Putting Information Retrieval Theory into Practice – A Web Search Engine Project for an Undergraduate Computer Science Elective Course

Xiannong Meng
Computer Science Department
Bucknell University
Lewisburg, PA 17837

Abstract

This paper describes a semester project for an undergraduate computer science senior elective course, CSCI 379 *Computer Science Topics – Information Retrieval and Web Search*, taught at Bucknell University in the fall semester of 2002. In this course, students working in groups developed a Web search engine using information retrieval theory. The project includes implementing a basic Web server which acts as the dispatcher, an indexing component which builds the inverted indexing system for search, a ranking component which ranks the documents based on term frequency (tf) and inverted document frequency (idf), a retrieval component which takes the user query and retrieves the documents based on the ranking, and an crawling component which collects documents from the Web. The project is very practical in that students have to build a complete system, yet it involves many theoretical aspects of the information retrieval, algebra, and probability. It is an ideal project for a senior level course which requires a combination of the knowledge students have learned in their college years.

1 Introduction

The wide spread use of the Web brought increased interests to college undergraduate courses such as information retrieval and computer networks. Students in these courses learn the basic concepts of the Web, the information retrieval theory (IR) and the Internet. They hope to gain an understanding how the Internet and its applications work and practice basic programming skills needed to program the Internet related applications. Web search engines provide ideal case studies for such courses. Implementation of a successful Web search engine requires a combination of information retrieval theory and solid network programming skills. This paper describes a semester project for an undergraduate CS senior elective course, CSCI 379 *Computer Science Topics – Information Retrieval and Web Search*. In this course, students working in groups developed a functional Web search engine. The project includes implementing a basic Web server which acts as the dispatcher, an indexing component which builds the inverted indexing system for search, a ranking component which ranks the documents based on term frequency (tf) and inverted document frequency (idf), a retrieval component which takes the user query and retrieves the documents based on the ranking, and an crawling component which collects documents from the Web. The project requires many theoretical aspects of the information retrieval, algebra, and probability. It is an ideal project for seniors since it provided students with the opportunities to integrate the knowledge gained in their college years. The rest of the paper is organized as follows. We will provide an overview of the course in Section 2. The project itself is described in detail in Section 3. Interested readers may use it as a blueprint for their projects. Section 4 discusses some issues and lessons learned in the project. Section 5 contains some related information and a brief review of other similar courses. Section 6 is a summary.

2 Overview Of The Course

This is a computer science elective course open to seniors who completed a junior level data structure and algorithm course. The *Modern Information Retrieval* [1] by Baeza-Yates and Ribeiro-Neto was used as the textbook. Other materials available on the Web were used as supplements. A resource list is provided at the course Web site[4]. The course contained 42 one-hour lecture periods. We presented most of the main topics in a typical information retrieval theory course. First we gave an overview of the information retrieval theory. Then we introduced one of the most exciting applications of the information retrieval theory, Web search engines. Introducing an interesting application earlier motivates students to learn better the course materials. Students were able to see the connection between the general IR theory we discussed in the lectures and their actual applications. The basic vector model was used to model the documents. We discussed indexing, retrieval evaluations, relevance feedback, Web crawling, link structure analysis and general text properties. The Appendix lists the schedule of the course which has a complete list of the topics. The key characteristic is that the lecture contents were closely matched with what were required for the particular phase of the course project. Towards the end of the course after they finish the programming project, students write a survey paper on various subjects of IR and Web search, and present the findings to the class.

3 Web Search Engine – The Course Project

The goal of the project is to produce a limited scale, but functional search engine applying IR theory. The search engine should be able to provide a list of relevant documents when a query is given. The project is in a limited scale in the sense that it is required to collect a limited number of documents (e.g. in the order of a few hundreds to a few thousands). The limitation is due to the memory constraint on the desk-top computers that students used and the disk quota that was allocated to students.

It is a multi-phase, team project. The following subsections give an overview of the project and a brief description of each of the four phases of the project.

3.1 An Overview

A search engine consists of a collection of software components that work together to accomplish the task of collecting, analyzing documents over the Internet and giving the user a list of relevant URLs when a query is issued to the search engine.

A typical search engine has the following components.

