Teaching Web Information Retrieval to Undergraduates

Frank McCown
Harding University
Computer Science Dept
Searcy, Arkansas, USA 72149
fmccown@harding.edu

ABSTRACT

Topics in the field of Web Information Retrieval (IR) and web search are slowly being introduced at the undergraduate level. In this paper, we show how the curriculum for a new undergraduate course on web search engines was developed, and we share our experiences in having students develop their own search engine components from scratch or modify and extend a popular open source search engine. We hope our experiences will be helpful to other Computer Science departments that are looking to develop an undergraduate Web IR course.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Curriculum

General Terms

Algorithms, Experimentation

Keywords

open source, search engines, teaching, Web IR

1. INTRODUCTION

The utility and ubiquity of web search is making Web Information Retrieval (IR) an increasingly popular research topic. The incredible success of Google is a striking example of how important it is for academia and industry to foster innovation in the field of Web IR. Other tech companies like Yahoo and Microsoft continue to make significant investments in web search, and Tim Berners-Lee et al. have begun promoting the study of Web Science [2] in which Web IR is a strong component.

While Web IR is often taught at the graduate level, it is increasingly being introduced at the undergraduate level as an elective (see [10] for a complete discussion of IR curriculum issues in CS). Web IR is often an appealing subject to computer science (CS) students who use search engines like Google and Bing on a daily basis and are curious as to how they function. Using data structures learned in CS 2, an undergraduate student can build a complete web search engine in a semester's time. This can provide extra motivation for students when they are able to apply theoretical concepts to a practical application.

At Harding University, we began preparing in the fall of 2007 to offer an elective undergraduate course called *Search Engine Development*. We wanted to develop a course for undergraduate students in CS with a minimal amount of prerequisites so as to maximize the course's potential audience. We made CS 1/2 and our core Internet Development 1 course the only prerequisites. This allowed sophomores, juniors, and seniors to enroll.

When developing the content for this course, we looked for similar offerings at other universities to determine what subjects should be included and what types of programming assignments should be given. From our survey, we created a curriculum which covered a majority of the topics taught at the undergraduate and graduate levels at many universities, and we experimented with two different approaches to the class programming projects. In 2008, we required students to build a complete search engine from scratch (as suggested in [6] and [8]). In 2009, we required students to modify and extend the open source search engine Nutch [21].

In this paper, we present the curriculum for an undergraduate course focusing on web search which we developed from our survey of thirteen graduate and undergraduate Web IR courses. We discuss text books and reading material for the course, and we share our projects and our experiences having students build a search engine from the ground-up and modifying a complex open source search engine.

2. CURRICULUM

Although the instructor who would be teaching the search engine course had performed extensive research in using search engines for preserving the Web (see, e.g., [19]), he had never taken a Web IR course and had only studied IR independently. Having no base materials from which to develop a Web IR course, it was decided that a curriculum survey should be performed of existing course offerings in Web IR.

The courses listed in Table 1 (sorted by course title) were found using word-of-mouth, Google, and referring links from class web pages. This list is not meant to be an exhaustive list of all Web IR courses, but it is representative of the types of courses that focus mainly on Web IR issues. Sev-

Table 1: A selection of Web IR courses offered to graduates and undergraduates.

Title	Instructor	University	Audience
Information Retrieval	Susan Gauch	University of Arkansas	Grad
Information Retrieval and Search Engines	C. Lee Giles	Penn State	Undergrad
Information Retrieval and Web Search	Raymond J. Mooney	University of Texas at Austin	Undergrad
Information Retrieval and Web Search	Rada Mihalcea	University of North Texas	Grad
Search Algorithms	Christian Schindelhauer	Univ of Paderborn (Germany)	Unknown
Search Engine Technologies	Nigel Ward	Univ of Texas at El Paso	Both
Search Engine Technology	Dragomir R. Radev	Columbia University	Grad
Search Engines	Bruce Croft	Univ of Massachusetts Amherst	Undergrad
Search Engines: Tech., Society, & Business	Marti Hearst	Univ of California, Berkley	Both
Web Information Retrieval	Xiannong Meng	Bucknell University	Undergrad
Web Search and Mining	C. Manning and P. Raghavan	Stanford University	Grad
Web Search Engines	Ernest Davis	New York University	Grad
WWW Search Engines	Brian D. Davison	Lehigh University	Both

eral courses that came up during the search only focused on IR with little Web application or were non-technical in nature; these were excluded from the list. We noted if the course's audience was primarily graduate, undergraduate, or both. This helped us pick topics more readily accessible to undergraduate students.

