A Web Search Engine

Saisuma Dodda
Department of Computer Science
University of Illinois at Chicago
sdodda4@uic.edu

ABSTRACT

A search engine is an information retrieval system that indexes web pages out of numerous pages available on World Wide Web (WWW) based on the search query. This project describes designing and implementing a web search engine for UIC domain from scratch. This search engine includes deploying a web crawler integrated with an intelligent IR (Information Retrieval) system. The software was built modularly starting from crawling web pages restricted to UIC domain, preprocessing the web pages, indexing followed by adding a friendly user interface and then integrating the system with an intelligent component which implements page-rank algorithm.

Keywords

Search engine, Web crawling, Vector space model, Page-rank

1. INTRODUCTION

A quintessential web search engine includes main components like:

1. A spider (web crawler) for crawling the web pages
2. A preprocessor that parses each web page by removing unwanted tags and retaining only required data
3. An indexer to compute inverted index for the vocabulary of the web pages
4. An intelligent component which is page-rank algorithm for this project
5. A user friendly interface to interact with the search engine

The software is written in python3 in a user readable format in order to make it easily extensible for future work. As crawling thousands of web pages is time consuming, over 5000 pages from UIC domain have been crawled, pre-processed and the corresponding pickle files have been included to ultimately help in getting top hits for user queries in no time.

The remaining part of this report is organized to detail main components, major challenges encountered in building the search engine, discussion on weighing scheme, similarity measure along-with a comparison of possible alternatives followed by how to run and discussion on evaluation of results and future scope of this project.

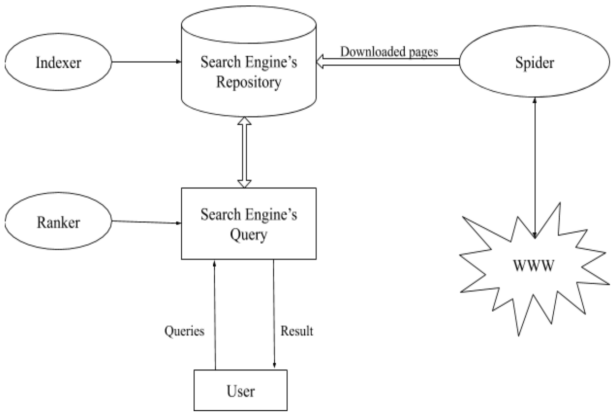


Figure 1: Search Engine flow

2. MAIN COMPONENTS

This section discusses the major components of a web search engine.

2.1 Spider

The web crawler or spider starts from UIC CS domain and crawls through the www gradually implementing a Breadth First strategy (BFS) using Beautiful Soup, a Python library. It starts at a root node (https://www.cs.uic.edu/), traverses through all the hyperlinks adding each of it to a queue and the process is repeated until the queue is empty or the threshold of 5000 pages is reached within the specified domain.

All those URL's already visited are tracked in order to avoid duplication and it has been ensured that every URL belongs to *uic.edu* domain. Multithreading provided by threading library has been used to parallelize and accelerate the crawling process. Each URL is dequed, opened using *urlopen* from request library only if it's status code is 200 (success). The URL is modified after dequeuing, *http* becomes *https*, every URL ends with a slash, the intra-page links represented by '#' have been removed and all the relative paths have been converted into absolute paths. All the links maintained in a dictionary are dumped into a pickle file to be used in future.

2.2 Webpage preprocessor

After crawling, all the pages have to be parsed in order to grab data from it. So, each page is preprocessed by eliminating SGML tags like *<script>, <script>, <meta>, [document]* and getting the plain text from remaining tags using ***html.parser*** available in Beautiful Soup library.
Then, the text is stripped, tokenized (split on whitespace), punctuations and digits are removed, words are stemmed using *PorterStemmer* available in *nltk*, the stopwords are eliminated based on stop_words list available in *nltk* and the words of length lessthan 2 have also been eliminated. The words retained have been stored in a dictionary of vocabulary with corresponding frequency of occurrence. The user query is also preprocessed in the same way.

