Personalized Web Search using Browsing History and

Domain knowledge

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

in

Information Technology

under the Supervision of

Ms.Reema Aswani

By

Faiz Hussain(111447)

to

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled "**Personalized web search using Browsing History and Domain Knowledge**",submitted by Faiz Hussain in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date: **Supervisor's Name : Ms Reema Aswani**

Designation :

Acknowledgement

I thankfully acknowledge the contribution of various different journals, magazines and books, from which some of the material have been collected to enrich this project. I am very much grateful to Ms. Reema Aswani,for his guidance and support throughout my project,and also for his constant inspiration from the very first day when I had started the work for this project.
I would also like to give my deepest and sincerest thanks to my friends for guiding me and telling about modification that can be done to this project.

Date: Name of the student :Faiz Hussain

Table of Content

List of Figures

List of Tables

Abstract

Abstract-Generic search engines are important for retrieving relevant information from web. However these engines follow the "one size fits all" model which is not adaptable to individual users. Personalized web search is an important field for tuning the traditional IR system for focused information retrieval. This project is an attempt to improve personalized web search. User's Profile provides an important input for performing personalized web search. This paper proposes a framework for constructing an Enhanced User Profile by using user's browsing history and enriching it using domain knowledge. This Enhanced User Profile can be used for improving the performance of personalized web search. In this project we have used the Enhanced User Profile specifically for suggesting relevant pages to the user. The experimental results show that the suggestions provided to the user using Enhanced User Profile are better than those obtained by using a User Profile.

CHAPTER 1. INTRODUCTION

1.1. About Search Engine

With the development of World Wide Web, web search engines have contributed a lot in searching information from the web. They help in finding information on the web quick and easy. But there is still room for improvement. Current web search engines do not consider specific needs of user and serve each user equally. It is difficult to let the search engine know what the user actually want. Generic search engines are following the "one size fits all" model which is not adaptable to individual users.

When different users give same query, same result will be returned by a typical search engine, no matter which user submitted the query. This might not be appropriate for users which require different information. While searching for the information from the web, users need infonl1ation based on their interest. For the same keyword two users might require different piece of information. This fact can be explained as follows: a biologist and a programmer may need information on "virus" but their fields are is entirely different. Biologist is searching for the "virus" that is a microorganism and programmer is searching for the malicious software. For this type of query, a number of documents on distinct topics are returned by generic search engines. Hence it becomes difficult for the user to get the relevant content. Moreover it is also time consuming. Personalized web search is considered as a promising solution to handle these problems, since different search results can be provided depending upon the choice and information needs of users. It exploits user information and search context to learning in which sense a query refer.

1.2. Proposed Architecture

In order to perform Personalized Web search it is important to model User's need/interest. Construction of user profile is an important part for personalized web search. User profiles are constructed to model user's need based on his/her web usage data.This project proposes an architecture for constructing user profile and enhances the user profile using background knowledge. This Enhanced User Profile will help the user to retrieve focused information. It can be used for suggesting good Web pages to the user based on his search query and background knowledge.

CHAPTER 2. RELATED WORK

2.1. FrameWork

Framework for Personalized search engine consists of user modeling based on user past browsing history or application he/she is using etc. And then use this context to make the web search more personalized. This section presents different approaches and the related work done in the field of Personalized Web search.

2.2. Different Scientist Perceptions

For providing personalized search results, *Micro Speretta* implemented a wrapper around the search site that collects information about user's search activity and builds user profile by classifying collected information (queries or snippets). They have used these profiles to re-rank the search results and the rank-order of the user-examined results before and after re-ranking were compared. They found that user profiles based on queries and user profiles based on snippets both were equally effective and re-rank gave 34% improvement in compare to rank-order.

Fang Liu identified that current web search engines do not consider the special needs of user or interests of user and proposes a novel technique which uses search history of user to learn user profiles. This work uses user's search history for learning of user profile and category hierarchy for learning of a general profile and then combines both profiles to categorize user's query to represent user's search intention and to disambiguate the words used in query.

Chunyan Liang also identifies that different users may have need of different special information, when they use search engines and techniques of personalized web search can be used to solve the problem effectively. Three approaches Rocchio method, k-

Nearest method and Support Vector Machines have been used in [3] to build user profile to present an individual user's preference and found that k-Nearest method is better than others in terms of its efficiency and robustness.

Xuwei Pan suggested a context based personalized web search model. In this paper the authors have given a personalized web search outcome which is in accordance with the need of user in various situations. The analysis of model has resulted in three concepts to implement the model, which is semantic indexing for web resources ,modeling and acquiring user context and semantic similarity matching between web resources and user context. The author has defined it as context based adaptive personalized web search.

K. W. T. Leung have proposed a Personalized Web search model with location preferences. In this paper the location and content concept has been separated and is organized into different ontology to make an ontology-based, multi-facet (OMF) profile which is captured by web history and location interest. This model actually gives results by outlining the concepts in accordance with the preference of user. By keeping the diverse interest of the users in mind, location entropy is introduced for finding the degree of interest and information related to location and query. The personalized entropies actually estabilize the relevant output content and location content. At last, an SVM based on the ontology is derived which can be used for future purpose for ranking or reranking. The experiments shows that the results produced by OMF profiles are more accurate in comparison with the ones which use baseline method.