Many of the courses listed in Table 1 contained extensive websites with syllabi and project guidelines which greatly helped us in preparing our curriculum. In a few cases, we contacted the instructors of the course to inquire more about the course's content.

We pulled together a core curriculum for our search engine course based on the curriculum surveyed and based on the instructor's research interests. The course had several high-level learning objectives:

1. Students should have a basic understanding of how the Web is organized and its fundamental properties.
2. Students should understand how search engines collect Web content, index it, and present the most relevant results for a given query. They should also understand the technical, legal, financial, and social obstacles that make web search a very difficult problem.
3. Students should understand how to design and construct software that implements several significant Web IR concepts.
4. Students should be aware of new research directions in the field of Web IR and the new features being developed by today's commercial search engines.

To meet these learning objectives, the course covered (in order) the following topics:

- Web IR background
  1. Web IR vs. traditional IR
  2. Web search history
- Web Characterization
  1. Structure of the Web (Bow-tie, scale-free and small world networks, power-law distributions)
  2. Deep web and surface web
  3. Linkrot and web archiving
- Search engine characteristics
  1. Metasearch, vertical, and universal search engines
  2. Organic results, paid inclusion and placement
  3. Types of web search queries
  4. Commercial search engine query characteristics
  5. Evaluating search engine effectiveness (relevancy, precision, and recall)
- Web crawling
  1. Crawler architecture
  2. Politeness policies (robots.txt)
  3. Selection policies (URL characteristics, spider traps, repository freshness)
  4. URL normalization
  5. Sitemap protocol
- Indexing
  1. Boolean retrieval model
  2. Vector space model
  3. Inverted index and implementation issues
  4. Stemming and stop lists
  5. Synonyms, homographs, and misspellings
  6. Term frequency and inverted document frequency
- Ranking
  1. Link analysis algorithms (HITS and PageRank)
  2. Rank fusion algorithms (Borda and Condorcet count)
  3. Eye tracking studies
- Other
  1. Web spam and adversarial IR
  2. Search engine optimization (SEO)
  3. Detecting near-duplicate pages
  4. Personalized search

There were a number of other topics that we would like to introduce into our course, time permitting, in later offerings. These include: text mining (document classification, clustering, etc.), search interfaces, Semantic Web, latent semantic indexing, collaborative filtering, query log analysis, distributed computing issues (Google File System, Hadoop), web agents, and machine learning.

One of our CS department alumni who is now a developer for a major search engine company looked over our curriculum and had few suggestions for improving it. This curriculum is representative of other Web IR courses described in the literature [6, 20].

3. TEXTBOOKS

Many of the courses listed in Table 1 required one or more of the following text books (ranked by popularity):

1. *Introduction to Information Retrieval* by Manning, Raghavan, and Schütze (2008) [17]

2. *Modern Information Retrieval* by Baeza-Yates and Ribeiro-Neto (1999) [1]
3. *Mining the Web: Discovering Knowledge from Hypertext Data* by Chakrabarti (2003) [7]

An early edition of the text by Manning et al. [17] was made freely available on the Web, and that could account for some of its popularity. A number of other texts were also listed as recommended texts but not primary texts (e.g., [14, 12, 24]). Most instructors supplemented class readings or used exclusively the research papers that originated many Web IR topics (e.g., shingling [5], PageRank [22], HITS [15], and the bow-tie structure of the Web [4]). As Mizzaro [20] points out, these papers are usually easy for students to read and understand.

We were surprised that none of these courses used the text *An Introduction to Search Engines and Web Navigation* by Levene (2005) [16]. This text provides a good overview of many topics mentioned in the curriculum list, contains complete references to the many research papers cited in each chapter, and uses language appropriate to an undergraduate audience. Levene's book is currently out of print which could account for its lack of adoption.

We used Levene's book as our primary text (students had no problems finding used copies online). The instructor would occasionally require other readings that were found online about a given topic (Wikipedia provided a good starting point for many topics). We pointed students to many of the classic research papers mentioned previously but did not make them required reading.

A book just released in 2009 by Croft, Metzler, and Strohman entitled *Search Engines: Information Retrieval in Practice* [9] was written specifically to an undergraduate CS audience. It presents pseudocode for implementing a number of Web IR concepts and contains a number of exercises at the end of each chapter that use Galago [11], an open source search engine written by the book authors in Java. We may use this text in a future offering of our Web IR course.

