Web Search Engine on UIC Domain

CS 582 Final Project

Rahul Sai Samineni
Computer Science
The University of Illinois at Chicago
Chicago, Illinois
rsamin4@uic.edu

***Abstract*—This document is the final course project report for the CS582 Information Retrieval course at the University of Illinois at Chicago in the Fall 2022 semester. The purpose of this project is to design and implement his web search engine for the UIC domain range. Search engines include web crawlers, web page preprocessing and indexing, and IR systems that implement vector space models to retrieve web pages relevant to an input user query.**

***Index Terms*—Web Crawler, IR system, Vector-space model, etc.**

I Software Description

The software is designed in Python3 and all features are modularized so it can be further extended for use in other projects and future work. Search engines crawl, collect, and retrieve over 6000 websites from UIC domains. This will take approximately 2-3 hours. To run your code without crawling the UIC domain first, the Pickle folder contains all the pickle files needed to run your code. You can quickly run the search engine by running the searchquery.py script file from your terminal. However, we also provide Python scripts for performing web crawling and pre-owning web pages.

I-A Web Crawler

This module is responsible for crawling the web pages to extract all the links and page content that will be used as output for search engines.

The web crawler is executed by the script crawler.py. Web traversal begins from the UIC-CS department page (https://www.cs.uic.edu/) and is restricted to crawling only within the UIC domain (https://www.uic.edu/). The crawler follows a Breadth-First Search (BFS) strategy with the UIC-CS page as the root node while maintaining a First in First Out queue to store URL links to traverse next. For each web page traversed, its link is appended to a list of crawled pages, the HTML content is downloaded and parsed using the Beautiful Soup library, and all the links present on the page are extracted. Each link encountered is appended to the FIFO queue only if it belongs to the UIC domain, does not have an irrelevant file extension, and is not already present in the queue.

The crawler performance is optimized by using a dequeue instead of a list to implement the FIFO queue, as a dequeue can remove elements from the head in constant time. In contrast, a list can remove elements only from the tail in constant time. For the BFS strategy, links are popped from the head of the queue, and hence dequeue was chosen.

22 file extensions were marked as irrelevant and ignored for web traversal as they took much time to download and did not point to valid web pages that can be parsed and preprocessed. The file extensions ignored are: ’.pdf’, ’.doc’, ’.docx’, ’.ppt’, ’.pptx’, ’.xls’, ’.xlsx’, ’.css’, ’.js’, ’.aspx’, ’.png’, ’.jpg’, ’.jpeg’, ’.gif’, ’.svg’, ’.ico’, ’.mp4’, ’.avi’, ’.tar’, ’.gz’, ’.tgz’, ’.zip’.

The links were processed before being appended to the FIFO queue. The query parameters indicated by ‘?’ and the intra-page anchors indicated by ‘#’ were removed so that each URL added to the queue is a unique web page.

The BFS traversal of links gets terminated if the FIFO queue gets empty or on achieving the crawling limit. The crawling limit was set to 6000 to make the search engine more comprehensive. It took close to 3 hours to crawl 6000 pages.

Each web page is downloaded to the folder

'PagesFetched'. A dictionary saves the links of the pages that have been parsed and downloaded. The key for each link is the name of the file which stores the downloaded content of that link. The dictionary is saved on disk for later use by pickling it.

The file error_log.txt stores all the links that the crawler could not parse/downloaded, along with the associated reason for the exception for that link.

I-B Preprocessing

The preprocessing of the downloaded web pages is executed by the script preprocessing.py. For each downloaded page, a Beautiful Soup object is initialized. At first, all text on the web page is extracted, and then all text that is never visible on a browser is excluded. To achieve this the text inside HTML tags $< script >$, $< meta >$, and $< style >$ are excluded. After this, the standard preprocessing steps are performed on the extracted text to tokenize the text: removal of all punctuation, numbers, and stop words. Poster Stemmer from the nltk library is used to stem each token. After stemming, any residual stop words and tokens with a length of 1 or 2 characters were removed. The tokens of each downloaded web page are stored as a dictionary.

