Spring 2016 EECS 767 Final Report

**Information Retrieval System**

Xi Chen (ID:), Yang Tian (ID:), Yuanwei Wu (ID: 2858647)

Department of Electrical Engineering and Computer Science

The University of Kansas, Lawrence, 66045, KS

**Abstract**

In this final project, three versions of search engine have been implemented and tested using different queries. The niche crawler with multi-threaded spiders has been developed to crawl web page sources from different domain (i.e. KU, Wikipedia, NYTimes). The preprocessing of documents includes removing html tags, tokenizing, removing stop words and stemming. The queries are processed using stop words list and stemmer. Boolean model and Vector Space model have been constructed in the search engine 1. Search engine 2 has been built by adding Term proximity to search engine 1. Search engine 3 has been implemented by adding the relevance feedback user option to search engine 2. The user interface of the information retrieval system has been developed to demo the performance of the three versions of search engine.

**1. Introduction**

Information retrieval (IR) is a process of locating and returning the relevant material to a user’s query. In classical SQL queries of database, the structured data are exactly matched to the query based on the ranking defined in query. However, in IR system, a bunch of unstructured objects (i.e. text document, image, audio) that satisfy the information need are searched and ranked from within large collections [1, 2].

The computer-based IR systems were started in 1950s [4]. This traditional text retrieval system is the version of web search 1.0 [3]. Two important tasks for an IR system were developed at that time: index documents and retrieve them [4]. In 1952, M. Taube et. al proposed to index items using a list of keywords [4] which is still used today. The so-called Boolean model was used to search and retrieve documents. In the Boolean model, each query was a logical expression of terms, and the returned documents were those which exactly matched the query [1, 4]. In 1960s, Prof. Salton at Cornell University, who is one of the pioneer researchers in the field of IR, suggested to rank the similarity between a document and query vector using cosine coefficient [1, 4]. The idea of relevance feedback was also introduced at this time. This process iteratively re-ranks the documents based on the feedback from users. One of the examples of relevance feedback is the “related articles” link on Google Scholar [4]. In 1970s, the theories of term frequency (tf), inverse document frequency (idf) and the weight of tf\*idf were developed to rank the relevant documents and queries [1, 4]. Since mid 1990s to 2000s, machine learning has been used to learn the ranking, i. e. Rocchio’s relevance feedback algorithm [1, 4]. Since the year of 2012, the search engine giants, i. e. Google and Baidu have been incorporating deep learning to their search engines [5].

As the fast development of world wide web in the 1990s, search engine become a typical and important application of IR system. This is called the web search 2.0 of the IR system [3]. The idea of web search is to use the links between pages to construct a polite crawler to traverse and gather most web pages on the entire internet [4]. The two important tasks are: link analysis and relevance ranking [1, 4]. Web pages have much richer structure than the plain texts [3]. A hyperlink between web pages denotes the page relevance. The anchor of the hyperlinks points to the target page. The anchor texts describe the contents of that target page. Those are very useful sources for page ranking [1, 4]. Google used the anchor text in the early version of its search engine [1, 4]. The links of web pages are constructed in a graph, the link analysis is modeled as random walk and the page rank is calculated using Markov chains. Another important method of link analysis is Hyperlink Induced Topic Search (HITS). This algorithm was developed by Kleinberg in 1998 [1]. In this algorithm, hubs and authorities on a specific topic are determined to calculate the relevance.

The architecture of a typical search engine is shown in figure 1. As we can see from figure 1, the relevance ranking is the summation of two parts: IR score and Page Rank. The IR score calculates the content similarity using cosine score. The page rank calculates the importance and measures the quality of the page.

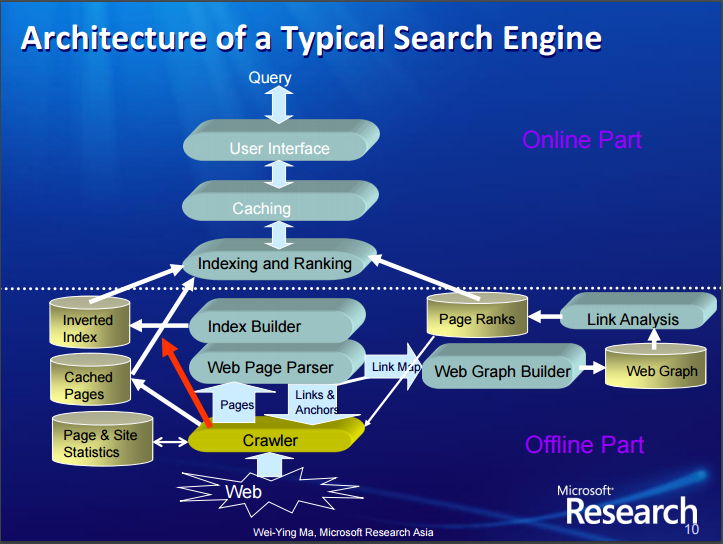


Figure 1. Architecture of a Typical Search Engine [3]