4. HOMEWORK AND EXAMS

A number of homework assignments were given throughout the semester that would reinforce understanding of the day's subject material. Many of these exercises were assigned in the sampled courses or appeared in various forms in the books discussed in the previous section. Two exams and a final comprehensive exam assessed the students' understanding of the material; they used many questions similar in nature to the homework assignments. A brief summary of several homework assignments follows:

1. Experiment with a commercial search engine, use various boolean queries and special operators, and compare the differences in how items are ranked.
2. Calculate the precision and recall for several queries. Calculate the P@10 for queries on Google, Live Search, and Yahoo.
3. Create a term-document incidence matrix and inverted index for the given document corpus, and determine which documents are returned by the given boolean queries.
4. Calculate the TF-IDF scores for the documents returned by Google for the given queries.
5. Calculate the PageRank and hub and authority scores for the given Web graph after one iteration.
6. Calculate the cosine similarity between the two given documents and the Jaccard coefficient using shingles of length n.

5. CLASS PROJECTS

The programming projects used by the sampled courses in Table 1 vary greatly. Some courses require the students to build a fully-functional search engine as suggested in [6, 8], and other courses use independent projects that implement a particular concept. Sometimes the students are to write all their code from the ground-up, sometimes instructors provide some code that must be modified or enhanced, and other instructors encourage students to find freely available code on the Web to implement their projects. Most instructors assigned projects to be completed in small groups using popular programming languages like Perl and Java.

For our search engine course, we devised a number of small individual projects and some larger ones to be completed individually or in teams of two. Although we teach our CS 1/2 courses in C++, we required the projects to be coded in Java, thus giving students exposure to a new language. Some in-class time was allocated to teach Java and servlet basics, but the students were mainly left to learn the nuances of the language on their own.

5.1 Using Search Engine APIs

We initially required our students to create a search engine using the freely available web search APIs by Yahoo. (Google and Microsoft have similar APIs [18].) The Yahoo BOSS API [27] is REST-based and allows simple HTTP requests to be made which return back an XML-formatted set of search results. This relatively small assignment gave students experience in using Java servlets to create a fully functional search engine.

Our students later modified their search engine to create a "universal" search engine, incorporating news and image results along with regular web page results. This assignment made our students more aware of current efforts to enhance the search experience.

5.2 Search Engine from the Ground-up

The first time we offered our search engine course, we required our students to write nearly all of the code to implement a crawler, indexer, and retriever with a web interface. Each of these large project built on the previous project which required the students to re-use code for each new project. The only code given to them by the instructor was a simple web crawler which had to be greatly enhanced. A brief description of each project follows:

1. **Crawler** – Enhance the given crawler to obey robots.txt, limit crawls to a given depth, normalize URLs, and store pages in a local repository. The crawler should replace previously crawled content that resides at the same normalized URL.
2. **Indexer** – Create a program which will create an inverted index on the crawled corpus using TF/IDF and a weighing criteria based on URL, title, and metatags. It uses smart parsing of HTML, stemming, and removal of stop words.
3. **Retriever** – Create a program that can return the first 10 results for a Boolean query executed on the inverted index. Create a web interface for the retriever

which shows a Google-style results page. Extra credit is available for implementing phrase search.

Each project was similar in difficulty and consisted of several hundred lines of code. When the final project was completed, our students had a fully functional search engine which could crawl, index, and search a sizeable web corpus. While all the students were successful in completing at least a somewhat functional search engine, some of the implementation details regarding efficiency became very pronounced as some search engines could retrieve a document significantly faster than other students' implementations.

5.3 Modify an Open Source Search Engine

The second time we offered our search engine course, we decided to approach the search engine projects from the opposite direction. Rather than have the students create a search engine from scratch, we had them take an already existing open source search engine and make some enhancements. We envisioned several goals for our students:

1. Gain experience working on "real" software that is currently being used by many individuals and companies.
2. Observe how many professional software developers write code and how well/poorly they document their code.
3. Gain experience learning to understand what someone else's code is doing.
4. Learn about open source development, including bug tracking, making enhancement requests, version control systems, and code submission.
5. Make a significant contribution to an open source project.

Several of the surveyed courses used open source software, but we were unaware of any course that required their students to make a significant contribution to the software. In a personal correspondence with Bruce Croft, his students have made some contributions to Galago in the past.

