor that affects the PageRank values of the nodes.
2. It is clear that by taking into account the preference pages of the user, the algorithm increases the ranking values of the pages in the user preferences, as expected. We can also see that the pages that are not in the user preferences received significantly lower rank values than their global rankings.

Experiment 3:

In this experiment, we ran our program again in http://www.stanford.edu domain to retrieve the first 500 pages with our WebCrawler starting from the university's home page exactly the same way we did in the second experiment. Our primary purpose in this final experiment was to observe the results of the same domain for same size of a graph with a different "teleportation" constant, c, than the second experiment. Instead of using 0.15 for the teleportation constant, we used 0.25 for the constant value. Similar to the second experiment, we searched for some actual query in the university's WebPages. Then, we ranked the results before and after user preferences were taken into account by using the same steps we did for the second experiment.

Step 1:

Table 4.10 shows the links that were returned by the result page similar to the second experiment:

http://www.stanford.edu/home/students/
http://www.stanford.edu/home/welcome/students/
http://vaden.stanford.edu/
http://www.stanford.edu/dept/vpsa/
http://www.stanford.edu/group/psa/
http://www.stanford.edu/group/sutsa/
http://www.stanford.edu/group/DRC/
http://www.stanford.edu/group/wingchun/

Table 4.10 Part of the search results

Step 2:

Table 4.11 shows the global PageRank values of the selected links in order after running the CM without any user preferences. Highlighted items in the Table 4.11 are the links, which will be included in the set of preference pages in the next step.

Link	Global Page Rank
http://www.stanford.edu/home/students/	0.00746461
http://www.stanford.edu/group/DRC/	0.00695564
http://www.stanford.edu/home/welcome/students/	0.00674111
http://vaden.stanford.edu/	0.00574174
http://www.stanford.edu/dept/vpsa/	**0.00537524**
http://www.stanford.edu/group/wingchun/	**0.0051458**
http://www.stanford.edu/group/sutsa/	**0.00512705**
http://www.stanford.edu/group/psa/	0.0051152

Table 4.11 Ranks sorted by global page rank

Step 3:

After computing the global PageRank values of these links, we retrieved the user-preferred pages, shown in Table 4.12, by using the end-user module (EUM) to build the u vector. Then, we ran the CM again with these user preferences to compute the personalized PageRank values for the domain. Table 4.13 shows the global PageRank values and the personalized PageRank values (PPV) of the selected links sorted according to the PPV.

Link	Source
http://www.stanford.edu/	Favorites
http://www.stanford.edu/group/wingchun/	**Favorites**
http://givingtostanford.stanford.edu/	Favorites
http://summer.stanford.edu/	Favorites
http://transportation.stanford.edu/	History
http://www.stanford.edu/dept/vpsa/	**History**
http://www.stanford.edu/group/SURE/	History
http://www.stanford.edu/faculty/positions.html	History
http://www.med.stanford.edu/alumni/	Hubs
http://www.stanford.edu/group/sutsa/	**Hubs**

Table 4.12 User Preferences

Link	Global Page Rank	Personalized Page Rank
http://www.stanford.edu/dept/vpsa/	**0.00537524**	**0.0220298**
http://www.stanford.edu/group/wingchun/	**0.0051458**	**0.0220115**
http://www.stanford.edu/group/sutsa/	**0.00512705**	**0.0210103**
http://www.stanford.edu/group/DRC/	0.00695564	0.00209855
http://vaden.stanford.edu/	0.00574174	0.00173641
http://www.stanford.edu/home/students/	0.00746461	0.000599034
http://www.stanford.edu/home/welcome/students/	0.00674111	0.000032536
http://www.stanford.edu/group/psa/	0.0051152	0.0000090805

Table 4.13 Ranks sorted by personalized page rank

Results:

Other than the results of the Experiment 2, we have the followings:

1. Changing the Teleportation Probability, c, did not affect the algorithm. The algorithm still worked in the expected way with some minor changes in the ranking values.

2. Since the linkage structure changed by time, the order of the rankings has been affected slightly in this experiment.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this project, we have accomplished the computation of the personalized PageRank according to the user preferences as follows:

1. For the simplicity of computation, we gathered the list of URLs (Universal Resource Locator) with the linkage structures of the specific web domain (in this experiment, the Lamar University domain), by using the WebCrawler module we implemented.
2. We obtained user preferences, through bookmarks, histories, etc, by using the End-User Module to generate the preference vector.
3. We computed the Personalized PageRank Vector (PPV) by using the Computation Module (CM).

From the experiments, it is very clear that the user preferences affect the rankings of the pages dramatically, as was proposed by the main idea of personalized page ranking.

In our experiment, we gathered user preferences from static sources, such as the user's bookmarks and selected pages, and from dynamic sources like the user's browsing history.

5.2 Future Work

It is clear that the personalized PageRank computation can go beyond the limited domain we used in our experiments. We limited the size of the sample web graph for the sake of simplicity and in considering the availability of memory resources. Future work might involve scaling the idea we have presented to the entire web graph.

Though the use of linkage structure of the web has been proven good enough for searching the web, it does not represent the traffic flow through a certain link. A particular user's traffic on a link represents the importance of the pages connected by that link. The traffic pattern shows how frequently a user visits these websites. As future work, one can implement a different history folder in a user's personal computer in which he will store not only a list of pages visited but also the frequency of the visits to those web pages and how much time the user spent on those pages.

Therefore, sources of user preference can be extended to the other dynamic sources as follows:

1. The time spent on a particular page by the user on visiting it.
2. The frequency of the user's visits to the page, either randomly or frequently.

The main objective of this project was to rank the web pages, which was the output of the search engine. In the future, this ranking can be integrated with the search engine. Initially, a user issues the query, and receives results. Future work could allow the user to modify his query to indicate his preferences.

Although some previous work [12] discussed the concept of query modification, it did not include the computation of page rank and re-ordering. Using this modified query, it would be possible to compute the page rank and do the computation of PPV again to re-organize the results more related with the user-preferred pages.

BIBLIOGRAPHY

[1] Glen Jeh, Jennifer Widom. Scaling Personalized Web Search, Technical Report, Computer Science Department, Stanford University, 2002.

[2] Larry Page, Sergey Brin, R. Motwani, T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report, Stanford University Database Group 1998.

[3] Chris Ding, Hongyuan Zha, Xiaofeng He, Parry Husbands, Horst Simon. Link Analysis: Hubs and Authorities on the World Wide Web. LBNL Technical Report (2001).

[4] Cyrus Shahabi, Yi-Shin Chen. Web Information Personalization: Challenges and Approaches. The 3rd International Workshop on Databases in Networked Information Systems (DNIS 2003).

[5] C.R. Dyer. Lecture Notes, University of Wisconsin - Madison.

[6] http://www.cs.utsa.edu/~kwek/cpts483/mit/lecture1.ps

[7] Nils J. Nilson. Introduction to Machine Learning.

[8] Uthayakumar Gevaran. Topics in Machine Learning Web Project, Department of Computer Science, Brandies University.

