UIC Web Search Engine

Based on Cosine Similarity and Query Dependent Page Rank

Irshad Babubhai Badarpura
Department of Computer Science
University of Illinois at Chicago
Chicago, Illinois - USA
ibadar2@uic.edu

ABSTRACT

With the advent of the Internet – the world became a better-connected place. It helped to reach many people seamlessly and very effectively. As the internet expanded, it became important to make sure users get the relevant information. This is where the search engine comes into picture. Search Engine helps users find the relevant information available on the internet. Today, Search Engine indexes tens of billions of pages to get the answer the user is looking for. The average person conducts between 3 and 4 searches every day [1].

In this report I will be highlighting the different components I used to build a Search Engine for UIC domain from scratch. The software was built in modules, each module performing a specific task such as Crawling, Tokenizing, Indexing, Page Ranking, and creating a Graphical User Interface.

The software uses the concepts of vector-space model which uses Cosine Similarity and Query Dependent PageRank. Based on the results obtained from using these methods on the same queries show that Query Dependent PageRank outperforms vector space approach. Along, with that, the Query Dependent Page Rank also outperforms the current naïve page rank implementation.

KEYWORDS

Search Engine, Crawler, Tokenization, Cosine Similarity, Tf-Idf, Query Dependent PageRank

1. INTRODUCTION

The software built is a Search Engine which focuses on the websites which fall under the University of Illinois at Chicago (https://www.uic.edu/) domain. The Search Engine starts to crawl from the URL: https://www.cs.uic.edu and checks all the URLs that are pointing to or pointed by the current URL and visits each of them one by one and it continuous this process. For this project I have taken into consideration only the first 3500 URL's after which the search engine stops. The Search Engine uses Breadth First Search approach to recognize which URL's to traverse next.
For each web page traversed the software extracts the information contained in its HTML tags and performs various operations to clean the data and form tokens which will be explained in the following sections.
For ranking the web pages according to the users query two different approaches are used. First, Tf-Idf vectorization which uses cosine similarity to rank the most relevant pages based on the Tf-Idf weight for the web page and the corresponding query. Second, Query Dependent PageRank in which the traditional page rank method is modified based on the user's query. These methods will be explained in the sections below.

2. WEB CRAWLER

The Crawler is designed using the Breadth First Search approach. It starts from: https://www.cs.uic.edu and adds it into a list. From here, it adds all the hyperlinks corresponding to this page into the list and then removes this URL from the list. This process is continued for all the URL's in the list. Another reason for using BFS approach is to avoid cycles. To avoid duplicates the elements of this list i.e., the URL's are then placed into a visited set so that if the same page is traversed again it will be not considered making sure that all the pages that are traversed remain unique. Once all the pages are traversed, we get a set of URLs which are unique and then using this URL's we can perform tokenization on the data for each of this URL. For the purpose of simplicity the following links ['.css', 'mailto:', '.js', '.jpg', '.jpeg', '.png', '.gif', '.pdf', '.doc', '.JPG', '.mp4', '.svg','favicon', '.ico'] were excluded.

3. TOKENIZATION

For each links in the visited set we create a tuple which consists of the URL, the tokens, and the hyperlinks for that URL. For tokenization, I opened each of the link and then using Beautiful Soup extracted the information contained in the various HTML tags – the tags I considered are P, title, H1-H6, span and Div. After getting all the information from these tags, we tokenize the URL on whitespace, removing punctuations, digits and everything is converted to lower case. After this step, stop words are removed and stemming is performed. Stemming is performed to reduce the word to its root word and makes it easy to compare words which have similar roots. After this step, the words with length less than 2 are removed and with this step the tokenization process is complete. This process is carried for each URL in the visited set of URLs. For tokenization word_tokenizer was used from

nltk.tokenize, for stop words nltk.corpus was used and for stemming PorterStemmer was used from nltk.stem

4. TF-IDF INDEXING

Tf-Idf Indexing involves creating an inverted index which acts as a data structure that stores each token which we get after tokenization of each of the URLs. The inverted index is structured as a dictionary whose keys are the tokens and values are a list that contain total number of times that token appears in all the URLs and it also contains a dictionary whose keys are the URL in which this token appears and its values are the number of times this token appears in that URL. The inverted index is then used to calculate the tf-idf of the document. The tf-idf is calculated as follows:

Equation 1: The formula to calculate tf-idf

$$tfidf(t, d, D) = tf(t, d) \times idf(t, D)$$

where t denotes the terms,
d denotes each document (URLs in this case)
D denotes the collection of documents (URLs)

4.1 TF

Term Frequency (Tf) is the number of times each word appears in each document. Then I adjusted the frequency by the raw frequency of the most frequent word in the document. Tf is calculated as follows:

Equation 2: The formula to calculate Tf

$$tf(t, d) = log(1 + freq(t, d))$$

where t denotes the terms,
d denotes each document (URLs in this case)

4.2 IDF

Inverse Document Frequency (Idf) of a word is the total number of documents (URLs) in the collections divided by the total number of times that word appears in all the documents (URLs). The Idf is calculated as follows:

Equation 3: The formula to calculate Idf

$$idf(t, D) = log \frac{|D|}{1 + |\{d \in D : t \in d\}|}$$

where t denotes the terms,
d denotes each document (URLs in this case)
D denotes the collection of documents (URLs)

5. GRAPHICAL USER INTERFACE

The Graphical User Interface of the Search Engine is implemented using tkinter toolkit. This interface basic functionalities using which the user can interact with the search engine.

The initial window displays an entry box where the users can enter their queries and then there two buttons using which the user can select the ranking method. The two options are either Tf-Idf search or Query Dependent PageRank search.

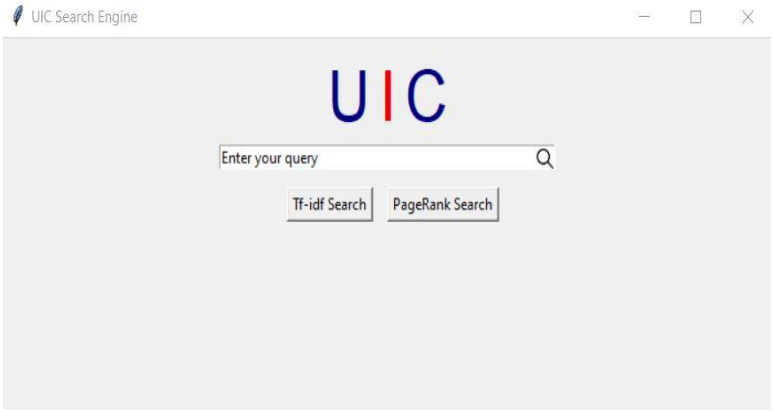

Figure 1: The main page of the Search Engine

Once the user selects the search method a list box displays the top 10 pages that match the query

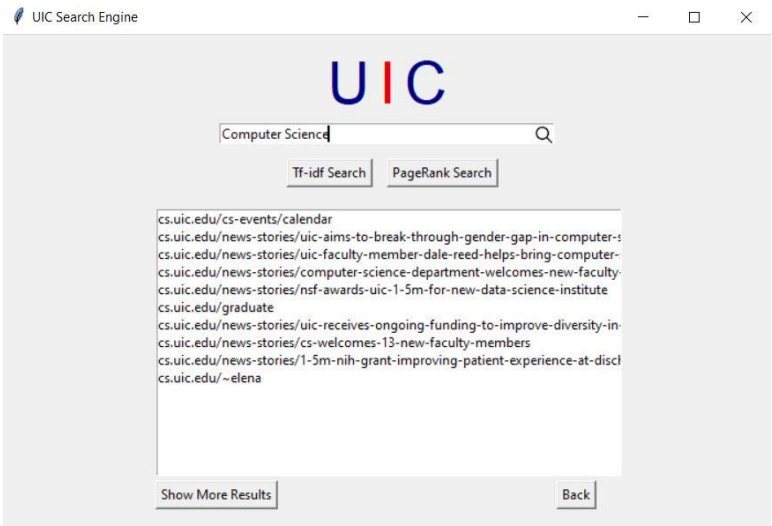

Figure 2: The list box displays the top 10 matches

If the user wishes to see more results than they can click can show more results button and a new window pops up which shows the top 20 pages that match the query