2.3 TFIDF Indexer (Vector-space Model)

An inverted index dictionary is constructed for all the words in the vocabulary. It is a hash map like data structure that stores a mapping from words to their locations in the crawled pages with word as key and weight as value. The weight is increased each time the word is encountered. The inverted index keeps track of the following measures for each word:

- **Term Frequency (TF)** - Number of times a word appears in a document
- **Inverse Document Frequency (IDF)** - Number of documents in which the word appears

TF indicates the significance of a particular word in a document. IDF is a measure of whether a word is occurs often or seldom in the corpus. TFIDF is calculated for each word in the corpus by multiplying TF with IDF. Inverted Index dictionary is dumped into a pickle file for future use.

Cosine Similarity is calculated between words in the query and all the crawled pages. The cosine similarity is then sorted in descending order to get the top results for the user query.

2.4 Page Ranker

The search results of a query are then ranked using pagerank algorithm, a method used by Google to measure the importance of web pages crawled. It works by counting the number and quality of links to a page by calculating probability distribution. It does not capture the distinction between hubs and authorities. Hubs are good sources of links while authorities are good sources of content.

The initial idea of pagerank is that the surfer starts at an URL which acts as the source. A node is created for this node in the page graph. All the links in that URL are found and a node for each of them is created. An edge is added between those having connection only if there is no edge already. After the threshold (5000) of crawled pages is reached, an iterative pagerank algorithm is initiated by assigning a pagerank of *1/v* where v is the number of nodes in the pagerank graph. In the next iterations, pagerank is calculated based on number of inlinks and outlinks of the nodes that have inlink to the current node with a damping factor (0.85 in this case) until convergence i.e., pagerank remains constant or till the limit of iterations is reached (10 in this case). For the nodes having no outgoing links, a trail pagerank score has been assigned to avoid dead-ends. All the pagerank scores are stored in a dictionary which is dumped into a pickle file for later use.

2.5 User Interface

A simple user friendly interface was built using Flask a library in Python with basic functionalities like taking input query and displaying results to the user.

Figure 2: User Interface

Figure 3: Top 10 results for the query

The user has options to enter the user query for search, request more pages than being displayed or exit the search engine. When the user enters search query and chooses *search*

the top 10 pages will be displayed and if *want more pages* is chosen then the top 20 pages will be displayed. If *exit* option is chosen then he'll be exited from the search engine provided with a button *Launch Search Engine* to enter again.

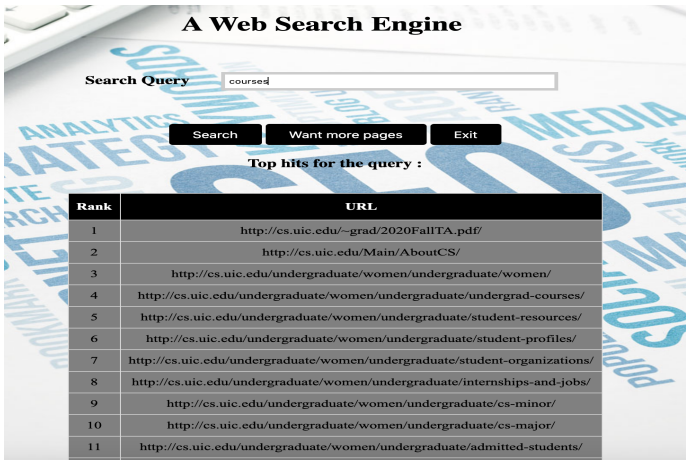


Figure 4: Next 20 results for the query

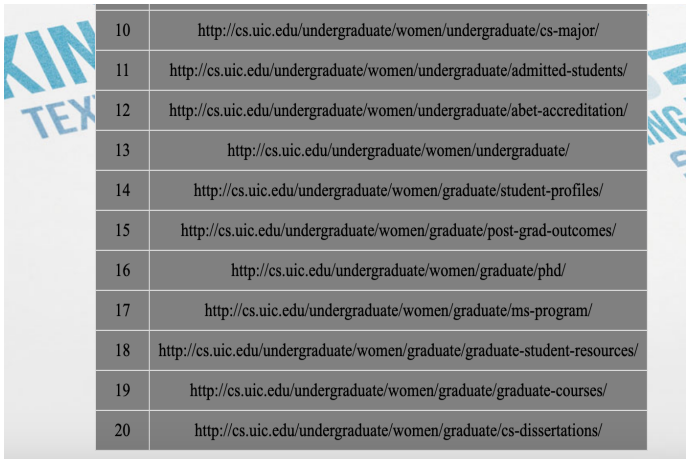


Figure 5: Next 20 results for the query (contd)