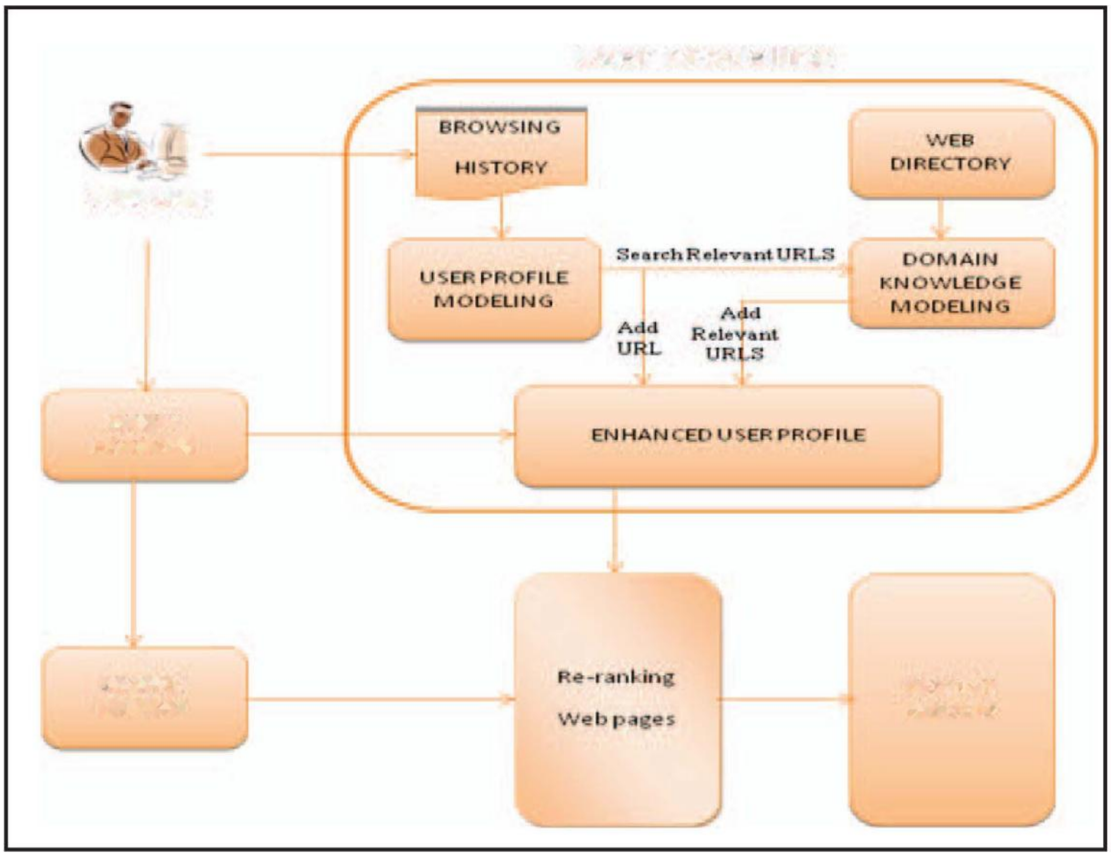
O. Shafiq have proposed a personalized web search model that combines community based and content based evidences based on novel ranking technique. Nowadays, uploading data on internet has become a daily activity. A massive amount of data is uploaded in the form of web pages, news, and blogs etc. on a regular basis. So, it becomes very difficult for the user to search for relevant content. Not only for users but also for search engines like Google and Yahoo it becomes difficult. Information overload is the only reason behind this difficult situation. Other than this user's preference is the second problem, which is not taken into consideration while producing the results. The author tried to solve this problem through this model which produce results on the basis of preference and interest of the user.

In this project, authors proposed a unique approach to find out the interest and preference of the user. It's a two way approach, first it will find out the activities of user through his/her profile in social networking sites. Secondly, it will find out information from what the social networking sites provide to the user through friends and community. Based on the results, user's interest and preference will be prioritized by the web search or it is personalized.

CHAPTER 3. FRAMEWORK FOR PROPOSED SYSTEM

3.1. General Framework

We propose a framework for personalized web search which considers individual's interest into mind and enhances the traditional web search by suggesting the relevant pages of his/her interest. We have proposed a simple and efficient model which ensures good suggestions as well as promises for effective and relevant information retrieval. In addition to this, we have implemented the proposed framework for suggesting relevant web pages to the user.



CHAPTER 4. PROJECT CONTENTS

4.1. Main Contents

Our system considers user's profile (based on user's weblogl navigation browsing history) and Domain Knowledge in order to perform personalized web search. Using a Domain Knowledge, the system stores information about different domain. Information obtained from User Profile is classified into these specified categories. The learning agent learns user's choice automatically through the analysis of user navigation/browsing history, and creates/updates enhanced User Profile conditioning to the user's most recent choice. Once the user inputs query, the system provides good suggestions for personalized web search based on enhanced user profile. Further our model makes good use of the advantages of popular search engines, as it can re-rank the results obtained by the search engine based on the enhanced user profile.

CHAPTER 5. DESRIPTION OF CONTENTS

5.1. Domain Knowledge Modeling

Domain knowledge is the background knowledge that we used to enhance the user profile. The source which we have used for preparing Domain Knowledge is DMOZ directory. For preparing Domain Knowledge, first we have crawled the web pages from DMOZ directory for some specified categories, where each category is represented by collection of URL's present in that category. After crawling, we have extracted the keywords from the crawled web pages. The collections of keywords form the vocabulary for the crawled pages. Now we form a term category matrix, which specifies weight of each term in each category. The weight may be represented by frequency of the term in that category. Here Wjj represents number of times the term tj is present in Category Catej. The matrix may be represented as follows:

Term/Category	**Cate1**	**Cate2**	**Cate3**	**………..**	**Caten**
t1	**W11**	**W12**	**W13**	**………..**	**W1n**
t2	**W21**	**W22**	**W23**	**……….**	**W2n**
t3	**W31**	**W32**	**W33**	**………**	**W3n**
……	**………**	**……..**	**……..**	**………**	
Tm	**Wm1**	**Wm2**	**Wm3**	**…….**	**Wmn**

Table 1. Terms – Category Matrix (TCM)