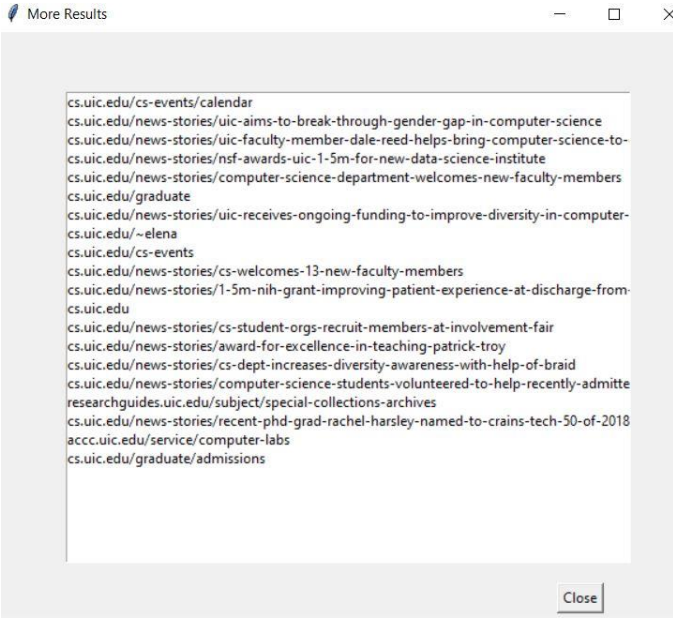

Figure 3: Show More Results Window

6. MAIN CHALLENGES

1. The most challenging part for me in this project was to think of a way in which I can integrate and facilitate the querying not only using just one method but using two methods. Once I figured this out, I understood how I had to start building my application and what would my application look like.
2. I also am new to Python so during the implementation I had to spend a lot of time leaning about the different python libraries and how to extract the data from different pages using beautiful soup. Also, I had to spend some time on learning about web scrapping and how to go around it, how to create a crawler and how to get links coming in and out of a page. Along, with that I also had to figure out which Python library I was going to use for creating the User Interface as I had to learn it and then implement it from scratch.
3. Another challenge which I faced initially was when I encountered certain files which are treated as links but point to an image or pdf. So, after doing some research I collected a list of such possible files and excluded it from the search engine consideration. These links: ['.css', 'mailto:', '.js', '.jpg', '.jpeg', '.png', '.gif', '.pdf', '.doc', '.JPG', '.mp4', '.svg','favicon', '.ico'] were excluded.
4. Another challenge I faced was on deciding the intelligent component which I was supposed to be using for the search engine. During the project proposal I mentioned that I was mostly going to use HITS algorithm but when I started implementing the project I found out it was very similar to naïve PageRank algorithm and it would not give more optimized results. So, I had to look for something which was better than the naïve PageRank algorithm and after some readings I found out Query Dependent PageRank algorithm and how it ranks pages based on the users queries I decided to implement it and it showed better results. Another reason of using Query Dependent PageRank was that since the domain was fixed most of the queries were correlated.
5. Another very crucial challenge was to decide how would I store the pages traversed on the local computer because there were going to be about 3500 pages and I had to store tokens for each of them. Along, with that to implement each functionality or perform tf-idf or PageRank calculation I would have to load the documents containing tokens for each page every time which would drastically slow down the search engine. So, I had to found out a way using which I can directly save the complete data structure so that it becomes easy to directly to open it and access it for any calculations and once all calculations are done I could directly call these files and run the search engine on them and this would drastically improve the efficiency and run time of the search engine. The alternative which I found out was to save everything in pickle.

7. WEIGHTING AND SIMILARITY

7.1 WEIGHTING SCHEME

The Weighting Scheme I used is simple tf-idf as it always places a word according to how important it is to a document in a collection. This importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus [2]. The tf-idf is calculated as follows:

$$w_{ij} = tf_{ij} \times \log_2 \frac{N}{n}$$, where

w_{ij} = weight of term T_j in document D_i
tf_{ij} = frequency of term T_j in document D_i
N = number of documents in collection
n = number of documents where T_j occurs at least once

7.2 SIMILARITY MEASURE

The similarity measure that I used to find the similarity between the page and the query is Cosine Similarity. Cosine Similarity basically measures the angle between two vectors. The two vectors here are the page and the query. It takes the dot product of the page and query vector and divides it by the square root of the sum of squares of weights of both the page and the query, and as we saw the weight is calculated using Tf * Idf. If the cosine similarity is close to 1 it indicates that two vectors are highly similar. If the cosine similarity if close to -1 it indicates that the vectors are highly dissimilar . Using this concept I calculated the cosine similarity for each document and the query and I placed this result into a dictionary where the key of the dictionary contains the document number(URL) and the value contains the cosine similarity value calculated using :

Equation 4: Cosine Similarity Formula

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

After the dictionary is formed I, sorted the dictionary based on the values of cosine similarity top to bottom and this way the ranking is calculated for the vector space model using inverted index, tf-idf, and cosine similarity.