We examined a number of open source search engines and finally settled on Nutch, an Apache project built on top of Lucene, a popular IR system. Nutch has a very active developer base and is used by a number of organizations. It is written in Java which, as explained earlier, allowed our students to be exposed to a new language with which they were not yet comfortable. Nutch includes a configurable crawler which can handle multiple content types and an indexer and retriever (which uses Lucene). Nutch uses Java servlets to present a web interface that can return results formatted in an aesthetically pleasing format. Nutch also uses Hadoop [3] and can therefore run efficiently on a distributed system.

Our students were first required to give a presentation on how to become a Nutch developer. This was a team assignment which required a lot of independent research. Some required components of the presentation included detailed steps on how to submit a bug fix or new extension to Nutch (including how JIRA works), how the various main classes of Nutch work together, and how to run JUnit tests for the classes in Eclipse (a popular Java IDE). The students were also to join the user and developer mailing lists and start monitoring the messages distributed daily.

After coming up-to-speed on Nutch, a few small projects and one major project were assigned:

1. **Crawler** – Learn how to run Nutch in Eclipse, and modify Nutch so it will output to a file all the URLs it successfully crawled.
2. **NutchBean** – Learn how NutchBean (the class responsible for querying the crawled corpus) works. Add some command-line parameters so NutchBean can search a user-supplied crawl collection.
3. **Site Query** – Modify Nutch's "site:" query operator to return all results from the given website even if no query term is provided.
4. **Sitemap Support** – Satisfy the outstanding Nutch enhancement request to support the Sitemap Protocol [26], a URL-discovery protocol supported by all the major search engines.

The first three projects required a great amount of code inspection, but only 10-20 lines of code were needed to complete the project requirements. The final project, adding Sitemap support to Nutch, required several hundred lines of code including vigorous JUnit tests to ensure the code was working properly. The final project was released open source as the Java Sitemap Parser [13] on Sourceforge.net, but time did not permit integration with Nutch.

6. DISCUSSION

Both the 2008 and 2009 search engine course offerings covered the same core curriculum given in Section 2, but they both used very different programming projects as outlined in Sections 5.2 and 5.3. The exams given in both offerings covered the same topics and were void of any programming-related problems. The exam averages were nearly the same for both offerings (83% in 2008, 82% in 2009), so choice of projects did not seem to have any impact in learning the overall concepts. The class sizes were too small, however (eight in 2008, three in 2009), to show statistically significant findings.

The instructor asked the students' opinions of the assignments given in both offerings, and several "grass is greener" remarks were made by students in both classes. The students that built the search engine from the ground-up felt they really knew the inner details of how a search engine worked, but all the code they wrote would likely never be used again outside of class. The Nutch students appreciated working on a real-world project, but they were not as aware of the implementation particulars of building a crawler, indexer, or retriever.

The Nutch students often expressed being overwhelmed by the complexity of the Nutch project. The instructor humorously referred to this as the "Nutch burn" since Nutch was relatively new to him as well. There was a large learning curve to getting Nutch properly configured and running, much less understanding its underlying code base which often contained poor documentation. One entire class period (75 minutes) was spent entirely on showing the students how to run Nutch in Eclipse and resulted in the instructor creating a lengthy wiki article on the topic [25]. The students' lack of Java proficiency also was problematic; much of the code used advanced programming concepts which even seasoned programmers might be slow to understand completely.

Overall, the Nutch-based course accomplished all the goals listed in Section 5.3, but it fell short in instilling an interest in Nutch that extended beyond the classroom. In [23], students showed a desire to continue working with the open source projects they had used in class assignments after the course had ended. None of our students expressed any desire to ever use or contribute to Nutch again. Most of our students still felt overwhelmed with Nutch's complexity, even after completing the projects.

Several lessons were learned from teaching this course twice:

1. Too much class time was spent teaching Java instead of Web IR topics. Students would have been able to complete the assigned projects more successfully and with less effort if they had entered the course with more Java programming experience. Therefore, future offerings of this course will require students to complete the department's OOP course that introduces Java.
2. The projects using Yahoo's BOSS API were excellent starting points and provided a gentle introduction into programming Java servlets. Students were able to build useful search engines with little effort and could play with how the search results were presented.
3. From a learning perspective, neither of the project approaches (search engine from scratch or modifying an open source search engine) is clearly better than the other. While the Nutch group learned more about open source software development, the other group had a better understanding of the programming details involved in building a search engine from the ground-up. Students from both groups learned the course subject material equally as well and satisfied the course's learning objectives.
4. From a teaching perspective, the instructor found it significantly more difficult to create class projects based on open source software. Most of this difficulty can be attributed to the very steep learning curve of the open source project which required the instructor to invest a significant amount of time becoming familiar with the code base. Students needed some hand-holding to gain a similar understanding of the code. Now that the instructor is more familiar with Nutch, it would be easier to reuse it in a future offering of the course. However, time must be exerted to keep up with changes in the continuously shifting code base.