5.2. User Profile Modeling

User profile is used to reflect user's interest and predict their intentions for new queries. User Profile also helps to deal with ambiguous queries. To create the user profile, we need to classify the web pages accessed by a user into particular category. Alchemy API has been used for classifying web pages. Alchemy API classifies a web page by giving it a particular category along with confidence (numerical value) which shows its probability of belonging to that particular category. If the web page is classified with confidence above the specified threshold level then only we have consider that page to contribute for that category. As we are using DMOZ for background knowledge, we have to map these Alchemy categories to DMOZ categories. Thus in our model, a User Profile is represented as a category preference vector, where weight of each category represents user's interest in that category. As shown in the Figure , users browsing history is used to build user profile. When the number of web pages browsed by the user grows above the specified threshold, the learning agent updates user profile. User interest will thus be represented by fix number of categories weights. It can be denoted by

$$U= \{cw_1, cw_2, cw_3, \ldots.., cw_m\}$$

Where, CWj will be the number of web pages of category i visited by that user, normalized by maximum number of page visits among all categories.

Alchemy Categories	**DMOZ Categories**
Arts & Entertainment	Arts
Business	Business
Computers & Internet	Computers
Culture & Politics	Regional
Gaming	Games
Health	Health
Recreation	Recreation
Science & Technology	Science

Sports	Sports

Table 2. Alchemy API to DMOZ Category Mapping

For modeling User Profile we have used Vector Space Model (VSM). We consider all the webpages present in browsing history of particular user. Each web page corresponds to a specific document. The outcome of vector space model is a term document matrix (TDM) which represents each webpage/document as a feature vector of terms. Here we consider each document as a URL.

	D1	**D2**	**D3**	**………**	**Dn**
T1	**W_{11}**	**Wl2**	**W13**		**W1n**
T2	**W21**	**W22**	**W23**		**W2n**
T3	**W31**	**W32**	**W33**		**W3n**
……….					
Tm	**Wml**	**Wm2**	**Wm3**		**Wmn**

Table 3. Terms – Document Matrix (TDM)

5.3. Enhanced User Profile

Enhanced User Profile is an important part in our framework. An Enhanced User Profile improves the User Profile by using the Domain Knowledge. For preparing the Enhanced User Profile we have considered each URL of the User Profile, match it with Domain Knowledge URLs and add most relevant URLs to the Enhanced User Profile.

Following steps explain the process of preparing the Enhanced User Profile. Perform the following steps for each document (URL) in user profile :

- Select the URL from the User Profile.
- Add the URL to the Enhanced User Profile.
- Find the cosine similarity of this URL with the URLs present in user specific categories from the Domain Knowledgebase.

- Rank the URLs on descending order of cosine similarity.
- Retrieve top 20 URLs.
- Calculate the average of the cosine similarity of these top 20 URLs.
- From the top 20 URLs add only those URLs to the enhanced user profile whose similarity value is above the average value.

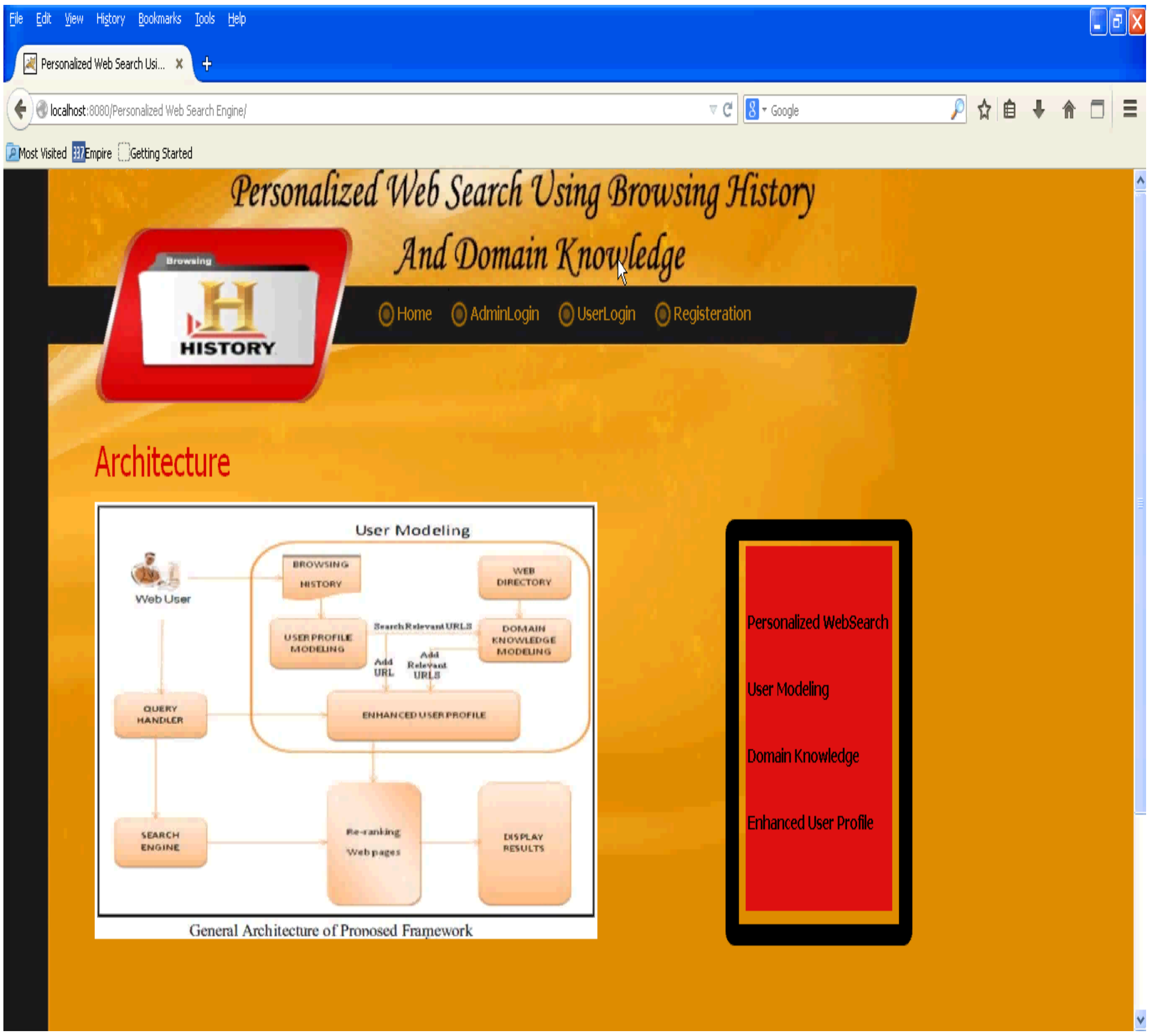
To summarize the process, for each URL (form user profile) most relevant URLs from the user specific Domain Knowledge category are added to prepare enhanced user profile.