7.3 ALTERNATIVE MEASURES

The Alternative Measures which can be used in place of cosine similarity are Dice coefficient and Jaccard Coefficient. The reason why I used Cosine similarity over Dice and Jaccard coefficient because it takes into consideration not only the dot product of the page and the query but also the product of the page length and the query length. Also, according to a paper[3] published when we compare these similarities cosine similarity gives the best result when retrieving top 10 queries and that is what exactly what I am doing in this project. Therefore, I used cosine similarity over Dice and Jaccard coefficient.

8. INTELLIGENT COMPONENT

The Intelligent component used in my software is PageRank. PageRank basically ranks the pages based on the number of links pointing to that page. It basically indicates how important a page is. PageRank in simple is a "vote', by all the other pages on the Web, about how important a page is. That means a link to a page counts as a vote and if there is no link it means no vote. So, the more the number of votes the higher the PageRank. This was just a high-level understanding of PageRank. In the sections below I will explain PageRank in detail and why I used Query Dependent PageRank rather than just using naïve PageRank algorithm.

8.1 THE RANDOM SURFER MODEL

The initial idea behind PageRank is:

1. A user starts at a web page
2. From the current web page, the surfer clicks on a random link
3. During this random surfing, the user visits some pages more often than others
4. This implies that the Pages visited more often are more important

So basically, in PageRank developed using the random surfer model, a surfer, jumps from web page to web page, choosing with uniform probability which link to follow at each step. To reduce the effect of dead ends or endless cycles the surfer will occasionally jump to a random page with some small probability β, or when on a page with no out-links[4].

For the purpose of calculation, we can reformulate this in terms of a graph where, a web page represented by a node and links from one web page to another represent an edge.

Let W be the set of nodes, N=|W|, Fi be the set of pages page i links to, and Bi be the set pages which link to page i. For pages which have no out links we add a link to all pages in the graph1 . In this way, rank which is lost due to pages with no out links is redistributed uniformly to all pages. If averaged over enough steps, the probability the surfer is on page j at some point in time is given by the formula:

Equation 5: Formula for calculating the probability that a surfer is at a page p(j)

$$P(j) = \frac{(1-\beta)}{N} + \beta \sum_{i \in B_j} \frac{P(i)}{|F_i|}$$

Now, the PageRank(PR) can be given as PR(j) = P(j) because the above equation is recursive and must be iteratively evaluated till P(j) converges[4].

8.2 QUERY DEPENDENT PAGERANK :

The reason I chose not to use naïve PageRank algorithm based on Random Surfer Model or HITS algorithm is because of a problem known as topic drift. This basically means when calculating PageRank either at crawl time or query time may result into pages with the most inlinks in the network being considered as opposed to what matches to the query. This is where Query Dependent PageRank comes into picture to give importance to not just the inlinks but also to the query so that more relevant pages can be retrieved.

Query Dependent PageRank is a more intelligent surfer. It probabilistically jumps from page to page depending on the contents of the page and comparing that with what the surfer is trying to query. The resulting probability distribution over pages is given as[4]:

Equation 6: Formula for calculating probability distribution over pages P(j)

$$P_q(j) = (1-\beta)P'_q(j) + \beta \sum_{i \in B_j} P_q(i) P_q(i \to j)$$

where Pq(i→j) is the probability that the surfer transitions to page j given that he is on page i and is searching for the query q. Pq'(j) specifies where the surfer chooses to jump when not following links. Pq(j) is the resulting probability distribution over pages and corresponds to the query-dependent PageRank score (QD-PageRankq(j) ≡ Pq(j))[4].

Pq(i→j) and Pq'(j) are arbitrary distributions given by:

$$P'_q(j) = \frac{R_q(j)}{\sum_{k \in W} R_q(k)} \qquad P_q(i \to j) = \frac{R_q(j)}{\sum_{k \in F_i} R_q(k)}$$

Both the distributions are derived from the $R_q(j)$, which is a measure of relevance of page j to query q. The choice of relevance function $R_q(j)$ is arbitrary.