Figure 6: Exit search engine

2.6 Intelligent component

The potential intelligent component chosen is ***PageRank*** which ranks a page directly proportional to the number of times a surfer visits the particular page.

2.6.1 PageRank: The Random Surfer

A random surfer starts at a source URL and jumps to other web pages indefinitely in random following all out links from that page with equal probability. Let V be the total set of pages n = $\mid V \mid$, *where* ϵ is the damping or teleport factor.

$$S(A) = \left[(1-\epsilon)\sum_{B \to A}\frac{S(B)}{out(B)}\right] + \frac{\epsilon}{n}$$

Figure 7: Page Rank formula

We repeat the algorithm for all A belongs to V. The results obtained from the vector-space model are ranked using pagerank algorithm for the results to be more consistent.

3. MAIN CHALLENGES

1. The main challenge was in choosing the intelligent component as the implementation of vector-space model using cosine similarity is very naive and simple to retrieve the top relevant documents.

2. Finding ways to speed up the crawler to crawl over 5000 pages was a difficult task as it ran for hours. Then, *Threading* library available in Python came to rescue as it implements multi-threading concept which reduced the run time drastically.

3. Preprocessing the URL and extracting only text removing SGML tags took a bit of effort to figure out tags to be eliminated and tags to be retained.

4. Integrating the pagerank component was a difficult task until the idea of ranking the final results obtained from cosine similarity hit for each different query.

5. Designing GUI using Python was the major exercise as I have never worked on it. Though, I initially started with ***easygui*** later the user interface was implemented using ***Flask*** as it looked more intuitive as a naive user.

4. WEIGHTING AND SIMILARITY

4.1 Weighting Scheme

The weighing scheme used for the search engine was simple TF-IDF of words in the document corpus as IDF normalizes the TF of a word in a document.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Figure 8: TF-IDF Formula

4.2 Similarity Measure

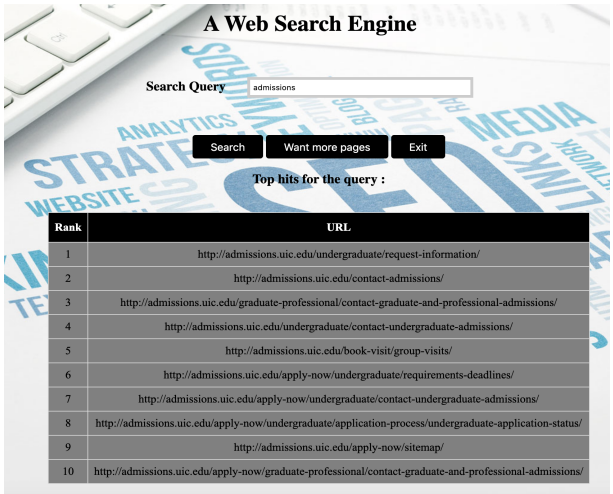
The similarity between documents in the corpus and user query was calculated using ***Cosine Similarity***. It uses the inverted index already calculated and proceeds by calculating the inner product between query and each document in the corpus which is normalized by query and document lengths.

$$\text{cosine}(d_j, q) = \frac{d_j * q}{|d_j| * |q|} = \frac{\sum_{i=1}^{t} w_{ij} * w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2} * \sqrt{\sum_{i=1}^{t} w_{iq}^2}}$$

Figure 9: Cosine Similarity

where, d is the document and q is the query.