The cosine formula used for the similarity of the URL u in User Profile to each web pages dj in Domain Knowledge is as follows:

$$cosine(d_j, u) = \frac{< d_j * u >}{\|d_j\| \times \|u\|}$$

A cosine similarity measure is the angle between the web page in User Profile u and the document vector dj.

5.4. Screenshot of Home Page



CHAPTER 6. EXPERIMENTAL RESULTS AND ANALYSIS

6.1. Statistics

In the absence of standard benchmark datasets which is suitable for our problem, we have designed our own dataset. In our Experiment, we have used the browsing history of 10 different users from our university, 6 from Computer department and 4 from Electronics department. Our Experiment is conducted for 50 queries of which 35 queries from Computer domain and 15 queries from Electronics domain.

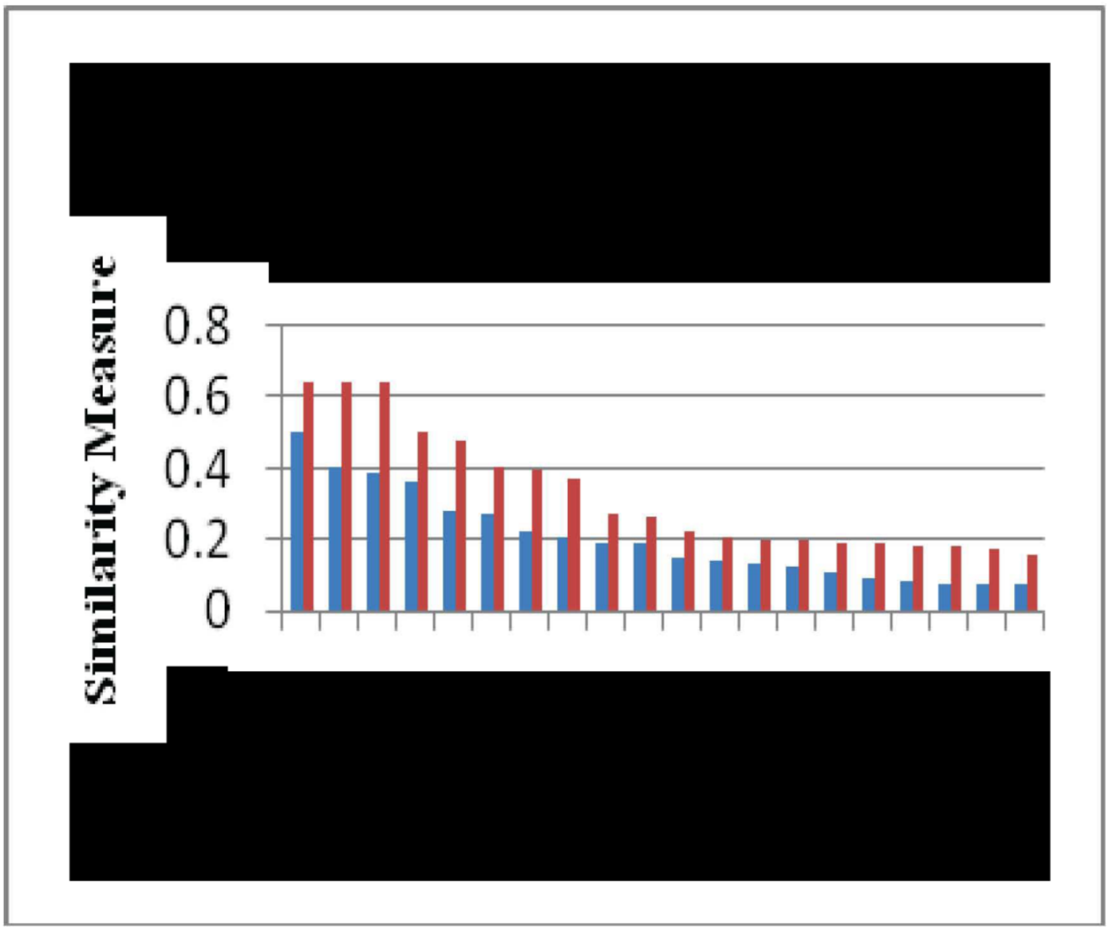
In order to collect the domain knowledge, we have crawled the datasets from DMOZ for the selected topics using Apache Nutch, while Apache Solr has been used for indexing crawled pages. By setting crawling parameters of Nutch we have restricted the crawling to specific DMOZ topic.