[9] Sergey Brin and Lawrence Page. The Anatomy of Large-Scale Hypertextual Web Search Engine, Computer Science Department, Stanford University.

[10] Jeffrey Dean and Monika R. Henzinger. Finding Related Pages in the World Wide Web.

[11] http://www.aaai.org/AITopics/html/machine.html

[12] E. J. Glover, G. W. Flake, S. Lawrence, W. P. Birmingham, A. Kruger, C. L. Giles, D. M. Pennock. Improving Category Specific Web Search by Learning Query Modifications, SAINT 2001, SanDiego, CA.

[13] J. Wu, K. Aberer. Semantic Web Graph Implied by User Preferred Activities, School of Computer and Communication Science, Lausanne, Switzerland.

[14] The Web Robots Pages. http://info.webcrawler.com/mak/projects/robots/robots.htm

[15] David Eichmann. The RBSE Spider - Balancing Effective Search Against Web Load. In *Proceed-ings of the First International World Wide Web Conference*, pages 113-120, 1994.

[16] Oliver A. McBryan. GENVL and WWWW: Tools for Taming the Web. In Proceedings of the First International World Wide Web Conference, pages 79-90, 1994.

[17] Brian Pinkerton. Finding What People Want: Experiences with the WebCrawler. In Proceedings of the Second International World Wide Web Conference, 1994.

[18] http://simon.incutio.com/archive/2002/10/01/gaussJordanElimination

[19] Cyrus Shahabi and Yi-Shin Chen, Web Information Personalization: Challenges and Approaches.

[20] Larry Joel Goldstein. Algebra and Trigonometry and Their Applications. Second edition, Wm.C.Brown Publishers

Appendix A

Partial source code for different modules

1. The Module Dock (MD)

```
procedure TForm1.Label1MouseEnter(Sender: TObject);
begin
     WCEnter();

end;

procedure TForm1.Label1MouseLeave(Sender: TObject);
begin
     WCLeave();
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
     Close();
end;

procedure TForm1.EUMEnter;
begin
     Shape3.Brush.Color := clBackground;
     Shape4.Brush.Color := Shape3.Brush.Color;
     Shape7.Brush.Color := Shape3.Brush.Color;
     Label3.color:=Shape3.Brush.Color;
     Label2.Color:=Shape3.Brush.Color;
     Label8.Color:=Shape3.Brush.Color;
end;

procedure TForm1.EUMLeave;
begin
     Shape3.Brush.Color := clWhite;
     Shape4.Brush.Color := Shape3.Brush.Color;
     Shape7.Brush.Color := Shape3.Brush.Color;
     Label3.color:=Shape3.Brush.Color;
     Label2.Color:=Shape3.Brush.Color;
     Label8.Color:=Shape3.Brush.Color;
end;

procedure TForm1.Shape7MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
     if EUMRunning=false then
     begin
          if (((x>10) and (x<200)) and ((y>10) and (y<180))) then
             EUMEnter
          else
              EumLeave;
```

```
     end;
end;

procedure TForm1.RunExternalProgram(path: string;programName:string);
var
  SEInfo: TShellExecuteInfo;
  ExitCode: DWORD;
  ExecuteFile:string;
begin
  ExecuteFile:=path;

  FillChar(SEInfo, SizeOf(SEInfo), 0);
  SEInfo.cbSize := SizeOf(TShellExecuteInfo);
  with SEInfo do begin
    fMask := SEE_MASK_NOCLOSEPROCESS;
    Wnd := Application.Handle;
    lpFile := PChar(ExecuteFile);
    nShow := SW_SHOWNORMAL;
  end;
  if ShellExecuteEx(@SEInfo) then begin
    repeat
      Application.ProcessMessages;
      GetExitCodeProcess(SEInfo.hProcess, ExitCode);
    until (ExitCode <> STILL_ACTIVE) or
         Application.Terminated;
    if programName='Web Crawler' then
    begin
         WCRunning:=false;
         WCLeave();
    end
    else
    begin
        EUMRunning:=false;
        EUMLeave();
    end;
  end
  else ShowMessage('Error starting' + programName);

end;

procedure TForm1.Label1Click(Sender: TObject);
begin
     WCRunning:=true;
     RunExternalProgram('wc.exe','Web Crawler');
end;

procedure TForm1.Shape7MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
     EUMRunning:=true;
     RunExternalProgram('c:\Windows\system32\calc.exe','End User Module');
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
     WCRunning:=false;
     EUMRunning:=false;
```

```
end;

procedure TForm1.WCEnter;
begin
     if WCRunning=false then
     begin
          Shape1.Brush.Color := clBackground;
          Label1.Color:=Shape1.Brush.Color;
     end;


end;

procedure TForm1.WCLeave;
begin
     if WCRunning=false then
     begin
          Shape1.Brush.Color := clWhite;
          Label1.Color:=Shape1.Brush.Color;
     end;
end;

procedure TForm1.Label9Click(Sender: TObject);
begin
     RunExternalProgram('Results.exe','Results');
end;

end.
```

2. The Results

```
function SortByColumn(Item1, Item2: TListItem; Data: integer):    integer;
stdcall;
begin
     if Data = 0 then
        Result := AnsiCompareText(Item1.Caption, Item2.Caption)
     else
         Result := AnsiCompareText(Item1.SubItems[Data-
1],Item2.SubItems[Data-1]);
     if not Ascending then Result := -Result;
end;

procedure TForm1.LVColumnClick(Sender: TObject; Column: TListColumn);
begin
     if Column.Index = LastSortedColumn then
        Ascending := not Ascending
     else
         LastSortedColumn := Column.Index;

     TListView(Sender).CustomSort(@SortByColumn,Column.Index);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
     LastSortedColumn := -1;
```

```
     Ascending := false;
end;

procedure TForm1.ReadAndSort;
var
   tempL:TListItem;
   datp:psingle;
   dat:single;
   links:TStringList;
   ranks:TStringList;
   i:integer;
   k:integer;
begin
     links:=TStringList.Create;
     links.LoadFromFile('links.txt');

     ranks:=TStringList.Create;
     ranks.LoadFromFile('ranks.txt');

     i:=links.Count;
     for k:=0 to i-1 do
     begin

          tempL:=LV.Items.Add;
          tempL.Caption:=links[k];
          tempL.SubItems.Add(ranks[k]);
     end;
     LV.CustomSort(@SortByColumn,1);
end;

procedure TForm1.FormShow(Sender: TObject);
begin
ReadandSort;

end;

end.
```