**2. Programming languages**

Version control: Github

Languages: Python 2.7, PHP, HTML, CSS

**3. Design of our search engine**

In this final project, three versions of search engine have been implemented. The topology of our search engine is shown in figure 2. The web crawler with multi-threaded spiders has been developed to feed crawled web pages for the three search engines. As shown in figure 2, the documents were preprocessed with the following four steps.

a). Remove the HTML tags

b). Excluding punctuations and special characters

c). Convert into lower case

d). Using NLTK package to remove words in a stop list and do Porter stemming

The preprocessing for query is steps of b) c) and d) as shown previously.

As shown in figure 2, the search engine 1 is marked using red rectangle. The inverted index has been constructed which contains the terms dictionary and posting lists. The total number of distinctive terms in the inverted index is **(Add this number here ???)** Based on the inverted index, we built the TF-IDF table for all the documents. Boolean model and Vector Space model have been built based on the TF-IDF table. The relevance between documents and query is calculated using cosine similarity. The search engine 2 has been implemented by adding the term proximity into the search engine 1, which is shown using blue rectangle in figure 2. The search engine 3 has been built by expanding search engine 2 with relevance feedback, which is labeled using green rectangle.

Figure 2. Topology of our three versions of our search engine

**3. Methods**

In this part, we introduced the design of web crawler, the procedures of preprocessing, the methods used in Boolean model and Vector Space model, the term proximity and relevance feedback.

**3.1 Web crawler**

Web crawler is used to traverse and gather web pages in specific domain on the internet. The topology of web crawler is shown in figure 3. The web crawler contains the following three components.

a). the first component is the MyHTMLParser, which inherits from the class of HTMLParser in the module of HTMLParser in python. In this component, the web page is fetched and parsed, and then the relative urls are normalized to absolute urls using urljoin().

b). the second part is the page filter (get\_page\_html in the source code), which removes those unwanted link (remove the links if “mailto”, “Tel:”, “.pdf” in the link, or domain name is not in the link).

c). the third component is the threaded spider. In this component, the crawled urls are stored and download in the files Crawled and DownLoad, respectively. The new urls obtained by MyHTMLParser are stored in the queue in the URL frontier.

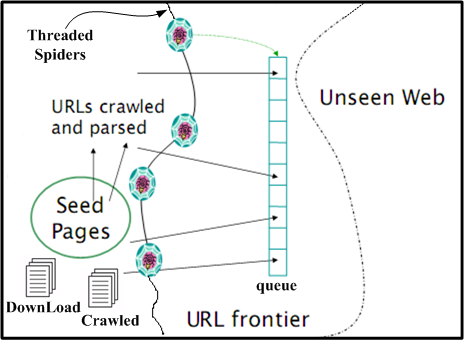
****

Figure 3. The topology of web crawler

Given a seed page, the urls in this seed page is parsed and stored in queue using one spider. Then, multi-threaded spiders are used to crawl web pages. Each spider gets one task from the queue, and the spiders won’t stop crawling until the queue is empty. The crawled web pages are then feed into the three versions of search engine.

**3.2 Preprocessing**

**3.3 Inverted Index & Boolean model**

Build the inverted index: DF dictionary and TF dictionary.

Build a Boolean table which is used to select the candidate documents which contain at least one term in the query.

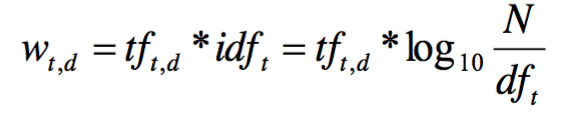
**(Those are from slides of final presentation.)**

**3.4 Vector space model**

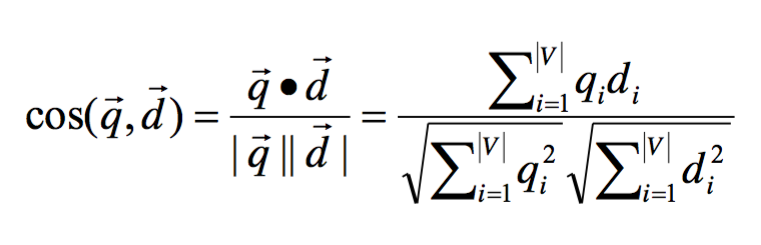
For building our vector space model, our implemented steps are as following:

Our model will calculate the inverse document frequency (idf) for each term in our dictionary.