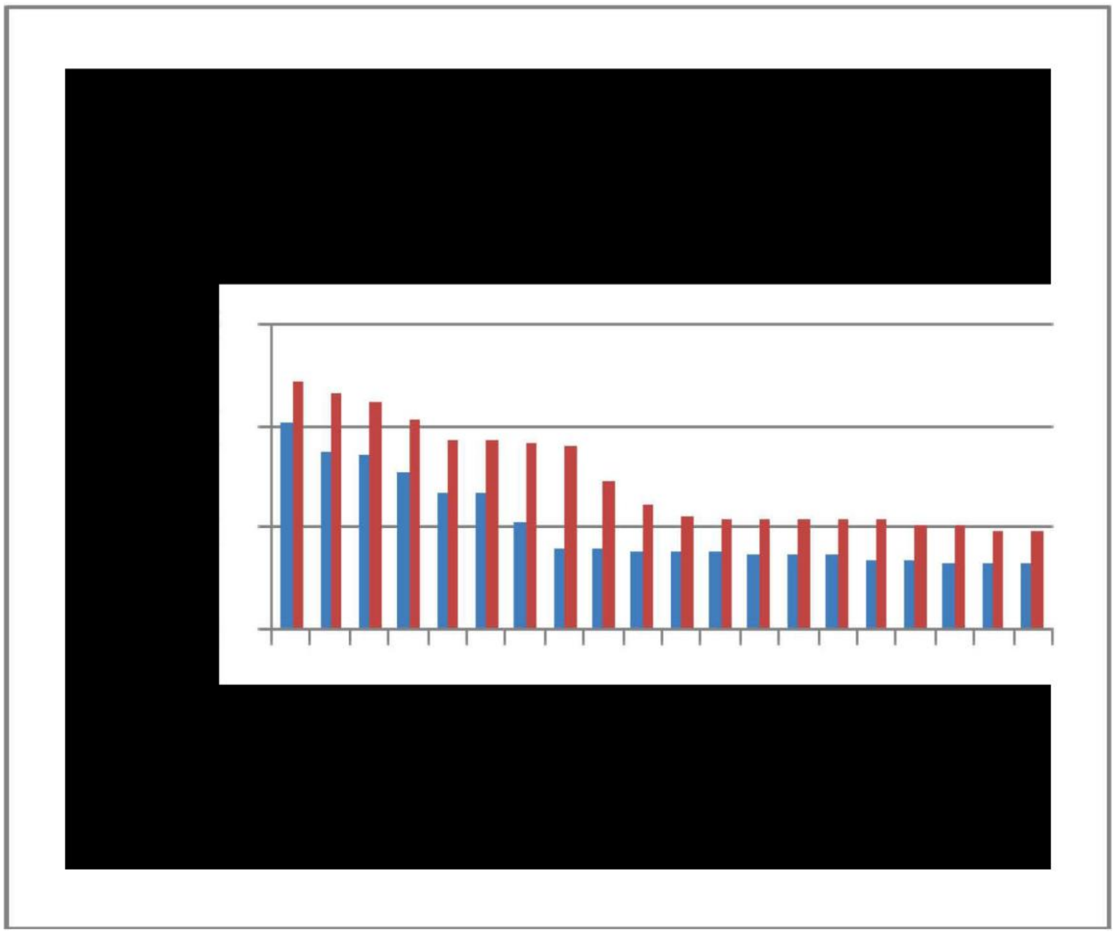
Using the information of user browsing history and domain knowledge, we create an Enhanced User Profile. Once the Enhanced User Profile is created, we take the user query and suggest the relevant web pages with respect the query. In our Experiment, we have used User Profile as a base case for suggesting the relevant pages and compared the results with the pages suggested from Enhanced User Profile. For each query, we suggest top 20 relevant documents from User Profile and for the same query we also suggest top 20 relevant documents from Enhanced User Profile. In order to compare the efficiency of the result, we compared the similarity of suggested documents with the user query.

In order to represent the result graphically, we have used the bar graph. The analysis of the result is done in 2 different ways. First one is the individual cosine similarities of suggested pages and the second one is the average cosine similarity obtained for top 20 suggested pages.

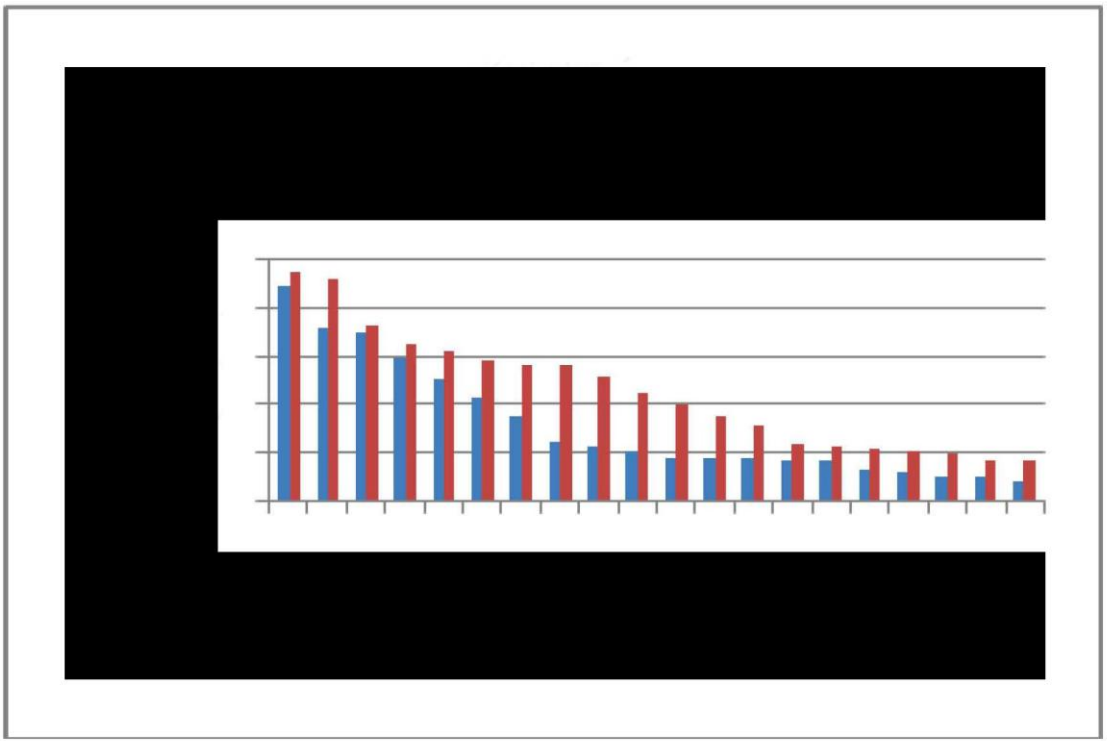
For each query, we draw a bar graph of the cosine similarity measure for each suggested web page. The Figure 4.l to Figure 4.4 shows the graph for Query1 to Query4.