After all, the text preprocessing is complete, and tokens are formed, an inverted index is computed. This inverted index is a dictionary of dictionaries. Each token of the vocabulary of the corpus of web pages is the key, and its corresponding value is a dictionary of web pages where the token occurs. The filename of the downloaded web page is the key in the internal dictionary, and the count of the token on that particular page is the corresponding value.

The preprocessing of each downloaded web page and the computation of the inverted index takes around 30 minutes to complete. So, like the dictionary of all crawled pages in the crawler, the inverted index and the dictionary of tokens are saved to disk for later use.

I-C Vector Space Model

The searchquery.py is executed for taking in user queries and retrieving the most relevant web pages from the UIC domain.

At first, all the pickle files for the links crawled, tokens of web pages, and the inverted index are unpickled and extracted.

The Inverse Document Frequency for each webpage-token pair is calculated using the inverted index and stored as a separate dictionary. The frequency of the most frequent token in each webpage is calculated, which will be required to calculate the Term Frequency of each token for its corresponding webpage. Then finally, the TF-IDF value for each token in the corpus is calculated and stored as a dictionary of dictionaries like the inverted index.

The document length of each web page is calculated to form document vectors for each webpage.

A query is an input by the user, and the query is preprocessed and tokenized, just like the webpage tokens. Then, the cosine similarity between the query and each downloaded webpage is calculated. The webpages are then sorted in the decreasing order of cosine similarity scores and displayed to the user.

II Challenges

- Web Crawling is very new to me. This is the first time I am working on web crawling. so it took a while to understand the basics of crawling. It took a long time compared to the rest of the project because we have to crawl 6000 pages and have some strange URLs which when encountered, had to be handled to prevent any edge cases.
- When I first implemented the engine with a basic crawler, I ran into a lot of issues like having a lot of JavaScript code terms in the inverted index table. Then I modified the crawler to remove the code under certain tags to retrieve only the body content of a website. An annoying thing was the fact that I had to crawl 10000 pages more than once, because I found the

pages in various and wrong formats. In the end, the entire list of formats that I had to blacklist had a size of 18 and it included ".docx", ".doc", ".avi", ".mp4", ".jpg", ".jpeg", ".png", ".gif", ".pdf", ".gz", ".rar", ".tar", ".tgz", ".zip", ".exe", ".js", ".css", ".ppt". For a more comprehensive web crawler, document/PDF files could help in improving the precision of the search engine, but in my case, these files were just increasing the time to download and parse these files.

- When parsing each downloaded webpage and extracting the text from them, the Beautiful Soup object was returning all different kinds of words from the webpage. These were not English words and were forming junk tokens, which could have impacted the performance of the vector space model. So I searched online for a whole day and, in the end, found a very helpful Stack Overflow post to address this issue. As per the post, ignoring the text inside tags $< script >$, $< meta >$, $< style >$, etc. HTML tags, the text fetched are the words that are actually visible on the webpage on a browser. Practically a user is looking for only this text when searching on a search engine.

III Weighting Scheme and Similarity Measure

III-A Weighting Scheme

The weighting scheme that I'm going to use in the project was the simple TF-IDF of the words on the web pages. Despite being a relatively simple and straightforward weighting scheme, it has proven over the years to be one of the most effective and efficient weighting schemes. It works very well because it considers both the impact of a term's frequency in the document and its frequency as well. First, we compute the Term-frequency. Next, we determine the inverse document frequency, which is defined as the specificity of a term that can be quantified as an inverse function of the number of documents in which it occurs. After defining both TF and IDF these are combined to produce the ultimate score of a term 't' in a document 'd'.

III-B Similarity Measure

The similarity measure that I am going to use to rank relevant web pages was cosine similarity. Cosine similarity accounts for both document and query length, Additionally, tokens that are not in either the document or the query do not affect their cosine similarity. Queries are usually relatively short, so the vectors are very sparse. This means that similarity scores can be calculated quickly.