- A **user interface** takes the user query, passes it to the retrieval component, and displays the results.
- A **crawler** visits the Web and collects information about all the documents it encounters over the Web.
- An **indexer** indexes each of the pages collected by the crawler and establishes links between keywords and the documents that contain them.
- A **ranker/retriever** ranks the documents for a given query according to certain measures and retrieves the most relevant documents for the user.
- A **back-end engine** takes care of network and file operations.

Figure 1 indicates the relation among different components in a typical search engine.

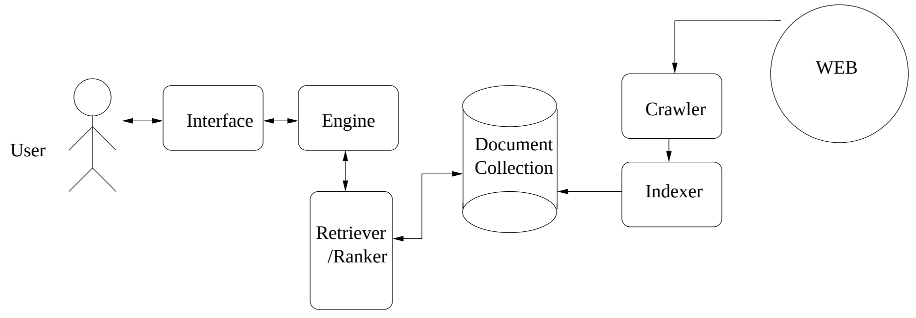

Figure 1: Components of a Typical Search Engine

These components can be divided into two major parts, somewhat independent of each other, as can be seen from Figure 1. Components on the left side of the *document collection* answer user's queries from the document collection. Components on the right side of the *document collection* collects and indexes documents from the Web.

All students chose Java as the implementation language, though other programming languages would be fine too. The completed projects contained somewhere between 2,000 and 3,500 lines of Java code.

3.2 Phase One – Building a Web Server

The first phase of the project is to build a simple Web server. This Web server will work as the basis of the search engine and acts between the document collection and the user through a regular Web browser. The Web server runs as an independent server program. The user accesses the server through a browser (e.g Netscape or InternetExplorer). The server retrieves the Web page specified by the URL.

Since network application programming was not a pre-requisite of the course, students were given a simple Web server written in Java (about 200 lines of Java code). They are asked to understand what the program is doing and understand the HTTP protocol that governs the interaction between the Web server and a browser. The given server code accesses a fixed file. The students were asked to revise the program so that the server will access any file specified by the URL and be able to send back image files if an URL contains one.

3.3 Phase Two – Index the Documents

In this phase of the project students build the indexing component of the search engine. The indexer takes a sequence of file names as input. When integrated with other components to make a complete search engine, these files are sent from the crawler(s). For each of the files specified by the file name, the indexer processes the file in the following steps.

Lexical analysis (tokenizing) divides the input stream into tokens and extracts words as well as URLs from the input stream. Students were advised to convert characters into their lower case and squeeze multiple spaces between words into a single space for easy processing. The exception is that the cases and spacing in URLs are preserved.

Stopwords removal removes stopwords from the input stream. Most students used some stopword list from the Internet as the base and added a few of their own.

Stemming reduces a input word into its stem. Students were encouraged to use existing programs on the Internet to do this since an implementation of a typical stemming algorithm such as Porter's [1] itself can take a while. We decided that should not be a focus of the project, thus using an existing algorithm or program is a reasonable choice.

Selecting indexing terms decides a set of words as the indexing terms. For simplicity students were advised to use all words from the document collection as index terms, after removing all stopwords and stemming the remaining words.

Updating the indexing system builds the inverted index system that can retrieve documents based on the index word. Figure 2 shows how a inverted indexing system may look like. The basic index system has a list of index terms across the whole document set. Each index has a list of posting nodes, each of which contains the information about that term and that document. In the simplest form a pair as document ID and term frequency is kept in the node. One may keep other information as well such as the location of the term within the document and the importance of the term as perceived within the text (whether or not a heading, bold faced ...). The index terms should be sorted alphabetically for easy search.