QUERY 1

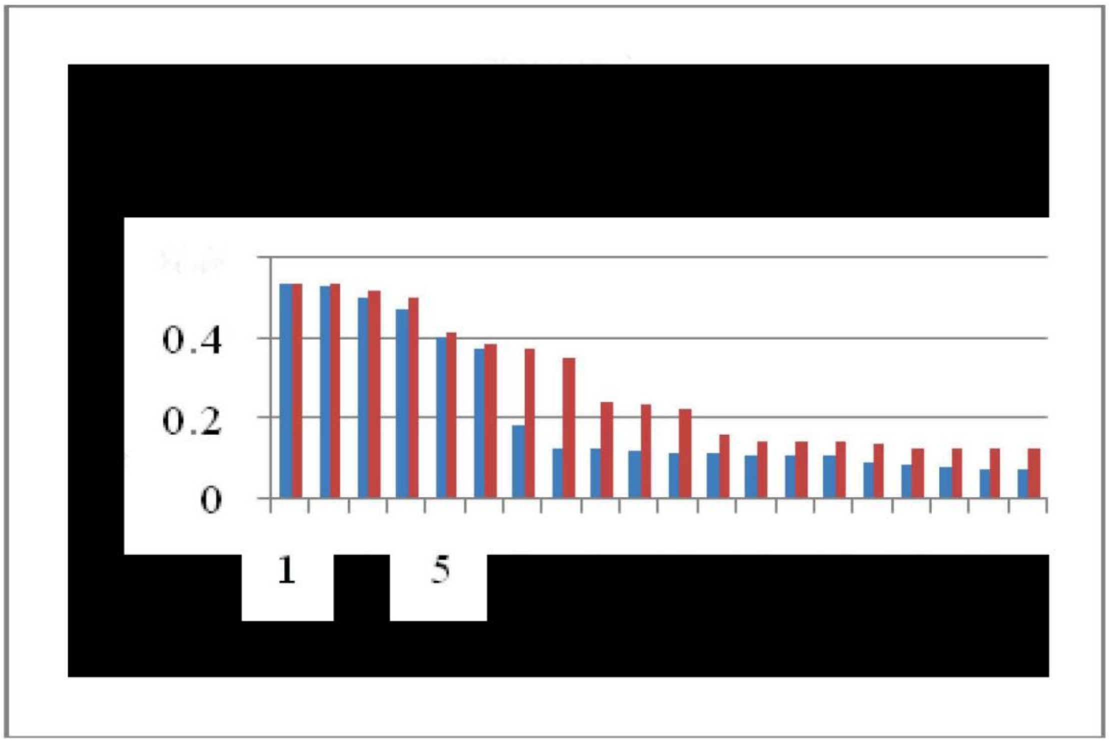




QUERY 2



QUERY 3



QUERY 4

In this section, we have analyzed the results for different Queries. For each query, we have retrieved top 20 relevant web pages with User Profile and Enhanced User Profile. As we can see clearly from the above figures (Query 1 to Query 4) that all the queries show the improved result for Enhanced User Profile as compared to those suggested using User Profile.

CHAPTER 7 . DOCUMENTS AS GEOMETRIC OBJECTS

7.1 How to rank documents for full text search

When we type a query into a search engine – say “Einstein on relativity” – how does the search engine decide which document to return? When the document is on the web ,part of that answer is provided by the PageRank algorithm, which analysis the link structure of the web to determine the importance of different webpages. But what should we do when the documents aren’t on the web, and there is no link structure? How should we determine which documents most closely match the intent of the query?

In this topic I explain the basic idea of how to rank different documents according to their relevance. They are based on the **VECTOR SPACE MODEL** for documents. The key idea is to transform search from a linguistic problem into a *geometric problem.* Instead of thinking of documents and queries as strings of letters, we adopt a point of view in which both documents and queries are represented as vectors in a vector space. In this point of view, the problem of determining how relevant a document is to a query is just a question of determining how *parallel* the query vector and the document vector are. The more parallel the vectors, the more relevant the document is.

This geometric way of treating documents turns out to be very powerful. It’s used by most modem search engines, including web search engines such as Google and Bing, as well as search libraries such as Lucene. The ideas can also be used well beyond search, for problems such as document classification, and for finding clusters of related documents. What makes this approach powerful is that it enables us to bring the tools of geometry to bear on the superficially very non-geometric problem of understanding text.

tf*idf forms the basis of scoring documents for relevance when querying a corpus, as in a search engine. It is the product of two terms: *term frequency* and *inverse document frequency*. Tf-idf is a transformation you apply to texts to get two real-valued vectors. You can then obtain the cosine similarity of any pair of vectors by taking their dot product and dividing that by the product of their norms. That yields the cosine of the angle between the vectors. TF-IDF is just a way to measure the importance of tokens in text; it's just a very common way to turn a document into a list of numbers (the term vector that provides one edge of the angle you're getting the cosine of).