By using the tf information in inverted index and idf got from the previous step, our model will calculate the tf–idf weight for each term in different documents (by the formula below) and provides the tf-idf documents vectors.

 (1)

Our model will calculate the length for each vector. For query, once our users input their query, our model will calculate the vector length for query by the same method. And then our model will use a Boolean model algorithm to select the query related documents as candidate documents (the candidate should contain at least one term in the query). Last, our model will calculate the similarity between query and each candidate document and sort the documents by similarity as output. We used following formula for similarity calculating:

 (2)

**(Those are from slides of final presentation.)**

**3.5 Words Locationing**

Words Locationing is done based on processed documents.

Only executed when results are shown on the webpage.

First, get all matches in each document.

Second, calculate the midpoint for all matched index and show up to 100 words around the midpoint.

Third, use all matched index got in the first step to do highlighting.

**(Those are from slides of final presentation.)**

**3.6 Term proximity**

Search engine 2 is implemented by adding term proximity into the scoring mechanism of our basic version.

Total Score = a \* Similarity + b \* Term proximity (3)

a and b are the variables used to control the weight proportion between similarity and term proximity. In our case, we set a =0.5 and b =0.5 and the coming out results seemed make sense.

To scoring the term proximity, we used the smallest window algorithm. Basically, each candidate document may have multiple windows that contain all terms of the query (We set the term proximity score to 0 if the document does not contain all terms of the query). We will select the window with min length for term proximity calculating since the terms in the query are more closely related in a document may tell the document are more closely related to the searching query. To implement the min-window algorithm, we used a two points idea as following:

First of all, we set two points at the begging of the document. And then we advance the point1 until the window contains all terms of the query and then we know that we have a solid window that contains all terms. Afterward, we push the point2 until the window won’t contain all terms if we push it one more position. Now we have one of the smallest windows. Then we advance the point1 to repeat this process until we find all possible min-windows. Finally, we can select the smallest window from our min-window candidates as our selected min-window.

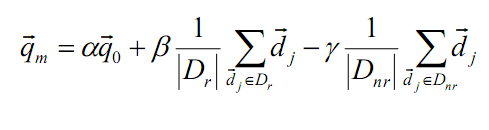
Since we assume the smaller the window size represents the better score, we then simply calculated our term proximity score as:

Term proximity score = 1/(the length of min-window)

Finally, we can recalculate our search score by using formula (3) above. And then we sort the new score will give us the new searching result combine similarity and term proximity.

**3.7 Relevance feedback**

The Rocchio’s algorithm shown in equation (3) is used in the implementation of relevance feedback.