III-C Alternative Similarity measures

Other similarity measures that can be used instead of cosine similarity include Inner product, Dice Coefficient, and Jaccard Coefficient. The Inner Product does not consider the length of the document or query and hence does not give good results when used. The cosine Similarity measure itself is a modification of the Inner Product measure. Jaccard similarity is good for cases where duplication does not matter, cosine similarity is good for cases where duplication matters while analyzing text similarity. The Dice coefficient is another similarity metric tested in this study; however, it is ineffective because it only considers the presence of terms rather than their frequency.

IV Evaluation

The IR system was evaluated using five custom queries and the top 10 web pages retrieved for each query.

1) Query: Registration

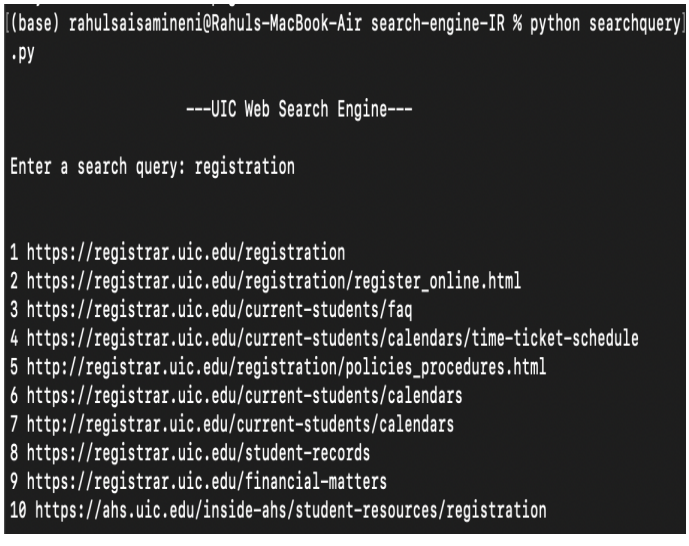

Fig. 1. Registration query

- All the retrieved queries are related to registration. Precision = 1.0

2) Query: Spring Course Catalog

```
(base) rahulsaisamineni@Rahuls-MacBook-Air search-engine-IR % python searchquery
.py

                    ---UIC Web Search Engine---

Enter a search query: Spring Course catalog


1 http://catalog.uic.edu
2 https://catalog.uic.edu
3 https://catalog.webhost.uic.edu/ucat/cat1113archive
4 https://catalog.uic.edu/ucat/archive-links
5 https://catalog.webhost.uic.edu/ucat/cat1315archive
6 https://catalog.uic.edu/search
7 https://catalog.uic.edu/gcat/degree-programs
8 http://catalog.uic.edu/gcat/degree-programs
9 https://catalog.webhost.uic.edu/ucat/catalog/pdf/index.html
10 http://catalog.uic.edu/ucat
```

Fig. 2. Spring Course Catalog

- All the retrieved queries are related to spring course catalog. Precision = 1.0

3) Query: Internships

```
(base) rahulsaisamineni@Rahuls-MacBook-Air search-engine-IR % python searchquery
.py

                    ---UIC Web Search Engine---

Enter a search query: Internships


1 https://careerservices.uic.edu/students/internships
2 https://engineering.uic.edu/news-stories/reaping-the-benefits-of-an-internship
-experience
3 https://cs.uic.edu/news-stories/reaping-the-benefits-of-an-internship-experien
ce
4 https://latinocultural.uic.edu/internships
5 http://engineering.uic.edu/undergraduate/guaranteed-paid-internship-program
6 https://engineering.uic.edu/undergraduate/guaranteed-paid-internship-program
7 https://ecc.uic.edu/students/freshman-internship-program
8 http://ecc.uic.edu/freshman-internship-program
9 http://go.uic.edu/fig
10 https://careerservices.uic.edu/students/flames-internship-grant
```

