Web Search Engine using links and content information in PageRank

Sheetal Sathyanarayana Prasad
sprasa22@uic.edu
College of Engineering, University of Illinois at Chicago
Chicago, Illinois

ABSTRACT

A web search engine gathers information from the web that matches the user's query and satisfies the user's information need. The aim of this project is to build a web search engine that searches for information in the UIC domain. The search engine constitutes a web crawler , scrapper, processing textual data, ranking the search results using probabilistic combination of links and content of pages in PageRank algorithm, spellchecker for the query and an user interface to display the search results. The results are evaluated manually by comparing the precision values of cosine ranking and the content based PageRank algorithm. The content based PageRank surpasses the cosine ranking in terms of precision for almost all queries.

KEYWORDS

Web Search Engine, PageRank, Random Surfer, Intelligent Surfer

1 INTRODUCTION

There is not a single day of our lives where we don't use Google. Google is the widely used search engine with worldwide market share between 70.83% and 91.98%. The Algorithm used by google search engine is called the PageRank algorithm named after Larry Page who is one of Google's founders. The motivation for this project is to build a search engine for the UIC domain incorporating the PageRank algorithm that considers content information and checks the relevance with the query.

Traditional Information retrieval methods perform poorly on the vast data of the web. The two methods that evaluate web pages efficiently are HITS (hubs and authorities) algorithm and PageRank algorithm. [4] Both the algorithms are based on the link structure of the web and assign high value to the page with higher inlinks. That is, a web page is considered to be more important if it has large number of visitors. HITS algorithm uses the traditional search engine and selects a set of pages , these pages are expanded for inlinks and outlinks. The pages are then categorised as hubs and authorities. Authorities are pages of high quality and hubs are pages that point to authorities. The PageRank Algorithm computes a value for each page at crawl time and is combined with a score at query time. The drawback of PageRank algorithm is that it does not consider the relevance with regards to the query. Despite the drawback PageRank performs more efficiently than HITS algorithm.

The web search engine [2] built for this project overcomes this drawback by combining the page content information relevant to the query and links structure.

2 SOFTWARE DESCRIPTION

The web search engine built for this project constitutes many modules which are briefly described in this section.

2.1 Web Crawler and Web Scrapper

In this project crawling of web pages is done within the UIC domain. The crawler starts from https://www.cs.uic.edu/ and crawls web pages in the UIC domain until either all web pages are crawled or 3000 web pages are crawled, whichever occurs first. Using BFS algorithm we create a set of visited lists that are unique and have no cycles. BFS algorithm is implemented by using a queue to which the Department of CS link is added. Each link encountered in the starting node is added to the queue. All the links that have been encountered are added to visited only if it hasn't been visited, by this method we avoid cycles and avoid revisiting already visited links.

For all links in the visited set, textual content is retrieved by scrapping relevant HTML tags. All the links are retrieved by scraping the content of 'a' tags in the HTML pages. Some tags considered in the project are headers, title , span, div, p etc. After scraping the web page for textual information this data is tokenized using word tokenizer. In the list of tokens punctuation are dropped along with numerical data. Stop words are removed from the tokens created before and after stemming. Stemming is performed by using porter stemmer. Lastly all words with length less than two characters are dropped. For every web page, a key value pair is created and saved in a dictionary, where key is the URL and the value is the cleaned tokens. The web crawler also creates a set of all words in the web pages crawled and retrieved.

2.2 Indexing

After the pages have been crawled and retrieved, these pages need to be stored in order to be fetched during querying process. The data collected in the crawling and scraping process are saved on the disk using python's pickle module. Multiple data structures are saved as pickles to be fetched during TF-IDF calculation and ranking. Using one of the pickle files, an inverted index is created which uses a complex hash map structure, where the key is each term in the entire collection and the value stores a list of documents the term occurs in, along with the term frequency in the respective document.

2.3 Ranking

The search engine scours the indexed documents to retrieve documents relevant to the user's query and information need. In this project ranking of documents is conducted using cosine similarity

and modified PageRank algorithm that considers the content of the web pages that match the query.

2.4 Spell Corrector

Misspelled words are very common in a search engine and a good search engine should work in spite of misspelled queries. To fix the issue of spelling mistakes I have implemented a spell corrector that pops up as a window when misspelled words are entered by the user. The window prompts the user with the question "did you mean (spelling corrected word)?" and requires a 'yes'/'no' response.