3. The Computational Module (CM)

```
/////////////////////////////////////////////////////////////////

            void Matrix::MultTimesEach ( double multiplier )
            {
                  for( int r = 0; r < rows; r++ )
                        for( int c = 0; c < cols; c++ )
                              item[r][c] *= multiplier;
            }
/////////////////////////////////////////////////////////////////

            void  Matrix::DiagonalModify( double value )
            {
                  for( int r = 0; r < rows; r++ )
                        item[r][r] = item[r][r] - value;
            }
```

```
/////////////////////////////////////////////////////////////////

void Matrix::GaussJEliminate (ostream& os, ostream& os2)
{

float ci[MaxCols], temp_pi_value, dummy_pi_value, big_pi_value,
argument_value, sm_value, ai[MaxRows][MaxCols] ;

float x[MaxCols],a[MaxCols][MaxCols],c[MaxRows];
int i,n,j,ell,i1,elx,pij,i2,sul,ix,sui,suj,jx,d;

n=rows;
for (i=1; i<=rows; i++)
{
      for (j=1; j<=(cols-1); j++)
      {
            a[i][j]=item[i-1][j-1];
            ai[i][j]=a[i][j];
      }
      c[i]=item[i-1][j-1];
     j++;
     ci[i]=c[i];
}

for (int iiii=1;iiii<n;iiii++)
{
        for(int jjj=1;jjj<cols;jjj++)
        {
         //       cout << "a["<<iiii<<"]["<<jjj<<"]="<<a[iiii][jjj]<<"
";
        }
       // cout << "\n";
}



/* FORWARD ELIMINATION PART*/
ell_value=n-1;
for (i1=1;i1<=ell_value;i1++)
{
      elx=i1+1;
      pij=i1;
      big_pi_value=fabs(a[i1][i1]);
      for (i=elx;i<=n;i++)
      {
            argument_value=fabs(a[i][i1]);
            if (argument_value>big_pi_value)
            {
                  big_pi_value=argument_value;
                  pij=i;
            }
      }
      if (pij != i1)
      {
            for (j=i1;j<=n;j++)
            {
```

```
            dummy_pi_value=a[pij][j];
            a[pij][j]=a[i1][j];
            a[i1][j]=dummy_pi_value;
            }
      temp_pi_value=c[pij];
      c[pij]=c[i1];
      c[i1]=temp_pi_value;

      }
      for (i2=elx;i2<=n;i2++)
      {
            for (j=elx;j<=n;j++)
            {
            a[i2][j]=a[i2][j]-a[i2][i1]/a[i1][i1]*a[i1][j];
            }
      c[i2]=c[i2]-a[i2][i1]/a[i1][i1]*c[i1];
      }
} /* NEXT i1 */
/* BACK SUBSTITUTE */
sul=n-1;

x[n]=c[n]/a[n][n];
for (ix=1;ix<=sul;ix++)
{
      sm_value=0;
      sui=n-ix;
      suj=sui+1;
      for (jx=suj;jx<=n;jx++)
            {
            sm_value=sm_value+a[sui][jx]*x[jx];
            }
      x[sui]=(c[sui]-sm_value)/a[sui][sui];
}


os << "\n\nSolutions to the equations\n";

float total=0.00;

for (i=1;i<=n;i++)
{
os <<"x("<<i<<") ="<<x[i]<<"\n";
cout <<"x("<<i<<") ="<<x[i]<<"\n";
os2 << i << " " << x[i] << "\n";
total+=x[i];
}

os << "total=" << total << endl;
cout << "total=" << total << endl;
}
```