4.3 Manual Evaluation



Rank	URL
1	http://admissions.uic.edu/undergraduate/request-information/
2	http://admissions.uic.edu/contact-admissions/
3	http://admissions.uic.edu/graduate-professional/contact-graduate-and-professional-admissions/
4	http://admissions.uic.edu/undergraduate/contact-undergraduate-admissions/
5	http://admissions.uic.edu/book-visit/group-visits/
6	http://admissions.uic.edu/apply-now/undergraduate/requirements-deadlines/
7	http://admissions.uic.edu/apply-now/undergraduate/contact-undergraduate-admissions/
8	http://admissions.uic.edu/apply-now/undergraduate/application-process/undergraduate-application-status/
9	http://admissions.uic.edu/apply-now/sitemap/
10	http://admissions.uic.edu/apply-now/graduate-professional/contact-graduate-and-professional-admissions/

Figure 10: Top 10 results for the query admissions

This section briefs manual evaluation on some of the user queries. I have evaluated results of five queries on admissions, UIC alumni, career fair, Computer science and Undergraduate.

As seen in the Figures 10, 11, 12, 13, 14 most of the results were more relevant to the query when compared to the plain search engine as pagerank uses the concept of equal probability. The results obtained from vector-space model are ranked again using pagerank before giving top hits to the user.

Rank	URL
1	http://advance.uic.edu/alumni-association/
2	http://engineeringalumni.uic.edu/
3	http://uic.edu/alumni/
4	http://advance.uic.edu/alumni-association/alumni-association/
5	http://engineeringalumni.uic.edu/get-involved/
6	http://engineeringalumni.uic.edu/events/
7	http://engineeringalumni.uic.edu/get-involved/join-us-on-linkedin/
8	http://engineering.uic.edu/about/alumni/
9	http://advance.uic.edu/alumni-association/upcoming-events/
10	http://uic.edu/research/student-research/research/student-research/

Figure 11: Top 10 results for the query UIC alumni

Rank	URL
1	http://uic.edu/about/job-opportunities/
2	http://ecc.uic.edu/
3	http://careerservices.uic.edu/
4	http://studentemployment.uic.edu/
5	http://uic.edu/depts/st_empl/
6	http://careers.uic.edu/
7	http://today.uic.edu/93799-2/
8	http://engineering.uic.edu/undergraduate/career-outcomes/
9	http://studentemployment.uic.edu/students/
10	http://ecc.uic.edu/events-2/

Figure 12: Top 10 results for the query career fair

Figure 13: Top 10 results for the query Information retrieval

Figure 14: Top 10 results for the query Computer science

4.4 Discussion on Results

The results returned by the search engine built were almost relevant for the queries except for the query *Information retrieval*. The reason for irrelevant is the limited number of pages crawled and restricted domain.
Table 1 shows the precision values for the five queries evaluated.

Search Query	Precision
admissions	1
UIC alumni	0.9
career fair	0.8
Information retrieval	0.5
Computer science	1

Table 1: Precision for User queries

5. COMPARISON WITH POSSIBLE ALTERNATIVES

The model that included page-rank as intelligent component out performed the plain search engine implementing vector-space model using cosine similarity. The results returned by plain search engine weren't that similar to the query whereas those returned by intelligent component model were more relevant.

6. RELATED WORK

Building a search engine does not just mean implementing some algorithms and retrieving the top results for the query but search engine optimization is the centric task. There has been a constant research going on in this domain along with how to further improve results for a query including web crawling and efficient use of storage.

7. FUTURE SCOPE

The current software has been built considering only unigrams which can be extended to bigrams or trigrams for TF-IDF vectorization. Psuedo relevance feed back can be implemented to improve results. Also, search engine can be extended to other domain as well instead of restricting to UIC domain and number of pages crawled can also be increased so that there will scope for more queries.

8. RUNNING THE SOFTWARE

The files crawler.py, preprocessing.py, page_rank.py, user_interface.py are the python files required. The pickle files invertedindex.pkl, crawled_urls.pkl, pagerank.pkl are provided and need to be placed in the same folder as python files. Run only user_interface.py as it loads all the pickle files and open localhost:<port> in the browser to search.