The spell checker corrects a misspelled word with 90% accuracy. The spell checker uses a probabilistic model to decide the correction for a misspelled word. [1] For examples the word 'lates' could be corrected to 'lattes', 'late', or 'latest'. These possibilities are called candidate characters c. The model finds the correction that is most probable given the misspelled word w. The formula for the calculation is shown below.

$$argmax_{cecandidates} P(c)P(w|c) \quad (1)$$

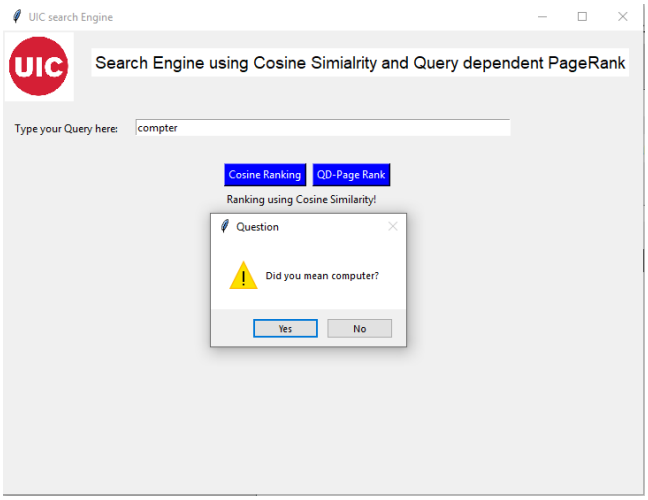

Figure 1: Screenshot of user interface illustrating spell-checker

2.5 User Interface

The searching is made easy for the end user with a graphical user interface. This GUI is built using Python's standard interface Tk GUI toolkit.The interface incorporates the colors of the university , Blue and Red and uses the University logo on the top left corner. The user can enter their query in a entry box. Two options are made available to retrieve the search results either using cosine similarity ranking or the modified PageRank via buttons. The results are displayed in a Listbox. If the user chooses to search using the alternative method a 'clear search' button is provided to clear the results and the query. More results can be added using 'load more results' button.

A Message box is used to prompt user for a yes/no if there is a spelling mistake in the query. This message box pops up as seen

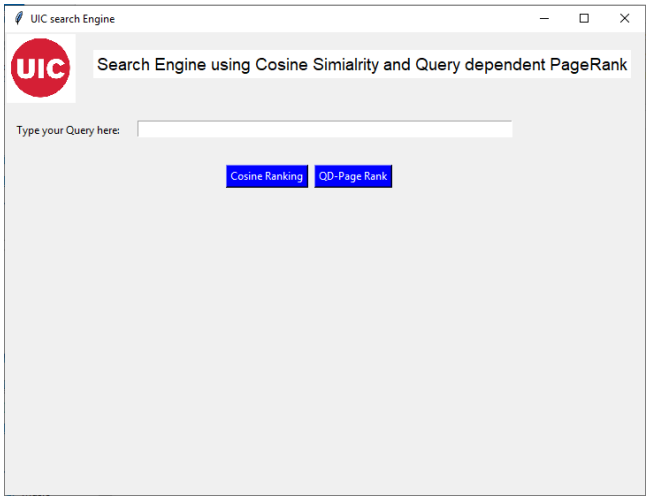

Figure 2: Screenshot of user interface before search

in Figure 1. If 'yes' button is clicked the search results shown are for the correctly spelled query. If 'no' button is clicked the search engine searches for the misspelled query.

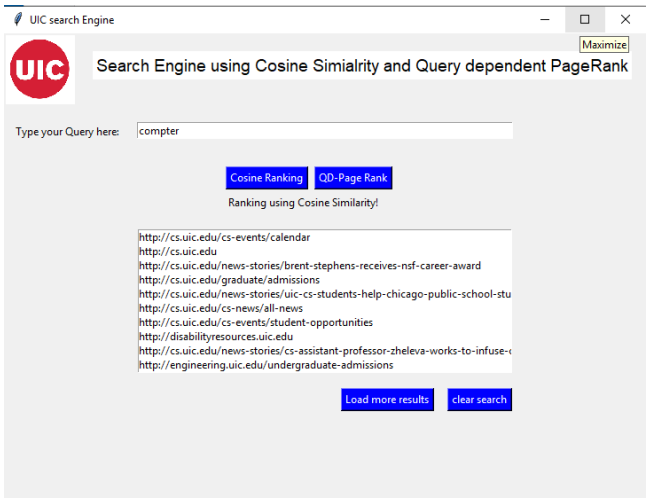

Figure 3: Screenshot of user interface after search

3 TERM WEIGHTS - MEASURING IMPORTANCE OF WORDS

After collecting the web pages and cleaning the textual data, the next task is to classify the importance of a web page (a sequence of words) by their topic. This classification if done by finding significant words in the web page documents. The intuition is that the word that appears more frequently is more significant, this is contrary to the truth. Most common words are stopwords and are removed from the corpus documents before the classification . Words that occur rarely in the collection but frequently in the document are considered more significant. This can be measured

using TF-IDF weighting. [5] TF-IDF is computed by multiplying the Term frequency (TF) with Inverse Document Frequency(IDF) given by the formula below.