4. The Web Crawler

```
unit Unit1;
```

```
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, Sockets,ComCtrls, ExtCtrls, Grids,ShellApi,
Menus, myClasses, ToolWin;


type TWCThread = class(TThread)
  private
    { Private declarations }
    fReadMemo:TStringGrid;              // links will be read from the
string grid
    fTI:integer;                        // index of the thread
    myCI:integer;              //index of the line that willbe read
    maxNumberofLinks:integer;// the number of links to be explored
    myLink:TLink;        // The object for the linkage structure
    BH:String;                          // BaseHost
    Page:String;                    // downloaded html(htm) file
    Cl: TTcpClient;                     // TCPClient object
    AccFiles:TStringList;               // Acceptable file list
    function isAcceptedType():boolean;
// Checks if the link is an accepted type
    procedure DownloadPage();
// Downloads the page specified by myLink.fCurrentURL
    procedure ProcessPage();
// Explore the downladed page
  public
        constructor Create(rMemo:TStringGrid;
        TI:integer;mnoflinks:integer;BHost:String);
// constructor
        procedure Start();
// the main procedure
        procedure addAccFileType(FileExtension:string);
// adds the Fileextension to unacceptable file list

  protected
    procedure Execute; override;
// thread's execute method
```

```
end;

type
  TForm1 = class(TForm)
    StatusBar1: TStatusBar;
    Cl: TTcpClient;
    Splitter1: TSplitter;
    MainMenu1: TMainMenu;
    Edit4: TMenuItem;
    Options1: TMenuItem;
    Panel2: TPanel;
    ToolBar1: TToolBar;
    ToolButton1: TToolButton;
    N1: TMenuItem;
    Start1: TMenuItem;
    Panel3: TPanel;
    SG: TStringGrid;
    Panel4: TPanel;
    Memo1: TMemo;
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    PB: TProgressBar;
    procedure Options1Click(Sender: TObject);
    procedure Start1Click(Sender: TObject);
    procedure ToolButton1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
    fStatus:String;          // the status
    fDomain:String;          // domain info like www.lamar.edu
    nol:integer;             // # of links to process
    noft:integer;            // number of threads to be created
    aFileName:TFileName;     // name of the matrix's output file
    oFileName:TFileName;     // name of the link's output file
    nodp:integer;            // number of decimal points
    BaseHost:String;         // Base host name like lamar.edu
```

```
  Public
    { Public declarations }
   procedure Start();                    // Starts explore process
   procedure don(sender:Tobject); // thread termination process
   procedure setStatus(value:String);  // Sets the status
   procedure setDomain(value:String);  // sets the domain
   procedure readSettings();// Reads and apply the user selection
   procedure formatStringGrid();        // Formats the grid
  published
      property Status:String read fStatus write setStatus;
            // status property
      property domain:String read fdomain write setDomain;
            // domain info
  end;

var
   // The Global variables
   CurrentIndex:TRTLCriticalSection;
// critical section for reading and writing link index
   ReadSection:TRTLCriticalSection;
// critical section for reading and updating active thread count
   ThreadStatus:TRTLCriticalSection;
// critical section for displaying thread status information
   LinkWrite:TRTLCriticalSection;
// critical section for displaying thread's process
   LinkCount:TRTLCriticalSection;
// critical section for updating linkage structure
   Form1: TForm1;
// the main form
   CI:integer;   // the index of the next available link( the link with
smallest index which is not explored yet )
   fNumberOFLine:Integer;        // number of line links found
   TC:integer;                                  // Thread count
   DT:integer;      // # of threads which are currently downloading
   fmyPages:TmyPages;             // The object for the matrix
   // constant HTTP headings
  const
  crlf =   '#13#10';
```

```
  ending ='HTTP/1.0';
  init = 'Connection: Close';
  init2 = 'User-Agent: Mozilla/4.0 (compatible; MSIE 5.5)';
  init3 = 'Accept: text/html, text/plain';
implementation
uses DomainInfo, Options, Unit2;
{$R *.dfm}

function TWCThread.isAcceptedType():boolean;
var
   filExtension:PChar;
begin
     // Checks whether found link has an acceptable type
     if ( myLink.LinkFile<>'' ) then
     begin
          filExtension:=strRScan(pchar(myLink.LinkFile),'.');
          if AccFiles.IndexOf(filExtension)>0 then
             result:=true
          else
              result:=false;
     end
     else
         result:=true;
end;

procedure TWCThread.DownloadPage();
var
   ReqText:String;
   so:String;
   HTTPOK:boolean;
   kb:integer;
begin
     // Downloads the link myLink.fCurrentURL;
      EnterCriticalSection(ReadSection);
      // increments # of threads which are in download process
          inc(DT);
      LeaveCriticalSection(ReadSection);
      Cl.RemoteHost:=myLink.CurrentHostName;;
```

```
        Cl.Connect;
    if Cl.active then
    begin
            // HTTP request
            ReqText :='GET ' +myLink.CurrentDir + myLink.CurrentFile + '
HTTP/1.1';
            Cl.Sendln(ReqText);
            ReqText:='Host: '+BH;
            Cl.Sendln(ReqText);
            ReqText:=init;
            Cl.Sendln(ReqText);
            ReqText:=init2;
            Cl.Sendln(ReqText);
            ReqText:=init3 +#13+#10;
            Cl.Sendln(ReqText);
            Page:='';
            HTTPOK:=true;
            if (Cl.WaitForData(1000000)) then
               so:=Cl.Receiveln();
            if (pos('200 OK',so)=0) then
            begin
                 // something is wrong
                 HTTPOK:=false;
                 EnterCriticalSection(LinkWrite);
                     fReadMemo.Cells[3,myCI]:=Format('T:%d Can not
download.',[fTI]) ;
                 LeaveCriticalSection(LinkWrite);
            end
            else
            begin
                 while(Cl.WaitForData(1000000)) do
                 begin
 // http is ok keep reading until there is no more data waiting for us to
read
                      so:=Cl.Receiveln();
                      Page := Page + ' ' +so;
                 end;
            end;
```

```
            Cl.Disconnect; // Disconnect;
              if (HTTPOK=true) then
            begin
                 // if page is downloaded succesfully, explore it to find
the out-links
                 EnterCriticalSection(LinkWrite);
                     fReadMemo.Cells[3,myCI]:=Format('T:%d
downloaded.Searching for links %d',[fTI,strlen(pchar(page))]) ;
                 LeaveCriticalSection(LinkWrite);
                 kb:=strlen(pchar(page));
                 ProcessPage();
                 EnterCriticalSection(LinkWrite);
                     fReadMemo.Cells[3,myCI]:=Format('T:%d Explored...%d
bytes',[fTI,kb]) ;
                 LeaveCriticalSection(LinkWrite);
            end;
      EnterCriticalSection(ReadSection);
          // decrements # of threads which are in download process
          dec(DT);
      LeaveCriticalSection(ReadSection);
  end
  else
  begin
       EnterCriticalSection(LinkWrite);
           fReadMemo.Cells[3,myCI]:=Format('T:%d Can not connect.',[fTI])
;
       LeaveCriticalSection(LinkWrite);
  end;
end;

constructor TWCThread.Create
(rMemo:TStringGrid;TI:integer;mnoflinks:integer;BHost:String);
begin
     // constructor for the Threads
     // initilaze data sturctures
     // sets the acceptable file types
     // sets the current link index( myCI ) to 0
     // Creates the TCPClients
```

```
    maxNumberofLinks:=mnoflinks;
    fReadMemo:=rMemo;
    BH:=BHost;
    myLink:=TLink.create;
    AccFiles:=TStringList.Create;
    addAccFileType('.htm');
    addAccFileType('.html');
    fTI:=TI;
    myCI:=0;
    Cl:=TTcpClient.Create(form1);
    Cl.RemotePort:=IntToStr(80);
    inherited Create(False);
end;

procedure TWCThread.Execute;
begin
    // thread execution gets start
    InitializeCriticalSection(CurrentIndex);
    InitializeCriticalSection(ReadSection);
    InitializeCriticalSection(ThreadStatus);
    InitializeCriticalSection(LinkWrite);
    InitializeCriticalSection(LinkCount);

    EnterCriticalSection(CurrentIndex);
    // [CI] increments the current available link index = last visited
link index
    // [myCI] sets its current link index to CI
       inc(CI);
       myCI:=CI;
    LeaveCriticalSection(CurrentIndex);

    Start();    // starts the threads

    // thread is done
    EnterCriticalSection(ThreadStatus);
       Form1.Memo1.Lines.Add(Format('Thread %d is done',[fti]));
    LeaveCriticalSection(ThreadStatus);
end;
```

```
// REads, downloads and explores the link from the string list
procedure TWCThread.Start();
var
   TCount:integer;            // number of threads which are in download
process
   nofl:integer;              // number of links found
   temp:string;
   isDownladed:integer;
begin
     EnterCriticalSection(ReadSection);
       nofl:=fNumberOFLine;     // number of links found
       TCount:=DT;
     LeaveCriticalSection(ReadSection);

     while ( (myCI<=maxNumberofLinks) and (( myCI < nofl ) or (TCount>0)))
do
     begin
          // while there are more links to be explored or there are more
threads which are in download process. This threads may found new links to
be explored.
              if (myCI<nofl) then
              begin
                   EnterCriticalSection(LinkWrite);
                       // reads the link from the string grid
                       // using its current link index (myCI)

                       temp:=pchar(fReadMemo.Cells[2,myCI]);
                       myLink.CurrentURL:=temp;
                       fReadMemo.Cells[3,myCI]:=Format('T:%d
downloading...',[fTI]) ;
                   LeaveCriticalSection(LinkWrite);

                   EnterCriticalSection(LinkCount);
      // add the link to the links list, if it is not already there

isDownladed:=fMypages.isDownloaded(mylink.CurrentURL);
                    LeaveCriticalSection(LinkCount);
```

```
                    if (isDownladed<0) then
                     // if the link has not been downloaded yet