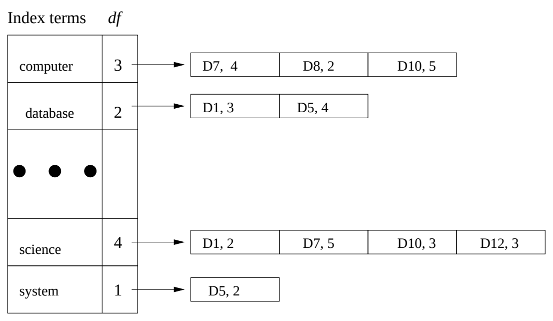

Figure 2: An Example of Inverted Indexing System

Students were asked to test their index system on a given set of Web pages that are accessible through our local file systems. The test set has a few HTML thousand files. Students may choose to test on other sets of files.

3.4 Phase Three – Ranking and Retrieving Documents

The end result of the indexing phase is a term list and a collection of posting lists, one for each term (See Figure 2 for the relation between the term list and the posting lists). Phase Three of the project adds two more components to the program. One is the term weight module. We used the well-known *tf-idf* (term frequency-inverted document frequency) in the vector model[1, 7] as the term weight. The other is the retrieval and ranking module, which should return a ranked list of documents for a given query.

Term frequencies *tf* collected in the indexing phase are used to build the term weight. A term frequency $t_{i,j}$ is the frequency of term i appeared in document j. Since we are using the *tf-idf* as term weight, students need to compute the *document frequency* now. Document frequency of a term is defined as the number of documents that contains the term, which is simply the length of the posting list. With *df* and *tf*, we can compute *idf*, inverted document frequency and the product of *tf* and *idf*. Thus the weight for term i in document j is computed as

$$w_{i,j} = tf_{i,j} * idf_i = tf_{i,j} * log(\frac{doc\ count}{df_j})$$

This value $w_{i,j}$ typically is stored in the *DocNode* on the posting list.

Once the term weight for each term is computed, students can complete the ranking and retrieving module. The basic idea is that each query is treated as one document in the vector model. When a query is given, a sequence of steps are taken. First all the documents containing the keywords in the query are retrieved. Then the similarity between the query and all these documents are computed. The similarity measure used here is the cosine similarity[1, 7] represented as follows.

$$similarity(D_i, D_j) \ = \ \frac{\sum_k w_{k,i} * w_{k,j}}{|D_i| * |D_j|}$$

The documents are sorted according to the similarity scores. This list is sent back to the user as the search result.

3.5 Phase Four – Crawling the Web

At this point students had built a basic information retrieval system that could answer queries and retrieve relevant documents, except that these documents came from local file systems instead of the Web. The next phase of the project is to actually collect documents from the Internet and feed them to the partial system that had been built.

The crawler starts with a given URL. It will retrieve the page specified by this starting URL. Parse the page, extract some more URLs from the page. Then visit the pages following these new URLs. This process will continue until either the time allowed has expired, or the number of retrieved pages has reached the limit, or there is no new page to visit.

The retrieved Web pages are passed to the Indexer you built in the second phase of the project for processing. There an inverted index will be built for all the Web pages retrieved, ready for an end user to search.

3.5.1 General Algorithm

Traverse the Web is very similar to traverse a general graph. Each Web page can be considered as a node in a graph. Each hyper-link can be considered as a link in a graph. From this point of view, crawling Web is not too much different from the graph traverse algorithm taught in a typical data structure class.

3.5.2 Some Issues To Be Considered

Because of the vast size of the Web, there are some technical and engineering issues we have to consider for a successful, less-intrusive crawler. We list here some of the issues to consider in crawling the Web.

- Obey the robot protocol. See http://www.robotstxt.org/wc/exclusion.html for specific details. The basic idea when crawling the Web is that first to check the server site (typically the root page) to see if the server administrator has put the `robots.txt` in place. If it is there, check the contents to see what directories are excluded for visiting. When visiting each page, also check the meta tag to see if the page is excluded for visiting.
- Self identification. When visiting a Web site the crawler should report to the server the name of the user agent, the host where the program is running and a valid email address of the user who initiated crawling.

- Self identification is a part of *good robot behavior*. Another important *good behavior* is not to visit the same site with a lot of rapid requests. Rather wait a few minutes before next visit.