$$w_{ij} = TF_{ij} IDF_i = TF_{ij} log_2 \frac{N}{DF_i} \tag{2}$$

Frequently-used parameters, or combinations of parameters, include:

- `Term Frequency`: Term frequency measures the number of occurrences in a document. If the corpus contains N documents and the term i occurs with the frequency of f_{ij} in document j, then the term frequency is calculated by:

$$TF_{ij} = \frac{f_{ij}}{max_k f_{kj}} \tag{3}$$

  The term frequency is normalized by dividing it by the total number of occurrences of the term that is most frequent in document j.
- `Inverse Document frequency`: For each term i the number of documents out of the total N documents that it appears in is given by DF_i. IDF for a term is calculated by :

$$IDF_i = log_2 \frac{N}{DF_i} \tag{4}$$

The term with high TF-IDF value characterize the topic of the document.

4 MEASURE OF SIMILARITY

In order to find which web pages are relevant to the user's query, similarity between the documents and the query needs to be calculated. In this project Cosine similarity is used. Here all the web pages and the query is represented by a vector. The similarity between two vectors in a multidimensional space is calculated by the cosine of the angle between them. The smaller the angle the higher the similarity. If the angle is 0 degrees, then the cosine similarity value is 1. The cosine similarity between two vectors is calculated by the following formula.

$$\cos(\theta) = \frac{\mathbf{d_j} \cdot \mathbf{q}}{\|\mathbf{d_j}\| \|\mathbf{q}\|} = \frac{\sum_{i=1}^{t} \mathbf{w}_{ij} \mathbf{w}_{iq}}{\sqrt{\sum_{i=1}^{t} (\mathbf{w}_{ij})^2} \sqrt{\sum_{i=1}^{t} (\mathbf{w}_{iq})^2}} \tag{5}$$

As seen by the formula, cosine similarity capture the angle and not the magnitude of vectors, in other words the size of the documents is not measured. A high cosine value indicates that the document is closer to the query. The implementation of cosine similarity in the project is done by parsing the inverted index structure created and equating in the above formula for each term in every document. Another structure is used to save this information where key is the document and value is the cosine value. By sorting the structure by value a ranked list of web pages is created. One advantage of cosine similarity is it is simple in implementation and the complexity is low.

4.1 Alternative Similarity measures

The other measures of similarity are Dice similarity and Jaccard similarity coefficient.Jaccard similarity performs well for Binary vectors. Dice Similarity calculates the intersection of the documents. According to the paper [6] cosine similarity gives the highest accuracy followed by Dice and Jaccard similarity for web retrieved documents. Hence cosine similarity was chosen for calculating the similarity in this project.

5 MAIN CHALLENGES

The search engine build in this project consists of multiple components with complicated architecture and implementation. Some of the challenges faced during the process of building this web search engine are:

- Quality Evaluation: Evaluating the quality of the search results was done manually in this project. This was a major problem as there is no user feedback or user behaviour data to evaluate the results. Evaluating a ranking algorithm efficiently can be done by collecting implicit feedback because relevance is subjective.
- Structuring the Program and Data: There are several components in the project and every component has some common inputs. Figuring out the structure of the project components and the flow of data among these components posed as the biggest challenge. Using python pickle to save this data made it convenient for retrieval by several components.
- Web Page Structure: The web pages that were crawled are neither well-structured nor free text which posed as a problem to collect all the textual data. Deciding which tags to be considered for scraping to get the page contents was difficult. Another problem faced here was all pages in the UIC domain do not follow the same convention in regards to HTML structure. To overcome this problem manual inspection of the page contents had to be performed for couple of web pages to generalize the scraping process to fit all the web pages.
- Duplicate links: While crawling the web within UIC domain a lot of links appear multiple times which is not added to the queue if it is already visited but the problem here is the number of times the repetition occurs is equal to the number of times the queue had to be checked. This affects the complexity of the crawling process even though duplicates are eliminated by using BFS algorithm.
- Time Complexity of the Project Components: Crawling the web, scraping, constructing inverted index and ranking the documents in two methods is a complicated task. After figuring out the structure to be used for the project it was difficult to implement them in the most efficient way possible. Implementation of cosine ranking and PageRank algorithm used multiple loops. The biggest challenge was crawling 3000 pages with limited computation overhead.
- Hyperlinks: Most UIC web pages have a many links from their home page to other web pages. For example from 'https://cs.uic.edu/graduate/' page we retrieve links such as '/cs-research/' which needed to be handled by appending to the parent link. Another problem with links was, a lot of links points to images , documents and other files which needed to be excluded.