                     // download it
                         DownloadPage()
                    else
                    begin
                        EnterCriticalSection(LinkWrite);
                            fReadMemo.Cells[3,myCI]:=Format('T:%d already
explored...',[fTI]) ;
                        LeaveCriticalSection(LinkWrite);
                    end;

                    EnterCriticalSection(CurrentIndex);
                  // increments the global CI ( available link index)
                  // reserves that for it self myCI:=CI;
                       inc(CI);
                       myCI:=CI;
                       form1.PB.StepIt;
                   // process bar on the form
                    LeaveCriticalSection(CurrentIndex);
               end
               else
               begin
                    EnterCriticalSection(LinkWrite);
                        fReadMemo.Cells[3,myCI]:=Format('T:%d
zzzzzzzz.....',[fTI]) ;
                    LeaveCriticalSection(LinkWrite);
              // there is no more links to explore
              // wait for the threads which are in download process
              // they may find new links
              // sleep for 1000 msec and check again
                    Sleep(1000);
               end;
               EnterCriticalSection(ReadSection);
                   nofl:=fNumberOFLine;
                   // update the number of links found
                   TCount:=DT;
                   // update # of threads in download process
```

```
            LeaveCriticalSection(ReadSection);
      end;
end;

procedure TWCThread.addAccFileType(FileExtension: string);
begin
      // add the extension to the string list
      if (AccFiles.IndexOf(FileExtension)<0) then
         AccFiles.Add(FileExtension);
end;

procedure TWCThread.ProcessPage();
var
   remaining,remaining2 :Pchar;
   nofl:integer;
   cri:integer;
   isExist:boolean;
   isLinkOk:Boolean;
begin
      // Page is the file just downloaded

      remaining:=PChar(Page);
      remaining:=AnsiStrpos(remaining,'href="');
      while (remaining<> nil)  do
      begin
            remaining:=strScan(remaining,'"')+1;
// start of the link
            remaining2:=strScan(remaining,'"');
// end of the link
            if (remaining2<>nil) then
            begin
                  remaining2[0]:=#0;
// seperate the link from the rest of the page
                  myLink.setLinkString(remaining,isLinkOk);                    //
assign the link to myLink.linkString
                  if isLinkOk=True then
                  begin
                        if (isAcceptedType()=true ) then
```

```
                        begin
                  // if the new link is an acceptable one

                              if (pos(BH,mylink.LinkURL)>0) then
                              begin
                 // if the new link is in the domain

                                     EnterCriticalSection(LinkCount);
 // add the link to the links list, if it is not already there

isExist:=fmyPages.addLink(mylink.CurrentURL,mylink.LinkURL);
                                     LeaveCriticalSection(LinkCount);
                                     if (isExist=false) then
                                     begin
                          // if the link is not processed before
 EnterCriticalSection(ReadSection);
                       // read the global number of link count
                        nofl:=fNumberOFLine;
                        inc(fNumberOFLine);
LeaveCriticalSection(ReadSection);
EnterCriticalSection(LinkWrite);
  // Add the link to the the string grid
fReadMemo.RowCount:=nofl+1;
fReadMemo.Cells[1,nofl]:=format('T =%s',[mylink.CurrentURL]);
fReadMemo.Cells[2,nofl]:=mylink.LinkURL;
fReadMemo.Cells[0,nofl]:=IntToStr(nofl);
LeaveCriticalSection(LinkWrite);

EnterCriticalSection(CurrentIndex);
     cri:=CI;
LeaveCriticalSection(CurrentIndex);

EnterCriticalSection(ReadSection);
            // Update the info on the form
            // increment number of links found var.
    if cri<form1.nol then
    form1.Label2.Caption:=Format('%d/%d',[cri,fNumberOfline]);
LeaveCriticalSection(ReadSection);
```

```
                    end;
                end;
            end;
          end;
            // goto next link
          end;
            remaining:=remaining2+1;
            remaining:=AnsiStrpos(remaining,'href');
     end;
end;
procedure TForm1.Start();
begin
      // Create threads
     fmyPages:=TmyPages.Create(nol,nodp,afilename);
     SG.Cells[1,1]:='Root';
     SG.Cells[2,1]:=domain;
     fNumberOFLine:=SG.RowCount;
     Application.ProcessMessages;
     with TWCThread.Create(SG,1,nol,BaseHost) do
               onTerminate:=Don;
     Application.ProcessMessages;
     TC:=2;
     while (TC<noft+1) do
     begin
          with TWCThread.Create(SG,TC,nol,BaseHost) do
               onTerminate:=Don;
          Application.ProcessMessages;
          inc(TC);
     end;
end;

procedure TForm1.don(sender:Tobject);
begin
      // this procedure will be call when a thread terminates
     dec(TC);
     if TC=1 then
     begin
          pb.Position:=pb.Max;
```

```
form1.Label2.Caption:=Format('%d/%d',[nol,fNumberOfline]);
          Status:='Done';
          fmyPages.WritetoFile(ofilename);
          DeleteCriticalSection(CurrentIndex);
          DeleteCriticalSection(ReadSection);
          DeleteCriticalSection(ThreadStatus);
          DeleteCriticalSection(LinkWrite);
          DeleteCriticalSection(LinkCount);
     end;
end;
procedure TForm1.Options1Click(Sender: TObject);
begin
     form2.showmodal;
     readSettings();
end;

procedure TForm1.Start1Click(Sender: TObject);
begin
     Start();
end;

procedure TForm1.ToolButton1Click(Sender: TObject);
begin
     pb.Position:=0;
     readSettings();
     formatStringGrid();
     CI:=0;
     TC:=0;
     DT:=0;       // # of threads in download process
     memo1.Lines.Add('______________________________');
     memo1.lines.add('domain = ' + domain);
     memo1.lines.add('# of threads = ' + inttostr(noft));
     memo1.lines.add('# of links   = ' + inttostr(nol));
     memo1.lines.add('matrix file      = ' + afilename);
     memo1.lines.add('links file       = ' + ofilename);
     memo1.lines.add('# of precision = ' + inttostr(nodp));
     Start();
end;
```

```
procedure TForm1.readSettings();
begin
     with form2 do
     begin
          noft:=StrToInt(options.Cells[2,1]); // no of threads
          aFileName:=options.Cells[2,4];
          oFileName:=options.Cells[2,5];
          nol:=StrToInt(options.Cells[2,2]);
            // no of links to search
          pb.max:=nol;
          nodp:=StrToInt(options.Cells[2,3]); // decimal points
          domain:=options.cells[2,6];
     end;
end;

procedure Tform1.formatStringGrid();
begin
     SG.Cols[0].Clear;
     SG.Cols[1].Clear;
     SG.Cols[2].Clear;
     SG.rowcount:=2;
     SG.ColWidths[0]:=40;
     SG.ColWidths[1]:=260;
     SG.ColWidths[2]:=400;
     SG.ColWidths[3]:=200;
     SG.ColWidths[4]:=300;
     SG.Cells[0,0]:='ID';
     SG.Cells[1,0]:='Form';
     SG.Cells[2,0]:='To';
     SG.Cells[3,0]:='Info';
     SG.Cells[4,0]:='Details';
end;
procedure TForm1.setDomain(value:String);
var
   temp:array[0..255] of char;
   hostStart,L,R:PChar;
begin
```

```
      fdomain:=pchar(value);
      hostStart:=pchar(value)+7;
      hostStart:=strScan(hostStart,'/');
      if hostStart<>nil then
         hostStart[0]:=#0;
      hostStart:=pchar(value)+7;
      L:=strScan(hostStart,'.');
      R:=strRScan(hostStart,'.');
      while (L<>R) do
      begin
           hostStart:=L+1;
           L:=strScan(hostStart,'.');
           R:=strRScan(hostStart,'.');
      end;
      BaseHost:=hostStart;
end;
procedure TForm1.FormShow(Sender: TObject);
begin
      formatStringGrid();
end;

procedure TForm1.setStatus(value:String);
begin
      StatusBar1.SimpleText:=value;
      Application.ProcessMessages;
end;

end.


unit myClasses;
interface
uses classes,sysutils;
type
    TmyPages = class
    Private
           fURL:TStringList;
           fDownloaded:TStringList;
```

```
            fNumberofOutLinks:TStringList;
            fA:array of array of integer;
            fn:integer;       // number of links
            fnodp:integer;    // number of decimals
            flinks:TStringList;
            MFileName:String;  // file name for the matrix
            function addItem(URL:String):integer;
            procedure IncreaseNOOL(ind:integer);
            procedure WriteAtoFile(fileName:string);
    public
           constructor Create(n,nodp:integer;fName:String);
           destructor Destroy();
           function isDownloaded(URL:String):integer;
           function getIndex(URL:String):integer;
           function getNOOL(URL:String):integer;
           function addLink(from:String;_to:String):boolean;
           procedure WritetoFile(FileName:string);
end;
type
    TURL=class
    private
            fURL:String;
            fHostInfo:String;
            fDomain:String;
            fFileInfo:String;
            fDirInfo:String;
    public
           procedure setURL(value:string);
           procedure setdirInfo(value:string);
           function getHostName():String;
           function getFileName():String;
           function getDomain():String;
    published
              property URL:string read fURL write setURL;
              property dirInfo:string read fDirInfo write setdirInfo;
end;
type
    TLink=class
```

```
    private
            fCurrentURLString:String;
            fCurrentURL:TURL;
            fLink:TURL;
            function getCurrentHostName():string;
            function getCurrentDomain():string;
            function getCurrentFile():string;
            function getCurrentDir():string;
            function getCurrentURL():string;
            function getLinkURL():string;
            function getLinkHostName():String;
            function getLinkDomain():String;
            function getLinkFile():String;
            function getLinkDir():String;
            procedure setLinkDirs(value:string);
            procedure setCurrentURL(value:String);
    public
           fLinkString:String;
           procedure setLinkString(value:String;var islinkOk:boolean);
           constructor create;
           destructor destroy;override;