We believe that in an ideal course students would create their own search engine and later use and modify an open source engine. From our experience, this is probably more than an undergraduate could accomplish in one semester's time. However, if a less demanding open source project were used (e.g., Galago), maybe a small project could be completed at the end of the semester.

7. CONCLUSIONS

We have described the process by which we developed a curriculum for an undergraduate course on Web IR. The course has used ground-up projects to build a search engine from scratch and projects that involve modifying an open source search engine, and we have shared some of the benefits and problems with both of these approaches.

We are planning to offer the search engine course again in the spring of 2011. We will likely use another open source search engine like Galago in this offering. We anticipate this course will continue to grow in popularity as opportunities in the field of web search continue to expand.

8. REFERENCES

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1st edition, May 1999.

[2] T. Berners-Lee, W. Hall, J. Hendler, N. Shadbolt, and D. J. Weitzner. Creating a science of the web. *Science*, 313(5788):769–771, August 2006.

[3] D. Borthakur. The hadoop distributed file system: Architecture and design. 2007. `http://projects.apache.org/projects/hadoop.html`.

[4] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer Networks*, 33(1-6):309–320, 2000.

[5] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the Web. *Computer Networks & ISDN Systems*, 29(8-13):1157–1166, 1997.

[6] F. Cacheda, D. Fernández, and R. López. Experiences on a practical course of web information retrieval: Developing a search engine. In *Proceedings of the Second International Workshop on Teaching and Learning of Information Retrieval (TLIR 2008)*, 2008.

[7] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann, 1st edition, October 2002.

[8] M. Chau, Z. Huang, and H. Chen. Teaching key topics in computer science and information systems through a web search engine project. *AMC Journal of Educational Resources in Computing*, 3(3), 2003.

[9] B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Addison Wesley, 1 edition, February 2009.

[10] J. Fernández-Luna, J. Huete, A. MacFarlane, and E. Efthimiadis. Teaching and learning in information retrieval. *Information Retrieval*, 12(2):201–226, April 2009.

[11] Galago. `http://www.galagosearch.org/`.

[12] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2nd edition, 2005.

[13] Java Sitemap Parser. `http://sourceforge.net/projects/sitemap-parser/`.

[14] K. S. Jones and P. Willett, editors. *Readings in Information Retrieval*. Morgan Kaufmann, 1 edition, July 1997.

[15] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[16] M. Levene. *An Introduction to Search Engines and Web Navigation*. Addison Wesley Publishing Company, November 2005.

[17] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 1 edition, July 2008.

[18] F. McCown and M. L. Nelson. Agreeing to disagree: Search engines and their public interfaces. In *Proceedings of JCDL '07*, pages 309–318, June 2007.

[19] F. McCown, J. A. Smith, M. L. Nelson, and J. Bollen. Lazy preservation: Reconstructing websites by crawling the crawlers. In *Proceedings of WIDM '06*, pages 67–74, 2006.

[20] S. Mizzaro. Teaching of web information retrieval: Web first or IR first? In *Proceedings of the First International Workshop on Teaching and Learning of Information Retrieval (TLIR 2007)*, 2007.

[21] Nutch. `http://lucene.apache.org/nutch/`.

[22] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Technical report, 1998.

[23] M. Pedroni, T. Bay, M. Oriol, and A. Pedroni. Open source projects in programming courses. *SIGCSE Bulletin*, 39(1):454–458, 2007.

[24] C. J. V. Rijsbergen. *Information Retrieval*. Butterworth-Heineman, 2 edition, March 1979.

[25] Run Nutch in Eclipse. `http://wiki.apache.org/nutch/RunNutchInEclipse1.0`.

[26] Sitemap Protocol. `http://www.sitemaps.org/protocol.php`.

[27] Yahoo! Search BOSS. `http://developer.yahoo.com/search/boss/`.