6 INTELLIGENT COMPONENT

The primary goal of the search engine project is to implement an Intelligent component , in this case it is a modified version of the PageRank algorithm. The fundamental property of the PageRank algorithm is the structure of the world wide web which can be interpreted as a graph. Each web page constitutes a vertex in the graph and the hyperlinks between web pages is represented by the edges. The algorithm assigns a rank to each web page depending on number of inlinks during the crawl time. Higher the number of inlinks higher the rank.

Consider a web surfer that jumps from one web page to another which make a choice of the next web page based on a uniform probability. In order to over come the disadvantages of reaching dead ends or endless cycles, the surfer intermittently jumps to a random page with some probability β.

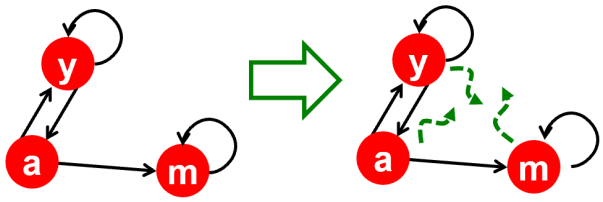

Figure 4: Image depicting how page rank overcomes dead ends

Let **N** represent the number of nodes. Given a node i, let F_i represent the nodes pointed by node i, B_i represent the nodes that are directed towards node i. In case of a node that has no outlinks, we add an edges from the node to every other node in the graph. These edges are added to uniformly redistribute the ranks that were lost. [3] After surfing sufficient number of nodes, on an average, the probability with which the surfer is on a page j at any given instance is given by the formula,

$$P(j) = \frac{(1-\beta)}{N} + \beta \sum_{i \epsilon B_j} \frac{P(i)}{|F_i|} \tag{6}$$

The PageRank score for the node j is also defined by this probability. Since the equation (1) is recursive, it is computed iteratively until P(j) converges. Initial distribution of P(j) is uniform in most cases. Consider a transition matrix Z, page rank is equivalent to primary Eigen vector of this matrix.

$$Z = (1-\beta)\left[\frac{1}{N}\right]_{N*N} + \beta M, \text{ with } M_{ji} = \begin{cases} \frac{1}{|F_i|}, & \text{if an edge from} \\ & \text{i to j exists} \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

$x_j^t = P(j)$ for a given an iteration t, The iteration $t+1$ can be computed using the formula $x^{t+1} = Z(x^t)$. Once the equation converges, we notice that $x^{T+1} = x^T$ or $x^T = Zx^T$. This also means that x^T is an eigen vector of Z. Also, since the columns of Z are normalized, x has an eigen value equal to 1. The one disadvantage of this model is that it does not rank the pages according to the query terms, this can be improved by a modification.

6.1 Modified PageRank Algorithm

In this project, the proposed intelligent surfer makes probabilistic hops from page to page depending on the page content and the query the user is looking for. The probability distribution over these pages would be given by the formula:

$$P_q(j) = (1-\beta)P'_q(j) + \beta \sum_{i \epsilon B_j} P_q(i) P_q(i \rightarrow j) \tag{8}$$

Where $P_q(i \rightarrow j)$ is the probability that the surfer moves from page j from the page I and is looking for the query q. $P_q^{'(j)}$ represents where the surfer jumps when not following links. $P_q(j)$ is the resulting probability distribution over sufficient pages and is equivalent to the query-dependent PageRank score.

Similar to PageRank, Query-dependent PageRank [3] is computed by iterative evaluation of the above equation for some initial distribution. It is also equivalent to the primary eigen vector of the transition matrix Z_q, where Z_q is given by the formula,

$$Z_{q_{ji}} = (1-\beta)P'_q(j) + \beta \sum_{i \epsilon B_j} P_q(i \rightarrow j) \tag{9}$$

Even though $P_q(i \rightarrow j)$ and $P'_q(j)$ are arbitrary distributions, our primary focus would be on the case where both of these probabilities would be derived from $R_q(j)$, which is a measure of relevance of the page j to query q.

$$P'_q(j) = \frac{R_q(j)}{\sum_{k \epsilon W} R_q(k)} \tag{10}$$

$$P_q(i \rightarrow j) = \frac{R_q(j)}{\sum_{k \epsilon F_i} R_q(k)} \tag{11}$$

This can be otherwise stated as, when choosing among multiple out links, the surfer tends to chose the links that is more relevant to the query q (according to R_q). As in the case of PageRank, when the surfer is at a node which has no relevance to the query, no outlinks or is a dead end, we add edges to every other node in the graph. Thus, on such a page, the surfer chooses a new page that is according to the distribution $P'_q(j)$.