    published
              property CurrentURL:string read getCurrentURL write
setCurrentURL;
              property LinkURL:String read getLinkURL;
              property LinkHostName:String read getLinkHostName;
              property LinkDomain:String read getLinkDomain;
              property LinkFile:String read getLinkFile;
              property LinkDir:string read getLinkDir;
              property CurrentHostName:String read getCurrentHostName;
              property CurrentDomain:String read getCurrentDomain;
              property CurrentFile:String read getCurrentFile;
              property CurrentDir:String read getCurrentDir;
end;
implementation

function TmyPages.addLink(from:String;_to:string):boolean;
var
```

```
    fromindex:integer;
    _toIndex:integer;
begin
      if flinks.IndexOf(Format('%s-%s',[from,_to]))<0 then
      begin
            flinks.add(Format('%s-%s',[from,_to]));
            fromIndex:=fURL.IndexOf(from);
            _toIndex:=fURL.IndexOf(_to);
            if fromIndex<0 then
            begin
                  fromIndex:=addItem(from);
                  fDownloaded.Add(from);
                  fNumberofOutLinks.Add(IntToStr(0));
            end;
            if _toIndex<0 then
            begin
                  _toIndex:=addItem(_to);
                  fNumberofOutLinks.Add(IntToStr(0));
            end;
            if _toIndex<fn then
            begin
               IncreaseNOOL(fromIndex);
               fa[_toIndex,fromIndex]:=1;
            end;
            result:=false;
      end
      else
           result:=true;
end;

procedure TmyPages.WritetoFile(fileName:string);
var
  F: TextFile;
  i: Integer;
  counter:integer;
begin
      counter:=fURL.Count;
    AssignFile(F, fileName);
```

```
    ReWrite(F);
    for i:=0 to fn-1 do
    begin
            WriteLn(F,Format('%d. %s',[i,fURL[i]]));
    end;
    CloseFile(F);
    writeAtoFile(MFileName);
end;

procedure TmyPages.WriteAtoFile(fileName:string);
var
  j,i: Integer;
  F: TextFile;
  ad:double;
begin
    AssignFile(F, fileName);
    ReWrite(F);
    WriteLn(F,'A');
    WriteLn(F,Format('%d %d',[fn,fn]));
    for i:=0 to fn-1 do
    begin
         for j:=0 to fn-1 do
         begin
              case fA[i,j] of
              0:Write(F,0.0:fnodp+2:fnodp,' ');
              1:begin
                     if StrToInt(fNumberofOutLinks[j])<>0 then
                         ad:=1/StrToInt(fNumberofOutLinks[j])
                     else
                         ad:=9;
                Write(F,ad:fnodp+2:fnodp,' ');
               end;
              end;
         end;
         Write(F,#13#10);
    end;
    for i:=0 to fn-1 do
    begin
```

```
          Write(F,fNumberofOutLinks[i],' ');
     end;
     CloseFile(F);
end;

function TMyPages.getIndex(URL:String):integer;
begin
      result:=fURL.IndexOf(URL);
end;

function TMyPages.getNOOL(URL:String):integer;
var
   index:integer;
begin
      index:=getIndex(URL);
      if (index>=0) then
         result:=StrtoInt(fNumberofOutLinks[index])
      else
          result:=index;
end;
constructor TmyPages.Create(n,nodp:integer;fName:String);
var
   i,j:integer;
   sifir:string;
begin
      fn:=n;
      fnodp:=nodp;
      fURL:=TStringList.Create;
      fDownloaded:=TStringList.Create;
      MFileName:=fName;
      fNumberofOutLinks:=TStringList.Create;
      SetLength(fa,n,n);
      sifir:=IntToStr(0);
        for i:=0 to n-1 do
        begin
             for j:=0 to n-1 do
             begin
                  fa[i,j]:=0;
```

```
            end;
        end;
      flinks:=TStringList.Create;
end;

destructor TMyPages.Destroy();
begin
      fURL.Destroy;
      fNumberofOutLinks.Destroy;
      flinks.Destroy;
end;

function TmyPages.isDownloaded(URL: String): integer;
begin
      result:=fDownloaded.IndexOf(URL);
end;
function TMyPages.addItem(URL:String):integer;
begin
      // the URL is just being processed (the first time)
      result:=fURL.Add(URL);
end;
procedure TMyPages.IncreaseNOOL(ind:integer);
var
   currentnofl:integer;
begin
      currentnofl:=StrToInt(fNumberofOutLinks[ind]);
      fNumberofOutLinks[ind]:=IntToStr(currentnofl+1);
end;
{ TLink }
constructor TLink.create;
begin
      fCurrentURL:=TURL.Create;
      fLink:=TURL.Create;
end;
destructor TLink.destroy;
begin
      fCurrentURL.Destroy;
      fLink.Destroy;
```

```
end;

procedure TLink.setLinkDirs(value:string);
var
   filename,DirName,temp:string;
   DirEnd,fileStart,DirStart:PChar;
begin
     DirName:=value;
     if (DirName[1]<>'/') then
        DirName:='/'+DirName;
     temp:=DirName;
     DirStart:=Pchar(temp);;
     if (pos('.',DirStart)>0) then
     begin
          fileStart:=strRScan(PChar(temp),'/')+1;
          fileName:=fileStart;
          if fileStart<>nil then
             fileStart[0]:=#0;
          DirName:=DirStart;
     end
     else
     begin
          fileName:='';
          DirName:=DirStart;
     end;
     DirEnd:=StrEnd(DirStart)-1;
     if DirEnd[0]<>'/' then
     begin
          DirNAme:=DirName + '/';
     end;
     fLink.dirInfo:=DirName;
     flink.URL:='http://' + getCurrentHostName() + fLink.dirInfo
+FileName;
end;

function TLink.getCurrentDir: string;
begin
     getCurrentDir:=fCurrentURL.dirInfo;
```

```
end;
function TLink.getCurrentDomain: string;
begin
     result:=fCurrentURL.getDomain();
end;
function TLink.getCurrentFile: string;
begin
     result:=fCurrentURL.getFileName();
end;
function TLink.getCurrentHostName: string;
begin
     result:=fCurrentURL.getHostName();
end;

function TLink.getCurrentURL: string;
begin
     result:=fCurrentURLString;
end;
function TLink.getLinkURL: string;
begin
     result:=fLink.URL;
end;

procedure TLink.setCurrentURL(value: String);
begin
     if (value<>fCurrentURLString) then
     begin
          fCurrentURLString:=pchar(value);
          if Assigned (fCurrentURL) then
             fCurrentURL.URL:=value
          else
              exception.Create('TURL object could not be created');
     end;
end;

procedure TLink.setLinkString(value: String;var islinkOk:boolean);
var
   cDir,filename,DirName,temp:string;
```

```
   cDirP,DirStart,tempP:PChar;
   tempvalue:string;
   tempDir,td:PChar;
   cuDir:String;
   curDir:String;
   tempDir2:String;
   t2:PChar;
begin
     if strlen(pchar(value))>0 then
     begin
          isLinkOk:=true;
     if (value[1]='/') then
         fLinkString:=PChar(value)+1
     else
         fLinkString:=value;
     tempvalue:=value;
     if pos('http://',tempvalue)>0 then
     begin
          // a full link
          // http://www.lamar.edu
          // http://www.lamar.edu/index.html
          // http://www.lamar.edu/academics
          flink.URL:=tempvalue;
     end
     else
     begin
          if pos('/',tempvalue)>0 then
          begin
            // This link is not in the same directory with the
            // page it has been linked.
            // the file name is the same
               if pos('..',tempvalue)>0 then
               begin
           // This link's directory comes before the
           // directory that the page is located
           // example
           // fromLink = 'http://www.lamar.edu/academics/
           // linkString = '../test/search.htm';
```

```
// the full link should be =
  //http://www.lamar.edu/test/search.htm
         cDir:=getCurrentDir();
         cDirP:=pchar(cDir);
         tempP:=strEnd(cDirP)-1;
         if tempP<>nil then
            tempP[0]:=#0;
         temp:=tempvalue;
         DirStart:=pchar(tempvalue);
         while (pos('../',DirStart)>0) do
         begin
              dirStart:=strScan(DirStart,'/')+1;
              tempP:=strRScan(cDirP,'/');
              if tempP<>nil then
                 tempP[0]:=#0;

         end;
         DirName:=dirStart;
         cDir:=cDirP;
         cDir:=cDir+'/';
         setLinkDirs(cDir+DirName);
    end
    else
    begin
      // This link's directory comes after the
      // directory that the page is located
      // example
      // fromLink = 'http://www.lamar.edu/
      // linkString = 'academics/search.htm';
      // the full link should be =
         // http://www.lamar.edu/academics/search.htm
         temp:=tempvalue;
         DirStart:=PChar(tempvalue);
         tempDir2:=DirStart;
         cuDir:=getCurrentDir();
         curDir:=cuDir;
         tempDir:=AnsiStrPos(pchar(tempDir2),'/');
         while ((tempDir<>nil) and(strlen(tempDir)>0) ) do
```

```
          begin
               if tempDir<>nil then
               begin
                    tempDir[0]:=#0;
                    td:=pchar(tempDir2);
                    if Pos(td,pchar(cuDir))>0 then
                    begin
                    t2:=AnsiStrPos(Pchar(cuDir),td);
                         t2:=t2-1;
                         t2[0]:=#0;
                         t2:=pchar(cuDir);
                         cuDir:=t2;
                    end
                    else
                    begin
                         tempDir2:=td+strlen(td)+1;
           tempDir:=AnsiStrPos(pchar(tempDir2),'/');
                    end;
               end;
          end;
          if (DirStart[0]='/') then
             DirStart:=DirStart+1;
          setLinkDirs(cuDir+DirStart);
     end;
end
else
begin
// This link is in the same directory with the
// page it has been linked.
// example
// fromLink = 'http://www.lamar.edu/
// linkString = 'search.htm';
// the full link should be =
  // http://www.lamar.edu/search.htm
// nothing needs to be done, except setting fromURL
     fileName:=tempvalue;
     setLinkDirs(getCurrentDir()+fileName);
```

```
               //flink.URL:='http://'+getCurrentHostName() +
getCurrentDir()+ fileName;
          end;
     end;
    end
    else
        isLinkOk:=false;
end;
function TLink.getLinkDir: String;
begin
     result:=fLink.dirInfo;
end;
function TLink.getLinkDomain: String;
begin
     result:=flink.getDomain;
end;
function TLink.getLinkFile: String;
begin
     result:=flink.getFileName;
end;

function TLink.getLinkHostName: String;
begin
     result:=flink.getHostName;
end;

{ TURL }
function TURL.getDomain: String;
begin
     result:=fDomain;
end;

function TURL.getFileName: String;
begin
     result:=fFileInfo;
end;

function TURL.getHostName: String;
```