To compute cosine similarity, you need two document vectors; the vectors represent each unique term with an index, and the value at that index is some measure of how important that term is to the document and to the general concept of document similarity in general.

If d2 and q are tf-idf vectors, then

$$\cos\theta = \frac{\mathbf{d_2} \cdot \mathbf{q}}{\|\mathbf{d_2}\| \, \|\mathbf{q}\|}$$

where θ is the angle between the vectors. As θ ranges from 0 to 90 degrees, cos θ ranges from 1 to 0. θ can only range from 0 to 90 degrees, because tf-idf vectors are non-negative.

There's no particularly deep connection between tf-idf and the cosine similarity/vector space model; tf-idf just works quite well with document-term matrices. It has uses outside of that domain, though, and in principle you could substitute another transformation in a VSM.

CHAPTER 8. PROJECT PARTS

8.1 Computing Word Frequencies in Web Pages using Binary Search Trees

The first step to building a search engine is to be able to analyze web pages based on their content. This will eventually allow us to rate how relevant a particular page is for a given user request. The content of a page is obviously correlated with the words it contains. For this assignment you will use a binary search tree to help you calculate and store the frequencies of words found in a given web page. Then you will print out the most frequent words found.

Begin by experimenting with the Scanner class and its associated test class provided below. The Scanner class is written to scan either a file or a string. For this assignment you will use the Scanner class for scanning a file. The constructor for the Scanner class requires a filename. It then opens the given file and will return the next token that it finds when the method getNextToken() is called. For our purposes a token will be a single non-alphabetic and non-numeric character, such as punctuation, or a string of contiguous alphabetic and numeric characters up until white space is encountered or the end of file.

Once you understand how the Scanner class works, modify the TestScanner class to read in a filename from the command line, and test that it works (see the section on Command Line Arguments below).

Next, you need to implement the insertElement, findMinimumNode, and findMaximumNode methods of the LinkedBinarySearchTree class. Make sure that your insertElement does not allow duplicate elements to be inserted in the tree. Once you have tested these changes to the LinkedBinaryTree class, you can start implementing the main part of this assignment.

8.1.1 Command line Arguments

Command line arguments are passed into a main method as an array of String:

```
public static void main(String args[]) {
```

To get the number of command line args, check value of args.length and to get a particular command line argument access the corresponding args array entry (arg[0] is the first command line argument as a String). To convert a command line argument from its String form to another type, use the valueOf methods of the Integer, Float, etc. classes. For example:

```
    # if this is the command line
#
% java MatrixMult 10 15  matrix_contents


    # this is how to convert the command line args 10 and 15 to int
#
int m =  (Integer.valueOf(args[0])).intValue();  // arg[0] is "10"
int n =  (Integer.valueOf(args[1])).intValue();  // arg{1] is "15"
String matrixFile = args[2];
```

8.1.2 Word Frequency Tree Classes

You should create a *WordFrequencyTree* class that implements the BinarySearchTree interface. This class will have a LinkedBinaryTree data member (it may need additional data members too), and its implementation of the BinarySearchTree interface methods just invoke the corresponding method on its LinkedBinaryTree data member. In addition it should have at least three constructors: a default constructor, a constructor that takes an input file argument and a constructor that takes an input file and an ignore file argument. The second two constructors will process the file(s) and create the initial word frequency tree as specified above.

8.1.3 Java Classes

These classes include the following:

- **Scanner class**
- **TestScanner class** and a simple test file testscannerfile
- **BTNode** interface
- **BinarySearchTree** interface
- **LinkedBTNode** class
- **LinkedBinarySearchTree** class: a partial implementation of the LinkedBinarySearchTree class, some of the methods are left for you to implement.
- **TraversalIterator** class: returned by the LinkedBinarySearchTree traversal methods

8.2 Processing User Queries to Find the Most Relevant Web Pages

A basic and very simple search engine which is used as the starting point for the project.

```
import java.io.*;
import java.util.Scanner;

class Index1 {

   WikiItem start;

   private class WikiItem {
      String str;
      WikiItem next;
```

```
    WikiItem(String s, WikiItem n) {
        str = s;
        next = n;
    }
}

public Index1(String filename) {
    String word;
    WikiItem current, tmp;
    try {
        Scanner input = new Scanner(new File(filename), "UTF-8");
        word = input.next();
        start = new WikiItem(word, null);
        current = start;
        while (input.hasNext()) {   // Read all words in input
            word = input.next();
            System.out.println(word);
            tmp = new WikiItem(word, null);
            current.next = tmp;
            current = tmp;
        }
        input.close();
    } catch (FileNotFoundException e) {
        System.out.println("Error reading file " + filename);
    }
}

public boolean search(String searchstr) {
    WikiItem current = start;
    while (current != null) {
        if (current.str.equals(searchstr)) {
            return true;
        }
        current = current.next;
        }
        return false;
    }

    public static void main(String[] args) {
        System.out.println("Preprocessing " + args[0]);
        Index1 i = new Index1(args[0]);
        Scanner console = new Scanner(System.in);
        for (;;) {
            System.out.println("Input search string or type exit to stop");
            String searchstr = console.nextLine();
            if (searchstr.equals("exit")) {
                break;
            }
            if (i.search(searchstr)) {
                System.out.println(searchstr + " exists");
            } else {
                System.out.println(searchstr + " does not exist");
            }
        }
        console.close();
    }
}
```

8.3 Adding a GUI front-end to the search engine and adding a cache of search query results

Part 1 of the assignment is building a GUI front-end to your search engine that also performs the fetching and displaying URLs job of a web browser. Part 2 is adding caching of query results to your search engine, and speeding up the execution of queries by using the cached results of previous queries.