In the case of multiple word query, $Q = q_1, q_2, \ldots q_n$, The surfer makes the selection of q depending on the probability distribution, $P(q)$ and uses that term to guide its behavior (according to equation 8) for a larger number of steps. The next query terms behavior is determined according to the distribution and this process repeats for all terms in the query. The total distribution over all the visited pages is determined by the formula

$$QD - PageRank_Q \equiv P_Q(j) = \sum_{q \epsilon Q} P(q) P_q(j) \tag{12}$$

7 EVALUATION

The Evaluation of performance of the web search engine built for this project is a tricky process. Effectiveness of a search engine depends on the relevancy of the search results to the query. Relevance is subjective and cognitive. For evaluating the performance of this project manual evaluation is done by calculating the precision of

the top ten search results. Precision is calculated by the following formula.

$$Precision = \frac{Number of Relevant Documents Retrieved}{Total Number of Documents Retrieved} \quad (13)$$

For the purpose of evaluating this project the denominator term of the above equation is 10 and the precision is compare with the Search engine that uses cosine similarity ranking for the same set of query. Below are the results for two search engines for five queries:

- Query 1: Engineering Major

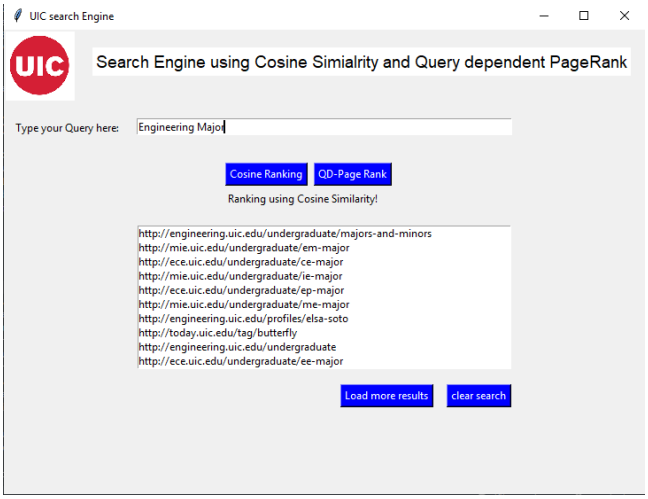

Figure 5: Results for query: Engineering major using Cosine Ranking

The precision of the above results is 0.7 that is, seven out of the top ten results are relevant.

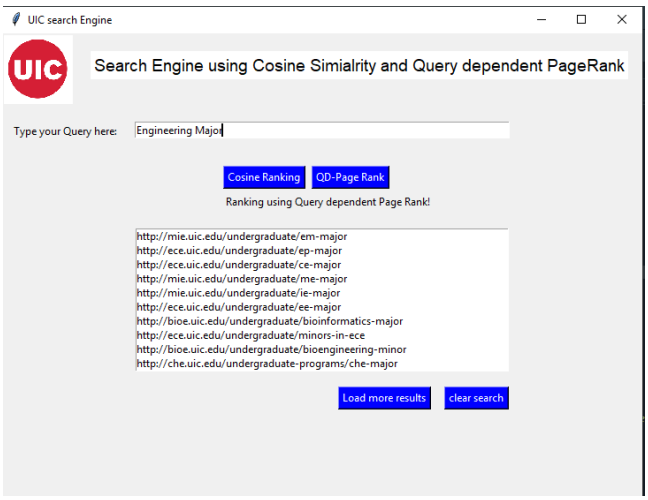

Figure 6: Results for query: Engineering major using Modified PageRank Ranking

All the web pages retrieved by the PageRank algorithm are relevant as it contains information about engineering majors.The precision is 1.0. In this case, the PageRank algorithm clearly outperforms the cosine ranking.

- Query 2: Monarch Butterfly
  Testing the effectiveness of searching an important article which has been linked in many UIC web pages.