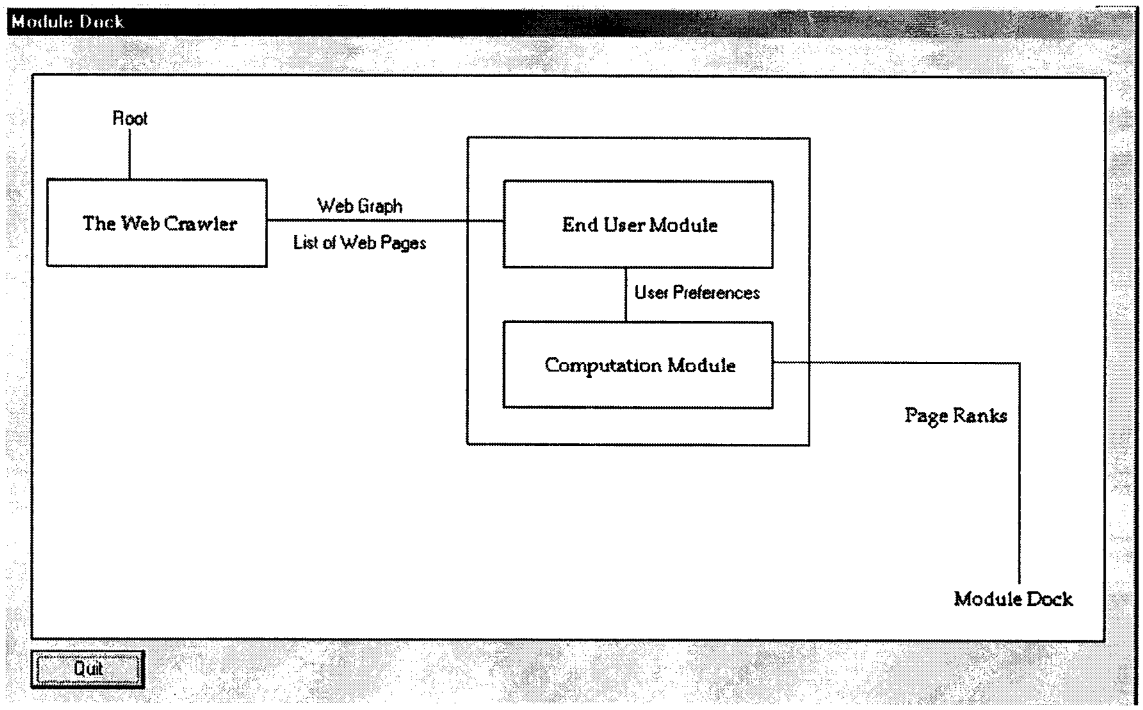
```
begin
     result:=fHostInfo;
end;

procedure TURL.setdirInfo(value: string);
var
   L:PChar;
   vP:PChar;
begin
     vP:=pchar(value);
          if strlen(vp)>0 then
          begin
               L := StrEnd(vp)-1;
               if L<>nil then
               begin
                    if (L[0]<>'/') then
                       fDirInfo:=value+'/'
                    else
                        fDirInfo:=value;
               end;
               if (vP[0]<>'/') then
                  fDirInfo:='/'+fDirInfo;
          end
          else
              fDirinfo:='/';
end;

procedure TURL.setURL(value: string);
var
   filestart,hostStart,dirStart,L,R:Pchar;
begin
     // URL should be a full link
     // example
     // http://www.lamar.edu/academics/search.htm
     if (fURL<>value) then
     begin
          hostStart:=PChar(value)+7;
          dirStart:=StrScan(hostStart,'/');
```

```
          if dirStart<>nil then
          begin
               if (pos('.',dirStart)>0) then
               begin
                    fileStart:=StrRScan(dirStart,'/')+1;
                    fFileInfo:=fileStart;
                    if fileStart<>nil then
                       fileStart[0]:=#0;
               end
               else
               begin
                    fFileInfo:='';
               end;
               DirInfo:=dirStart;
               if dirStart<>nil then
                  dirStart[0]:=#0;
          end
          else
          begin
              fFileInfo:=dirStart;
              DirInfo:='';
          end;
          fHostInfo:=hostStart;
          L:=strScan(hostStart,'.');
          R:=strRScan(hostStart,'.');
          while (L<>R) do
          begin
               hostStart:=L+1;
               L:=strScan(hostStart,'.');
               R:=strRScan(hostStart,'.');
          end;
          fDomain:=hostStart;
     end;
     fURL:='http://'+fHostInfo+DirInfo+fFileInfo;
end;
end.
```