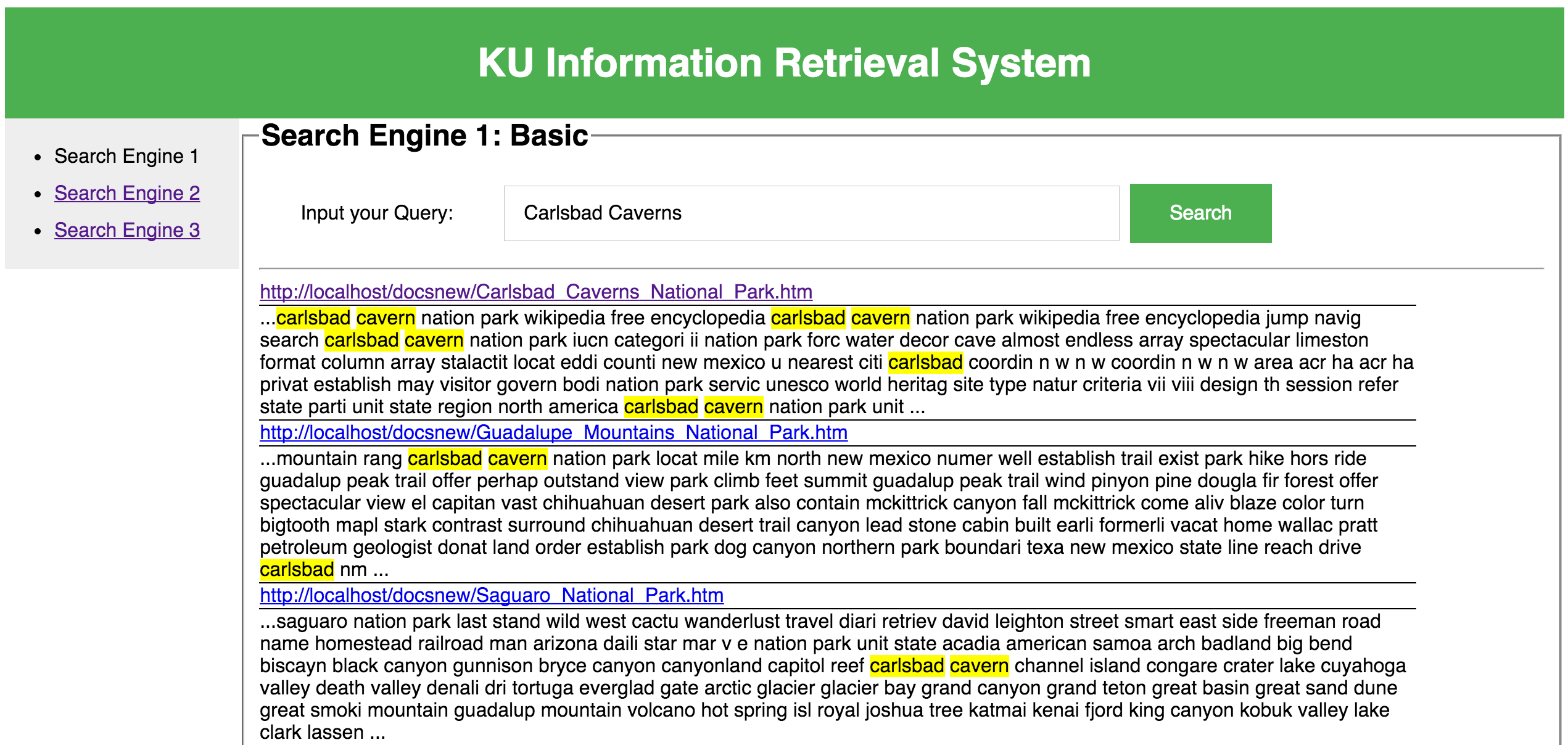
****  (4)

Search engine 3 is implemented by adding the relevance feedback user option to search engine 2.

**(Those are from slides of final presentation.)**

**4. Experiments**

**Add results when using different queries in the three versions of search engine??**

****

**(Those are from slides of final presentation.)**

**5. Conclusions**

**Reference**

[1] Introduction to Information Retrieval, by Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Cambridge University Press. 2008.

[2] <https://en.wikipedia.org/wiki/Information_retrieval>

[3] <http://research.microsoft.com/en-us/collaboration/global/asia-pacific/talent/webirhistoryfuturetrends.pdf>

[4] Sanderson, Mark, and W. Bruce Croft. "The history of information retrieval research." Proceedings of the IEEE 100.Special Centennial Issue (2012): 1444-1451.

[5] <http://www.wired.com/2016/02/ai-is-changing-the-technology-behind-google-searches/>

**Project log**

**Responsibility of each group member:**

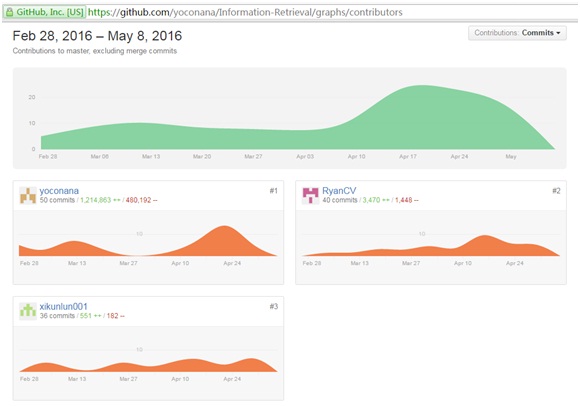
Xi Chen:

Yang Tian:

Yuanwei Wu:

Stop words and stemming for documents, TF-IDF calculation for documents, web crawler, UI design of search engine 2, slides for final project presentation, final paper report

**Log of group activity:**



**Source code**