8.3.1 GUI

Build a GUI front-end to your search engine. It should have the following 6 GUI components:

1. a query text box (where the user can type in a query)
2. a search button
3. a URL text box (where the user can type in a url)
4. a fetch URL button
5. a display area for search results
6. a display area for fetched urls

Your WebBrowser's main method will start up by taking a url_list and ignore_list command line options, create a ProcessQueries object, and then create the GUI front-end.

The WebBrowser GUI will work like the following:

- When the get URL button is selected, the url entered in the URL text box is evaluated, the url's webpage is fetched and its contents are displayed in the display box for fetched urls.
- When the search button is selected, the query entered in the query box is evaluated and the search results box displays the ordered list of matching webpages
- . In addition, the webpage of the best match is fetched and displayed in the display box for fetched urls.

8.3.2 Caching

Below is a demonstration of how your more efficient query processing program should behave. You will save all previous query results in a data structure called the cache. When the user types in a query, you will first check the cache to see if you have processed a similar request in the past.

```
Enter next Query String or -1 to quit
######################################
artificial

  Results for your Query "artificial":
  --------------------------------------------
  Query: artificial
        Query Result NOT Found in cache                    <-------- print out whether or
not query is in cache
        0 words from query found in cache          <-------- if not, print out how many
words of the query are in the cache

  Matching URLs:
  www.cs.swarthmore.edu/~meeden   priority: -8

######################################
Enter next Query String or -1 to quit
######################################
artificial intelligence

  Results for your Query "artificial intelligence":
  --------------------------------------------
  Query: artificial intelligence
        Query Result NOT Found in cache
        1 words from query found in cache

  Matching URLs:
  www.cs.swarthmore.edu/~meeden   priority: -17

######################################
Enter next Query String or -1 to quit
######################################
intelligence
```

```
Results for your Query "intelligence":
--------------------------------------------
Query: intelligence
    Query Result Found in cache

Matching URLs:
www.cs.swarthmore.edu/~meeden   priority: -9
```

Notice that the first time the user asks about "artificial" we cannot find anything in the cache. But the next time the user asks about "artificial intelligence" we can grab the stored information about "artificial" and perform a search of WordFrequencyTrees for just the "intellignece" part. When the query "intelligence" is entered, its result simply can be obtained from the stored informatation in the cache. Also note that we want to be able to recognize that a query is the same regardless of the word order, so "artificial intelligence" is the same as "intelligence artificial".

8.4 **Adding a hyperlink graph to the Web browser**

For this program you will add a graph of URL links to your Web browser. You will create the graph by parsing a starting url's file and finding href links of other local webpages, and parsing them, and so on. The graph will contain a vertex for each url, and an edge (u, v) if there is a link from url u to url v. The graph can be added as a data member to your ProcessQueries or WebBrowser class.

You will use the graph in two ways:

1. You will create a "Graph Window" button and add it to your Web browser. When this button is selected it will bring up a new window with the following options:
    - A display window to print results of button actions
    - A "Reachable From" button and text window: when a url is entered in the text window, and the button is selected, it calls the reachableFrom method of your graph and prints out the results to the display window.

    - A "Shortest Path" button and text window: when a url is entered in the text window, and the button is selected, it calls the shorestPath method of your graph and prints out the results of the shortest path from the url to all other urls reachable from it. You are required to implement the shortestPath method of the Graph class.
    - A "Print Graph" button: when selected, the graph is printed to the display window.

   We implemented the Graph Window GUI for you, you just need to add a button to your WebBrowser to pop-up the Graph Window.

2. You will add linkage information to compute a good search result: If a page that matches a query string is linked-to by many other pages its priority should be increased some amount based on its linked-to degree. You should use this new criterion combined with word frequency count information to order query results.

CHAPTER 9. CONCLUSION AND FUTURE WORK

In this project, we have proposed a framework for personalized web search using User Profile and Domain Knowledge. Based on the User Profile and the Domain Knowledge, the system keeps on updating the user profile and thus builds an enhanced user profile. This Enhanced user profile is then used for suggesting relevant web pages to the user. The proposed framework has been implemented by performing some experiments. These experiments shows that the performance of the system using enhanced user profile is better than those which are obtained through the simple user profile. Our work is significant as it improves the overall search efficiency, catering to the personal interest of the user's. Thus, it may be a small step in the field of personalized web search. In future this framework may be applied for re-ranking the web pages retrieved by search engines on the basis of user priorities.

REFERENCES

[I] M Speretta and S Gauch, "Personalized Search Based on User Search Histories", Proceeding Of International Conference on Web Intelligence, pp. 622-628,2005.

[2] F Liu, C Yu and W Meng, "Personalized Web Search for Improving Retrieval Effectiveness", IEEE Transactions On Knowledge And Data Engineering, pp. 28-40, Volume 16,2004.

[3] C Liang, "User Profile for Personalized Web Search", International Conference on Fuzzy Systems And Knowledge Discovery, pp. 1847-1850,2011 .

[4] X Pan, Z Wang and X Gu, "Context-Based Adaptive Personalized Web Search for Improving Information Retrieval Effectiveness", International Conference on Wireless Communications, Networking and Mobile Computing, pp. 5427 - 5430, 2007.

[5] K.W.T. Leung, D.L. Lee and Wang-Chien Lee, "Personalized Web search with location preferences", IEEE 26th International Conference on Data Engineering, pp. 70 I - 712, 2010 .

[6] O. Shafiq, R. Alhajj and 1. G. Rokne, "Community Aware Personalized Web search", International Conference on Advances in Social Networks Analysis and Mining, pp. 3351 - 355,2010