Appendix B

Screen Shots

1.The Module Dock (MD)

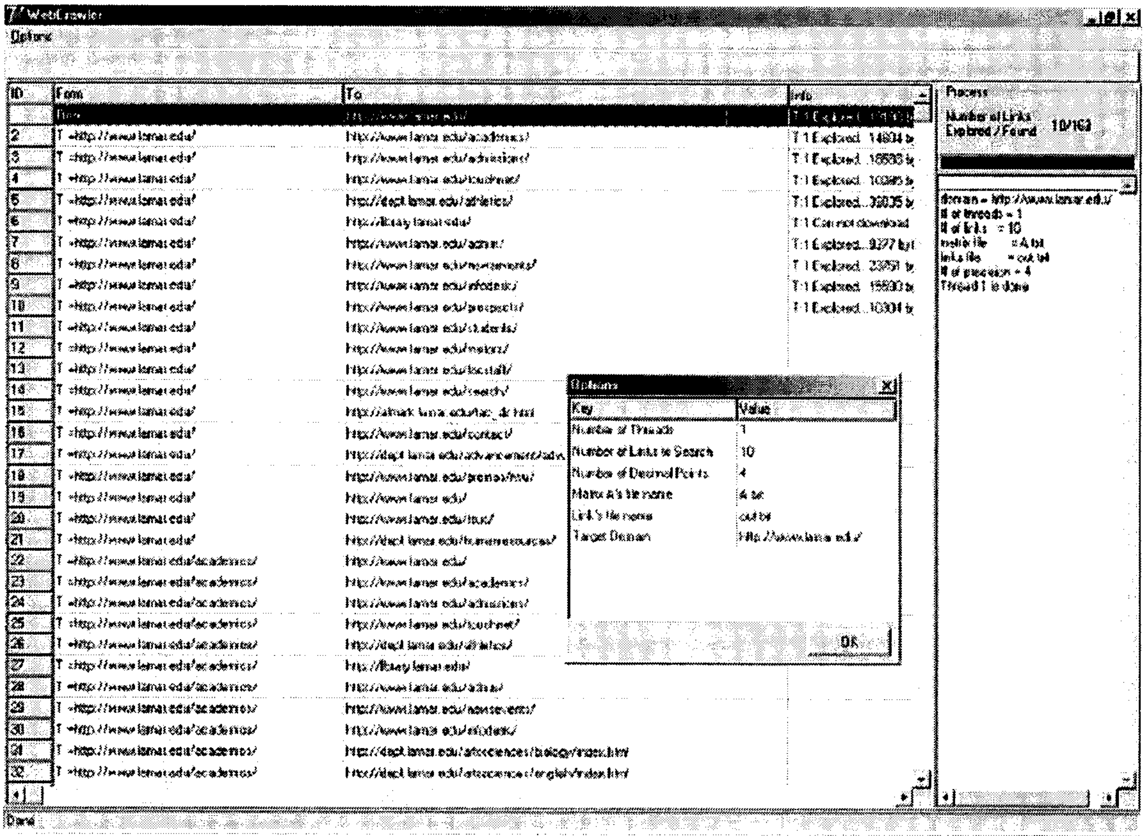


2. The Results

Results

URLs	Rank
http://almark.lamar.edu/fac_dir.html	0.71
http://www.lamar.edu/search/	0.64
http://www.lamar.edu/touchnet/	0.61
http://www.lamar.edu/admin/	0.54
http://www.lamar.edu/facstaff/	0.53
http://www.lamar.edu/visitors/	0.42
http://www.lamar.edu/students/	0.36
http://www.lamar.edu/academics/	0.31
http://www.lamar.edu/tsus/	0.31
http://www.lamar.edu/promos/twu/	0.27
http://www.lamar.edu/admissions/	0.26
http://www.lamar.edu/prospects/	0.23
http://library.lamar.edu/	0.22
http://dept.lamar.edu/advancement/advweb/frontpg.html	0.22
http://www.lamar.edu/contact/	0.19
http://dept.lamar.edu/humanresources/	0.15
http://www.lamar.edu/	0.13
http://dept.lamar.edu/athletics/	0.12
http://www.lamar.edu/newsevents/	0.11
http://www.lamar.edu/infodesk/	0.11

3. WebCrawler



3. User Preference - Input Interface