9. EVALUATION

I did an evaluation of the precision for 5 random queries created by me. In this evaluation I only considered the top 10 pages(that is only the first 10 pages retrieved). For most of the queries I found that the precision value obtained using the intelligent Query Dependent PageRank is more than the non-intelligent Tf-Idf search. The results of the 5 queries are as follows:

Query 1: "library"

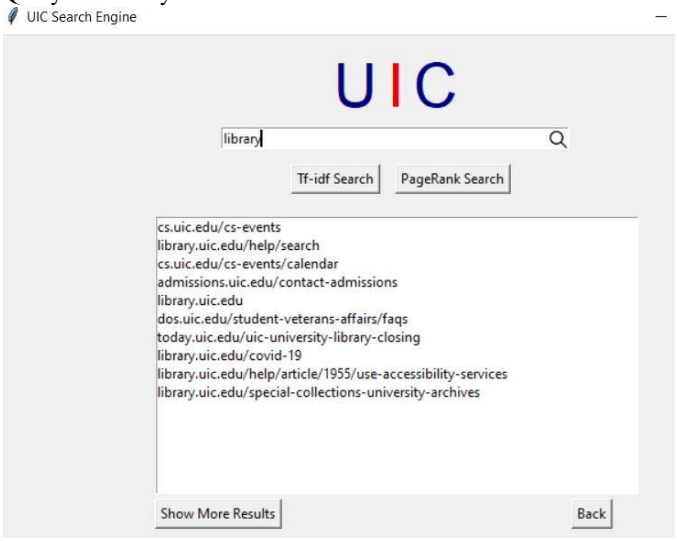

Figure 4: Top 10 pages for "library" using Tf-Idf search

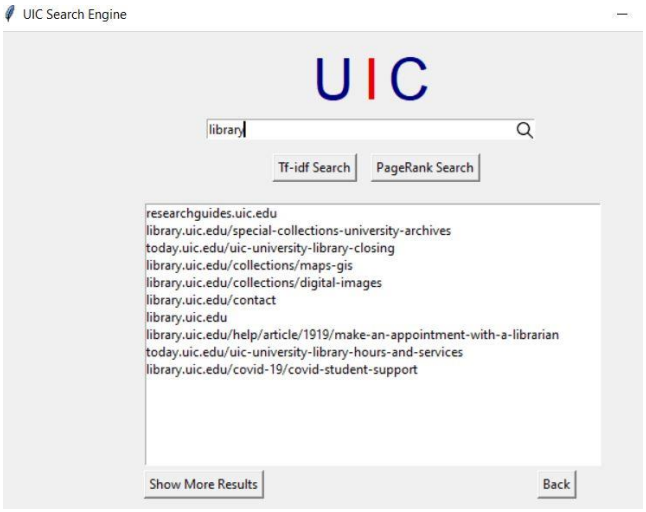

Figure 5:Top 10 pages for "library" using PageRank search

From figure 4 we can see that the precision that we get when we perform Tf-Idf search for query "library" is **0.8.** From figure 5 we can see that the precision that we get when we perform PageRank search for query "library" is **1.0.** So, here we can clearly see that the search with the intelligent component results in higher precision as compared to the one with a non-intelligent Tf-Idf search.

Query 2: "admissions"