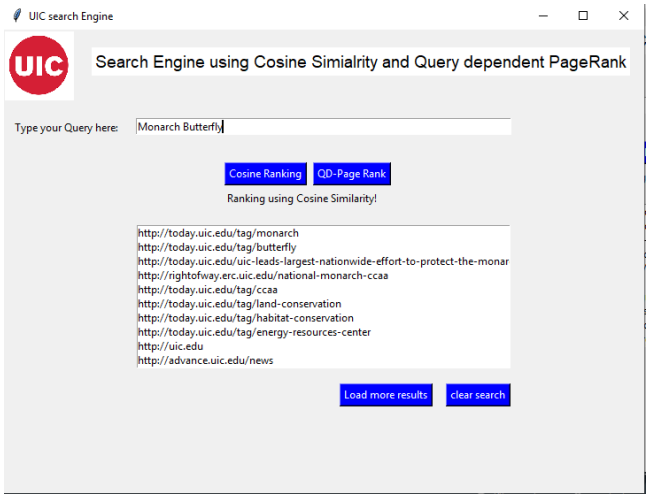

Figure 7: Results for query: Monarch Butterfly using Cosine Ranking

The precision for Cosine Ranking is 0.8 for the above query.

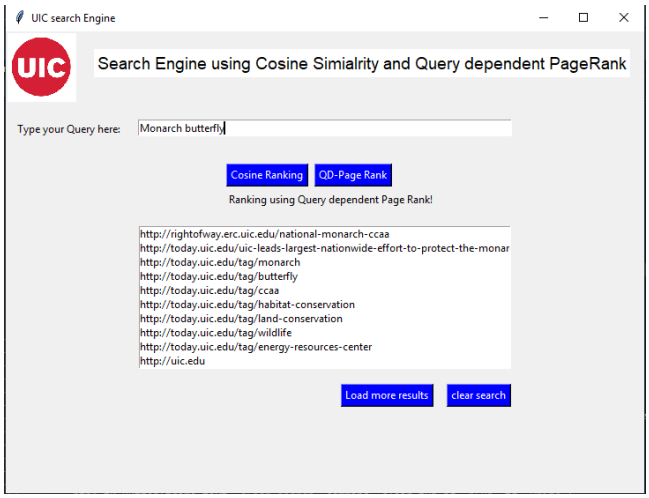

Figure 8: Results for query: Monarch Butterfly using Modified PageRank Ranking

The Difference between both the algorithms is clearly visible in this query results. The Precision for PageRank algorithm is 0.9. Although the difference in precision is only 0.1, the main difference is the link that appears on the top for PageRank result. This link is the most popular and referred by other web pages.

- Query 3: Data Science Now searching for a topical query for which the expected results are professor's work and research, course catalogs, articles etc.

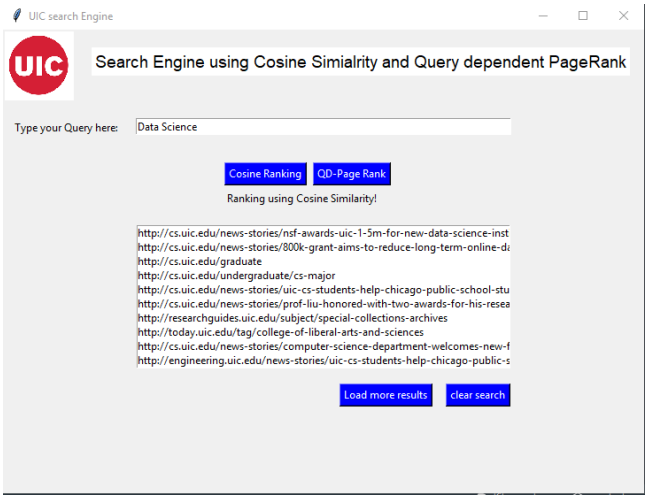

Figure 9: Results for query: Data Science using Cosine Ranking

The precision for Cosine Ranking is 0.6 for the above query. Whereas, the Precision for modified PageRank is 0.8. Even in this case the latter performs better.

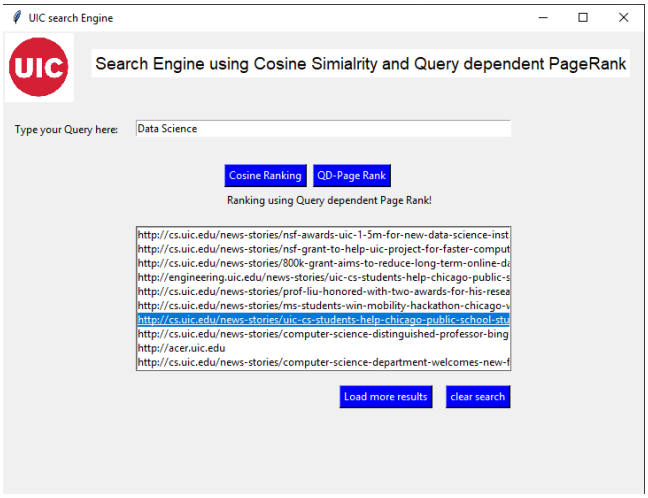

Figure 10: Results for query: Data Science using Modified PageRank Ranking

- Query 4: Transfer
  The above query can include undergraduate transfer, graduate school transfers, transfer between different schools. The search results are shown below.
  The precision for Cosine Ranking is 0.8 for the above query. Whereas the Precision for modified PageRank is 1.0. There