- If there is a need to save the downloaded pages to disk files for further processing, make sure to use `synchronized` thread in Java to avoid inconsistence of the file status.

- Complete partial URLs. Many Web pages contain partial URLs. That is, the URLs are relative to the current path. For example, one may encounter URLs within a page in the form of `../../home/page.html` or `mypage.html`. The crawler needs to expand these partial URLs to a full URL so that the crawler can access them later.

- The crawler has to keep track of the pages that have been visited in order to avoid infinite loop. This turns out to be a challenging issue because it is not practical to keep a complete list of visited pages. We can only keep certain number of pages in the *visited* list.

- Test the crawler in some small site within one's own Web site first (for example ones own university or college). Do not crawl off-campus sites until the crawler is fairly robust.

4 Issues and Lessons Learned

The course project was a huge success. Students' responses were overwhelmingly positive. They were extremely happy to see actual, relevant URLs returned by the search engine they developed for the given queries. Such comments are abundant in the team reports and course evaluations. A few issues call for special attentions.

- Giving students a project that is "real" in the sense that it works in production environment demystifies many of the concepts and technical details. Students would otherwise not be able to appreciate many important issues, especially many of the engineering issues, with which such a software project would have deal. It empowers the students. They are more confident in their capability to tackle complicated issues. At the same time, they appreciate the delicate details a piece of successful software would have to deal with.

- The project was manageable due to its well defined phases. Each phase concentrated on a section of the problem. When combined as a working search engine, there was not a sudden jump in complexity.

- The project was designed for team work. Students learned pros and cons working in a team environment. Most liked the team work. Many commented that it was very productive working as a team.

- Students were assigned to write a research survey paper in an area that deals with issues related search engines. Students chose a subject area interesting to them the most. After the experiences of implementing a simple search engine, students had a better and deeper understanding of the issues when they read research papers.

- One of the weaknesses of this project is that we didn't have a very good design phase. Some students pointed out that they'd wish there were more time for design. It would be preferable if more design were incorporated from software engineering point of view. The difficulty was the constraint on time. Students completed their programming project in the 11th week of the semester, leaving about three weeks for a research paper project. There was not much time for a detailed design. We need to investigate further how we can tackle the issue of design.

- Because students ran their project from lab computers or from their own personal computers, they encountered the problem of limited resources. The program would run out of memory, the disk quota would be exceeded. For practical purpose we had to tell students to limit the number of pages to be indexed to a few thousands.

5 Related Information

Traditional information retrieval predates the Web and the wide spread use of the Internet [7]. The main theme in the theory of information retrieval is to develop algorithms and data structures to increase the *recall* and *precision* of relevant documents for a given query [1, 7]. *Recall* measures how many relevant documents are retrieved out of all the relevant documents, and *precision* measures how many relevant documents are retrieved out of the total returned documents for a given query. To test an algorithm or a data structure, one would run the program against a given set of documents with known query results which have been generated and examined by human experts. Researchers exchange ideas and results in conferences such as TREC (Text REtrieval Conference) [8]. An information retrieval course can concentrate on these subjects without ever using the Web and the Internet. Examples of this type of course concentrating on the information retrieval side include CMPSCI 646 at the University of Massachusetts [2], CS529 at the Illinois Institute of Technology[3] among others. On the other hand, retrieving relevant information through the Web is a direct application of the information retrieval theory. Many recent information retrieval courses incorporate the subject of Web search into the teaching. Examples include DS 575 at the DePaul University [5], CS 378 at the University of Texas – Austin [6], CS 466 at John Hopkins University [9] among others. The programming projects in these courses deal with various parts of the Web search including search engines, crawling, retrieval and user interface. Most of these projects use the technology of CGI, making it rely on existing Web servers. What made our project different was that we emphasized the whole picture of Web search. Our project included every components in a Web search retrieval system, the Web server, the crawler, indexing, ranking and retrieving components. The only software requirement for our project is a high-level programming language that supports network programming. All student groups chose Java as their implementation language, other languages such as C/C++ will do as well.