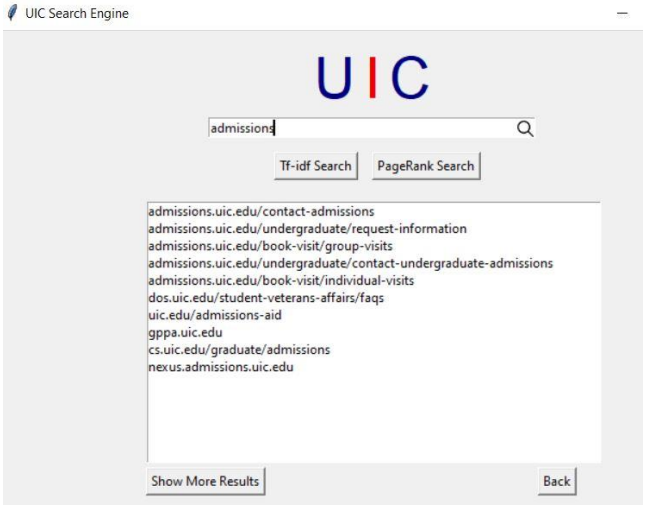

Figure 6: Top 10 pages for "admissions" using Tf-Idf search

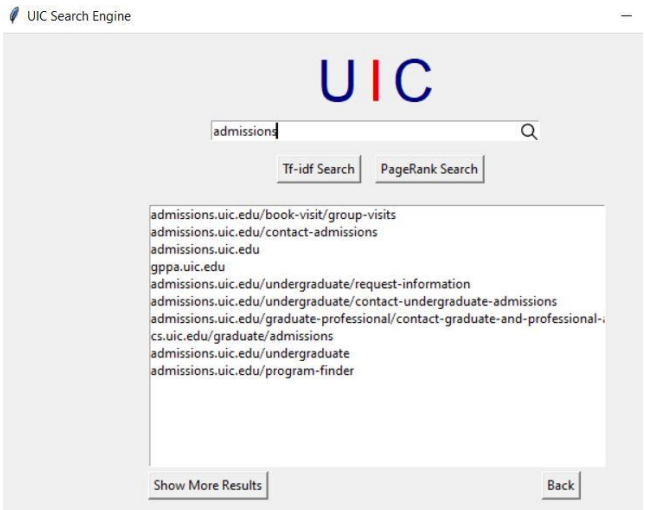

Figure 7: Top 10 pages for "admissions" using PageRank search

From figure 6 we can see that the precision that we get when we perform Tf-Idf search for query "admissions" is **0.7.** From figure 7 we can see that the precision that we get when we perform PageRank search for query "admissions" is **1.0.** From the above figures we can clearly see that the intelligent Query Dependent PageRank successfully retrieves all top 10 pages that match the query term.

Query 3: “Campus Employement”

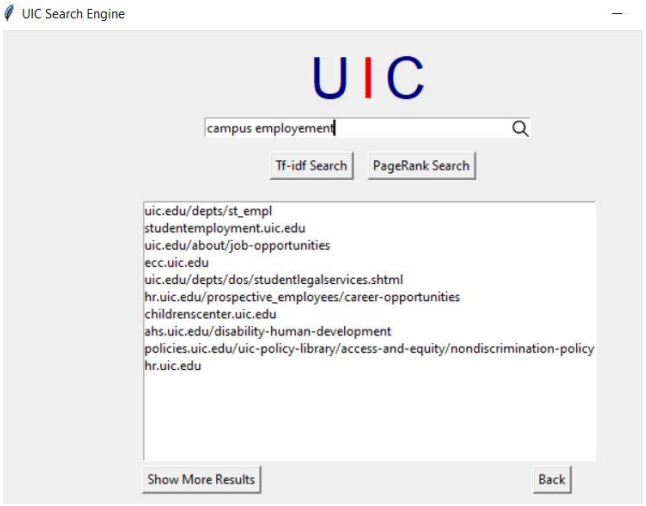

Figure 8: Top 10 pages for "campus employement" using Tf-Idf search

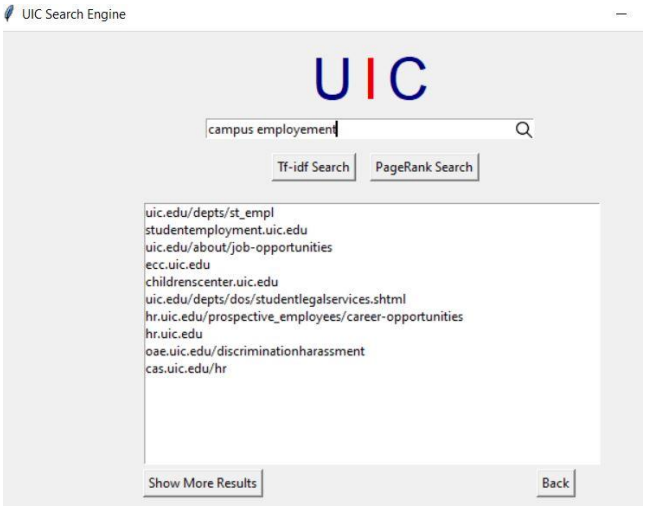

Figure 9: Top 10 pages for "campus employement" using PageRank search

From figure 8 we can see that the precision that we get when we perform Tf-Idf search for query “campus employement” is **0.5.** From figure 9 we can see that the precision that we get when we perform PageRank search for query “campus employement” is **0.6.** Another interesting thing to note is that I purposely entered the spelling of employement wrong, and you can notice that both the searches still manage of get pages related to employment. This is because I performed stemming not only on the tokens but also the query terms. Also, we can see that as we switch from one-word query to two words query the precision reduces for both the Tf-Idf search and the PageRank search.