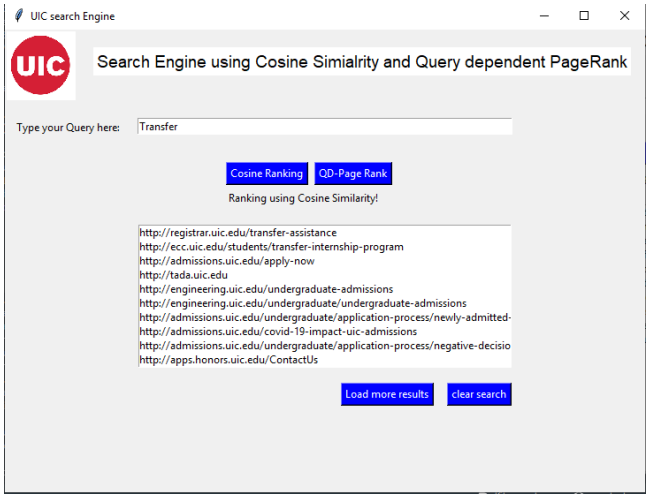

Figure 11: Results for query: Transfer using Cosine Ranking

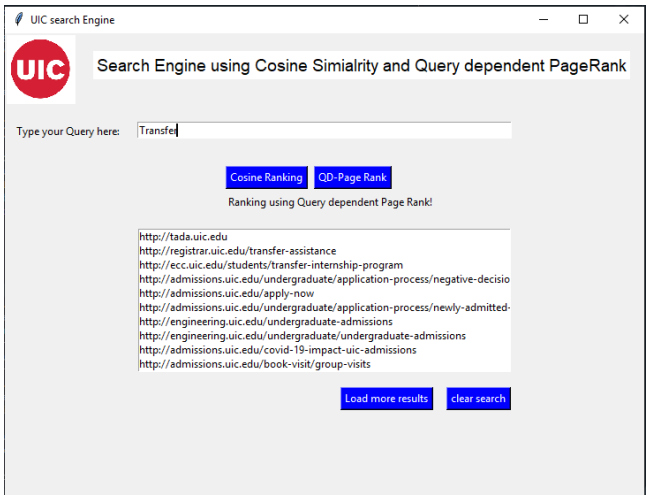

Figure 12: Results for query: Transfer using Modified PageRank Ranking

seems to be a trend where the precision of modified PageRank Algorithm is better than the simple cosine Ranking. Similar to the Monarch Butterfly query, the results for PageRank shows the most popular link on the top highlighting the heuristic behind PageRank algorithm.

- Query 5: Student Employment The results for the query include on-campus options as well as full-time employment and internship opportunities.
  The precision for Cosine Ranking is 0.6 for the above query. Whereas the Precision for modified PageRank is 0.9 . AS observed above the trend continues where the Modified PageRank algorithm performs better than the Cosine ranking.

Comparing the results for five queries, a trend is discovered where the Modified PageRank algorithm tends to work better than the cosine similarity ranking, not only in the precision but also in the