6 Summary

We described a semester project for the CSCI 379 *Computer Science Topics – Information Retrieval and Web Search course* at Bucknell University. Students implemented a complete Web search engine allowing the users to actually issue queries and get relevant URLs back. The project included all components of a typical search engine, indexing, crawling, ranking, retrieving and providing a Web-based user interface. Implementing the project required a balanced combination of information retrieval theory and solid network programming. The responses from students were very positive. We believe this is a very useful, challenging team project that can benefit students even after their graduation.

References

[1] Richardo Baeza-Yates and Berthier Ribeiro-Neto, *Modern Information Retrieval*, Addison-Wesley, 1999.

[2] <http://ciir.cs.umass.edu/cmpsci646/>, course Web site for CMPSCI 646 *Information Retrieval* at the University of Massachusetts, Fall 2002.

[3] <http://ir.iit.edu/˜dagr/cs529/>, course Web site for CS 529 *Information Retrieval* at the Illinois Institute of Technology, Fall 2001.

[4] <http://www.eg.bucknell.edu/˜cs379/IR-Web/>, course Web site for CSCI 379 *Computer Science Electives – Information Retrieval and Web Search*, Fall 2002.

[5] <http://maya.cs.depaul.edu/˜classes/ds575/syllabus.html>, course Web site for DS 575 *Intelligent Information Retrieval* at the DePaul University, Winter 2003.

[6] <http://www.cs.utexas.edu/users/mooney/ir-course/>, course Web site for CS 378 *Intelligent Information Retrieval and Web Search* at the University of Texas – Austin, Fall 2002.

[7] Gerard Salton, *Automatic Text Processing*, Addison-Wesley, 1989.

[8] Annual Text REtrieval Conference, <http://trec.nist.gov/>, National Institute of Standards and Technology.

[9] <http://www.cs.jhu.edu/˜yarowsky/cs466.html>, course Web site for CS 466 *Information Retrieval and Web Agents* at John Hopkins University, Spring 2002.

Biographical Information

XIANNONG MENG is an Associate Professor in the Department of Computer Science at Bucknell University in Lewisburg, Pennsylvania, U.S.A. His research interests include distributed computing, data mining, intelligent Web search, operating systems and computer networks. He received his Ph.D. in computer science from Worcester Polytechnic Institute in Worcester, Massachusetts, U.S.A.

Appendix – Course Schedule for CSCI 379 at Bucknell

CSCI 379.01 Fall 2002 Schedule

The following is a tentative schedule of the material that will be covered. References to chapters from the textbook are given for each topic. Materials from other reference sources will also be used in lectures.

Week 1:	8/28	Introduction and overview	Baeza-Yates Chpt. 1
Week 2:	9/2	Web search	Baeza-Yates Chpt. 13.1-13.4
Week 3:	9/9	Basic IR models	Baeza-Yates Chpt. 2.1.-2.5
Week 4:	9/16	Text processing and indexing	Baeza-Yates Chpt. 7.1-7.2, 8.1-8.2
Week 5:	9/23	Retrieval evaluation	Baeza-Yates Chpt. 3
Week 6:	9/30	Relevance feedback	Baeza-Yates Chpt. 5.2, 10.7, papers
Week 7:	10/7	**Exam One**	chapters covered since beginning
	10/9	Query languages	Baeza-Yates Chpt. 4
Week 8:	10/14	Query operations	Baeza-Yates Chpt. 5
	10/19	**Fall break**	
Week 9:	10/23	Crawling and meta-crawling	Baeza-Yates Chpt. 13, papers
Week 10:	10/28	Link analysis	Baeza-Yates Chpt. 13, papers
Week 11:	11/4	Text properties	Baeza-Yates Chpt. 6.1-6.4,6.6
Week 12:	11/11	**Exam two**	chapters covered since Exam One
	11/13	Text operations	Baeza-Yates Chpt. 7.3-7.6
	11/15	Guest lecture	Current research
Week 13:	11/18	User interface	Baeza-Yates Chpt. 10
Week 14:	11/25	String match	Baeza-Yates Chpt. 8.4-8.6
	11/27	**Thanksgiving break**	
Week 15:	12/2	Student team presentation	
	12/9	Float	
Final		Date determined by university	Comprehensive