Query 4: “Degree Programs”

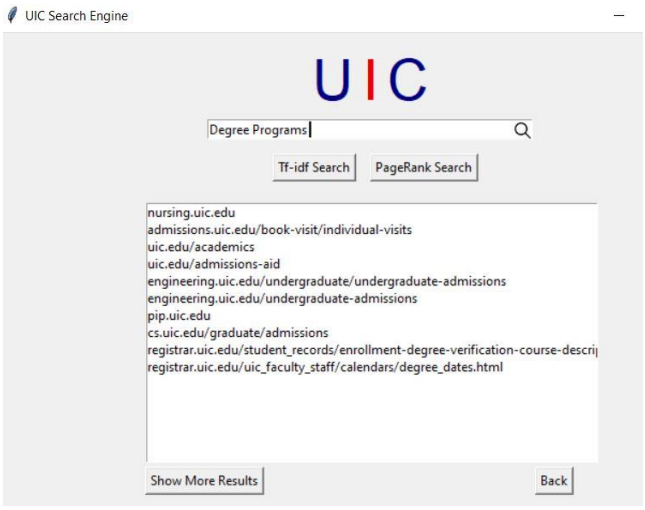

Figure 10: Top 10 pages for "Degree Programs" using Tf-Idf search

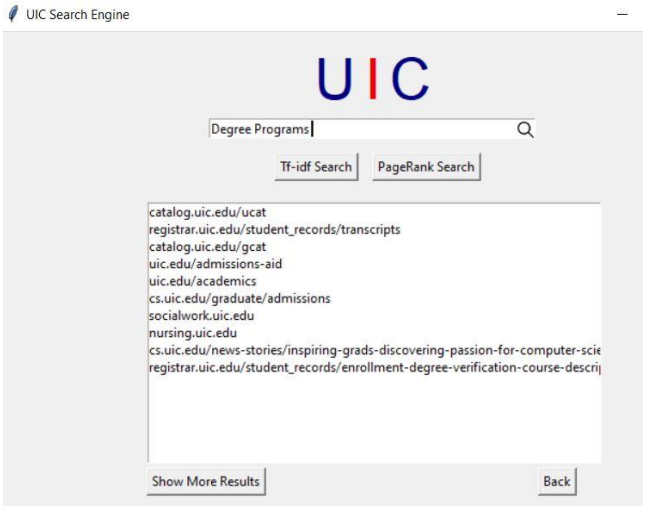

Figure 11: Top 10 pages for "Degree Programs" using PageRank search

From figure 10 we can see that the precision that we get when we perform Tf-Idf search for query “Degree Programs” is **0.4.** From figure 11 we can see that the precision that we get when we perform PageRank search for query “Degree Programs” is **0.7.** An interesting thing to notice here is that the Tf-Idf search does not retrieve the “catalog.uic.edu/ucat” page which is the most important page related to this search query as this page includes all the programs offered at the university whereas the PageRank search retrieves this page(“catalog.uic.edu/ucat”) and that too it is the first page retrieved which shows that my Query Dependent PageRank is more accurate and not just depends on the Tf-Idf score but also depends on the content of the page and the content of the query. Also, the PageRank search retrieves more relevant pages.

Query 5: “College of Engineering”