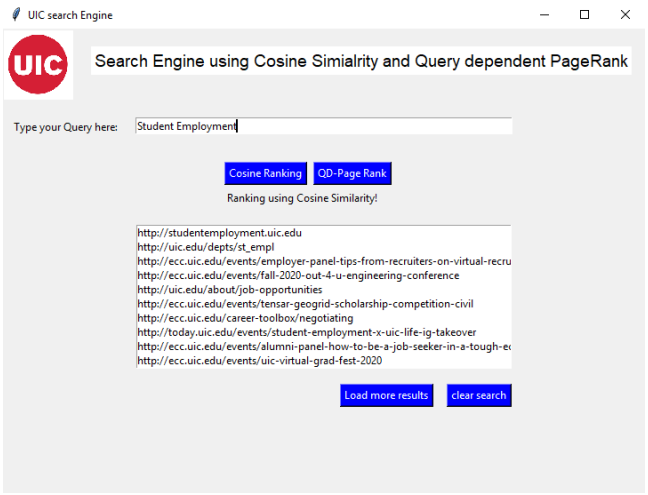

Figure 13: Results for query: Student Employment using Cosine Ranking

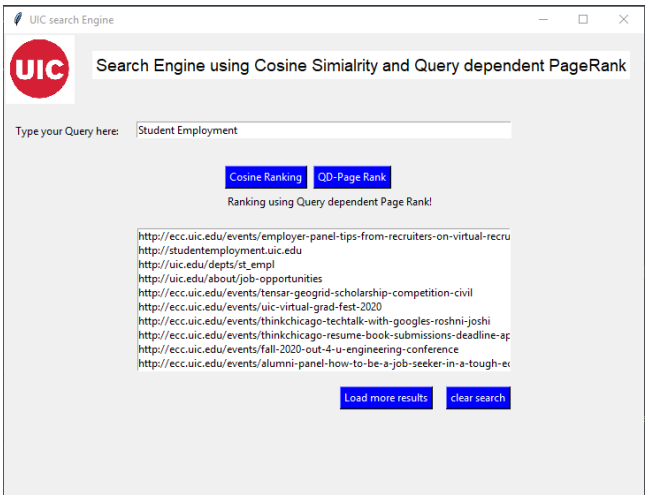

Figure 14: Results for query: Student Employment using Modified PageRank Ranking

order of ranking.Hence, the Intelligent component outperforms the traditional ranking in almost all cases. The graph in figure 15 illustrates this trend.

8 ERROR ANALYSIS

From the plot in Figure 15 it is evident that the intelligent PageRank algorithm combines the link structure and the content of web pages to perform efficiently in comparison to the traditional cosine similarity ranking. Like Wikipedia pops up on the top of a Google search for a celebrity, the PageRank algorithm also produces the most popular link in the graph on the top of the results. When a larger query set is considered , it is found that the precision drops slightly for single word queries. Another observation in the search results is that the query has to be framed carefully with the terms in

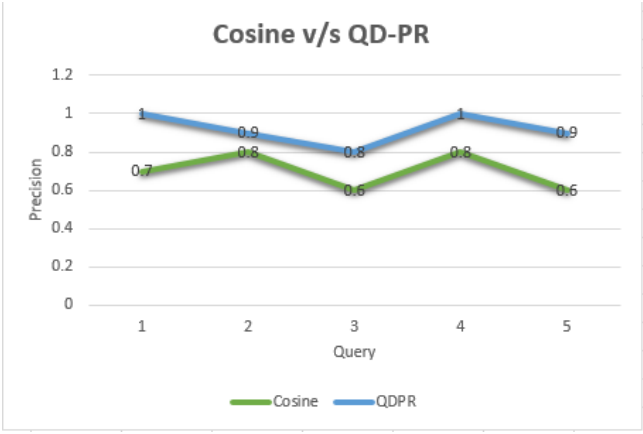

Figure 15: Plot of precision values for all queries

the web pages. If a search for a synonym of the query is prompted the results change. Lastly, the spellcheck does not consider some of the nouns like the names of Professors. When search is performed with a uncommon name, the spell checker window pops up. Some terms like Sparky, Campus, and University related terms are not available in the dictionary.

9 RELATED WORK

The PageRank Algorithm is clearly very efficient and popular. It has been improvised over the years. The PageRank algorithm is susceptible to link spams. Some methods of improving PageRank to avoid such spams are using TrustRank and Truncated PageRank. Another algorithm where dangling nodes are improvised is by Pro-PageRank where instead of weighting the links equally it considers the difference of weights.

10 FUTURE WORK

The search engine built for this project is semi-dynamic where 3000 web pages are scrapped and saved in the form of dictionaries. The results are calculated at run time for previously saved and stored web pages. The search engine can perform at run time but it makes a compromise on time and speed. This can be fixed in the future by using multiple threads to crawl the web and multiple processes can be used to scrape the web pages making the search engine run fast and meet the user's information need in dynamic time. To make the evaluation of the search engine better, user's implicit feedback can be collected. This will help in understanding what is truly relevant or not. N-grams can be used to make the search results more precise. The Queries can be expanded for synonyms to give more relevant documents within the context of the query. The spell checker can refer a more domain specific dictionary which for this case could include international names, and domain specific words used in articles and research papers.

REFERENCES

[1] How to Write a Spelling Corrector: http://norvig.com/spell-correct.html.
[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Seventh International World-Wide Web Conference (WWW 1998)*, 1998.

[3] David A. Cohn and Thomas Hofmann. The missing link - a probabilistic model of document content and hypertext connectivity. In *NIPS*, 2000.
[4] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2 edition, 2014.
[5] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
[6] Vikas Thada and Vivek Jaglan. Comparison of jaccard, dice, cosine similarity coefficient to find best fitness value for web retrieved documents using genetic algorithm. 2013.