Fig. 3. Internships query

- All the retrieved queries are related to internships except one. Precision = 0.9

4) Query: Advanced Topics in Software Engineering

```
(base) rahulsaisamineni@Rahuls-MacBook-Air search-engine-IR % python searchquery
.py

                    ---UIC Web Search Engine---

Enter a search query: Advance Topics in Software Engineering


1 https://ask.library.uic.edu/faq/345874
2 https://library.uic.edu/help/article/1957/use-assistive-technologies
3 https://engineering.uic.edu/undergraduate/majors-and-minors
4 https://it.uic.edu/student/software
5 https://it.uic.edu/services-support/student-resources/software-resources
6 https://ece.uic.edu/undergraduate
7 https://cs.uic.edu/profiles/mark-grechanik
8 https://ece.uic.edu/undergraduate/ce-major
9 https://www.cs.uic.edu/~drmark/
10 https://www.cs.uic.edu/~drmark
```

Fig. 4. Advanced Topics in Software Engineering

- All the retrieved queries are related to software engineering courses or articles except two. Precision = 0.8

5) Query: Computer Science

```
(base) rahulsaisamineni@Rahuls-MacBook-Air search-engine-IR % python searchquery
.py

                    ---UIC Web Search Engine---

Enter a search query: computer science


1 https://cs.uic.edu/undergraduate/cs-major
2 https://cs.uic.edu/undergraduate
3 https://catalog.uic.edu/ucat/colleges-depts/engineering/cs
4 https://cs.uic.edu/undergraduate/women
5 https://today.uic.edu/uic-aims-to-break-through-gender-gap-in-computer-science
6 https://cs.uic.edu/news-stories/computer-science-department-welcomes-new-facul
ty-members
7 https://cs.uic.edu/undergraduate/student-organizations
8 https://engineering.uic.edu/news-stories/computer-science-department-welcomes-
new-faculty-members
9 https://cs.uic.edu/news-stories/uic-breaks-ground-on-new-computer-design-resea
rch-and-learning-center
10 https://cs.uic.edu/news-stories/uic-faculty-member-dale-reed-helps-bring-comp
uter-science-to-every-chicago-public-school-student
```

Fig. 5. Computer Science

- All the retrieved queries are related to computer science. Precision = 1.0

V Error analysis and Results

1) The search engine worked well with custom queries. Most results are actually what you're looking for most of the time.
2) In all the questions above, one thing struck me, Since we used TF-IDF and cosine similarity, more weight on terms Frequency affecting results machine, study, research, etc.
3) Overall, if the query details are clear and good, the results fetched will be the most accurate.
4) In some cases, the results are related to. There are some dependencies between query results such as both the website and its domain both pages are fetched. it should be taken to be careful not to duplicate information.
5) Vector space model IR system based on TF-IDF schemes, words that occur more frequently in documents play a big role in defining a theme in that particular document. some words are not present in the document, but very often in their own document. by becoming Reduce ambiguity in getting the most similar documents as TF plays an important role in ranking documents.
6) An error I noticed when evaluating the query results was that some links were repeated in the results. While the crawler looks for duplicate links during link traversal, there are some links that exist on web pages with 'HTTP' and 'HTTPS' extensions. Crawlers now treat links pointing to the same page but with different

HTTP formats as different links. This can be worked on in the future.

VI Related Work

In order to create a web crawler, I ended up looking for various third-party crawlers online and building one from scratch with their requirements library and documentation.

I also found some papers. One describes the integration of the text rank algorithm into the vector search model and the other describes how to return the results. Relevant to the context of a search engine search query.

VII Future Work

The precision of implemented web search engines is pretty decent and acceptable considering it was developed from scratch.

However, The search engine performance can be improved by adding various features throughout the search engine.

- A GUI application can be made as the front end for the search engine using packages such as ”easygui”.
- The crawled pages need to be refreshed from time to time to update the search results.
- Text rank and query context determination algorithms built into the IR system further improve accuracy and retrieve more accurate pages.