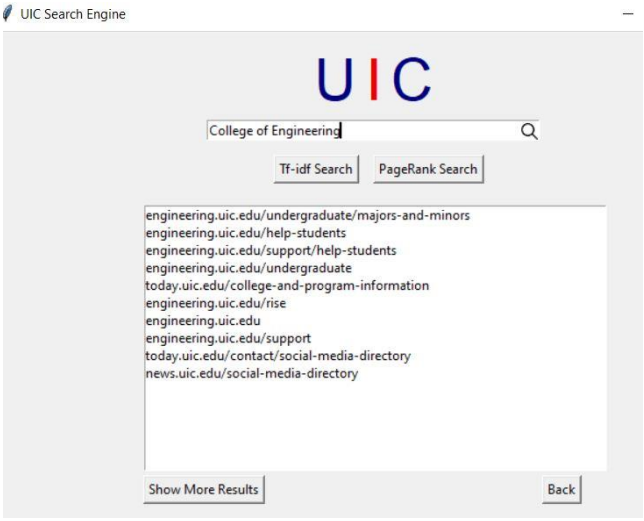

Figure 12: Top 10 pages for "College of Engineering" using Tf-Idf search

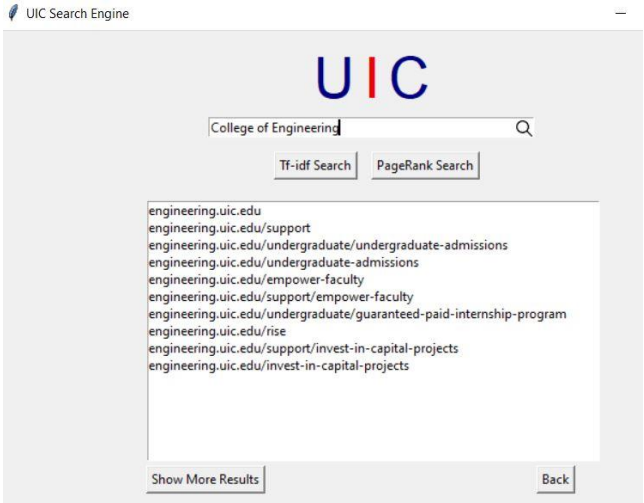

Figure 13: Top 10 pages for "College of Engineering" using PageRank search

From figure 10 we can see that the precision that we get when we perform Tf-Idf search for query “College of Engineering” is **0.7.** From figure 11 we can see that the precision that we get when we perform PageRank search for query “College of Engineering” is **1.0.** From the above figures we can clearly see that the intelligent Query Dependent PageRank successfully retrieves all top 10 pages that match the query term.

Query	Precision	
	PageRank	Tf-Idf
library	1.0	0.8
admissions	1.0	0.7
campus employement	0.6	0.5
Degree Programs	0.7	0.4
College of Engineering	1.0	0.7

The above table summarizes the precision values for Top 10 pages for 5 different queries. From the table we can clearly see that the Query Dependent PageRank clearly performs better than the Tf-idf search. Also, we can see that both the methods work best for single word queries.

10. ERROR ANALYSIS

After performing the manual evaluation as shown in the above table, the result is quite clear that the Query Dependent PageRank is more precise and accurate than the Tf-Idf search and that is something which I expected. Things that worked out well are:

1. If the user wishes to see more results outside the top 10 pages then they can do that with show more result option.
2. For most of the queries the page that was expected to be on rank 1 showed first.
3. Also, the most important parts like crawling, tokenizing, and calculating Tf-Idf and PageRank all worked well.

The thing which did not work well was the difference between the precision for single word queries and two-word queries.

11. RELATED WORKS

Since the inception, various improvisations have been performed on the naïve PageRank algorithm. One such method of improving the PageRank is Proposed PageRank algorithm which utilizes a normalization technique in view of mean value of page ranks. The proposed scheme reduces the time complexity of the traditional Page Rank algorithm by diminishing the number of iterations to reach a convergence point[5].

12. FUTURE WORK

The proposed software engine works well but can be improved in future. Currently, as I mentioned earlier the precision reduces for two-word queries and this can be addressed by using Bigrams and Trigrams for Tf-Idf vectorization. Also, the time required to crawl the web pages can be reduced by using multiple threads. Also, various other features like pseudo relevance feedback, auto suggestions and spell checks for English words can also be implemented.

13. REFERENCES

[1] https://99firms.com/blog/search-engine-statistics/#gref.
[2] https://www.geeksforgeeks.org/tf-idf-model-for-page-ranking/
[3] Vikas Thada, Dr. Vivek Jaglan, 2013, Comparison of Jaccard, Dice, Cosine Similarity Coefficient to Find Best Fitness Value for Web Retrieved Documents Using Genetic Algorithm, Vol. 2 IJET
[4] Mathew Richardson, Pedro Domingos, The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank
[5] Kuber Mohan, Jitendra Kurmi, A Technique to Improved Page Rank Algorithm in perspective to Optimized Normalization Technique Vol. 8 IJARCS