UIC Web Search Engine

With Query dependent Page Rank

Santhosh Mani
Computer Science Department
University of Illinois at Chicago
Chicago, Illinois
smani6@uic.edu

ABSTRACT

Search engines index tens to hundreds of millions of web pages involving a comparable number of distinct terms. They answer tens of millions of queries every day. In this report, we describe the project consisted in building a web search engine for the UIC domain from scratch. The software was built modularly, starting from web crawling, going through pages preprocessing, indexing and finally adding a Graphical User Interface.
The software uses query-dependent page rank algorithm to find the import nodes in the word graph. Extensive evaluation on a number of different topics in a domain demonstrates that the proposed approach significantly improves on query-dependent PageRank, outperforms the current implementation of naïve page rank algorithm.

Keywords

Search Engines; Information Retrieval; PageRank

1. INTRODUCTION

The search engine was built for the "University of Illinois at Chicago" (UIC) domain extracts information from all the URL's of the UIC. The collected data includes texts, phrases from the valid HTML tags, URL's pointed by and to the current URL. The related data is crawled and indexed and stored in the filesystem. 5000 web pages were crawled using BFS Algorithm and subsequently TFIDF vectorization for the contents of each URL. So when users perform a query in order to get some data or information, the related query is transferred to the search engine index and optimization algorithms are performed to retrieve the relevant documents. Two different algorithms are used for getting results on query. First one is TFIDF ranking which uses cosine similarity to determine the most relevant URL for a given query. In the second method Query dependent page rank is used. Here the teleportation of traditional page rank is modified based on user query. Details of both the implementation are discussed in the following sections.

2. CRAWLER

The web crawler uses BFS algorithm and it starts from the UIC CS subdomain: https://www.cs.uic.edu/ which is the root node. From here, it adds all other hyperlinks in this webpage to a queue and the webpage is removed. The next page in the queue is chosen and the process is repeated until the queue is empty or a threshold is reached. Initially all the weblinks up to a threshold are crawled and using each of the weblink the content of the page is preprocessed. Following links (".docx", ".doc", ".avi", ".mp4", ".jpg", ".jpeg", ".png", ".gif", ".pdf", ".gz", ".rar", ".tar", ".tgz", ".zip", ".exe", ".js", ".css", ".ppt") ending with theses suffix were excluded from consideration. Each URL time out was set to 5 seconds. Accessing text from pdf pages was optional and in order to reduce crawling time pdf pages were ignored. Crawler script return the result in dictionary format, with keys as URLS and a tuple consisting of URL, stemmed tokens and list of outgoing links for each of the key. The keys consisting of URLs were stripped from http or https as it does not add any value in indexing. Regarding the cleaning of the content of each webpage, nltk library was used to tokenize and porter stemmer was used stem each of these tokens.

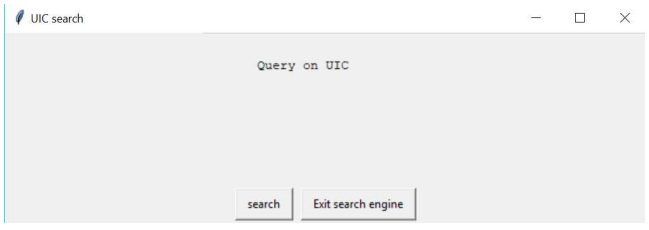
3. TFIDF INDEXING

I built an inverted index from the entire vocabulary from the Spider and performed tokenization, stop word removal, porter stemming while constructing the inverted index. The data structure indexes the word as its key with all its occurrences in each of the document along with its frequency in that document (Term Frequency). I also store the document frequency along with the inverted index. I then calculated the TF-IDF score of each word in that document and store it as another vector model. Term frequency and document length for each URL was stored using 1-D dictionary. IDF was stored in a 2-D dictionary.

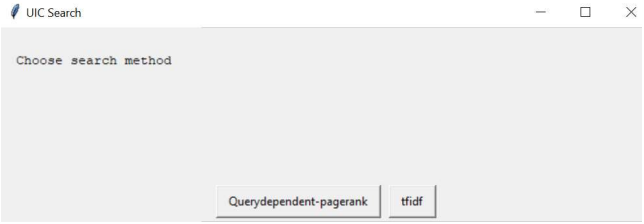
4. GRAPHICAL USER INTERFACE

Graphical user interface was implemented using Python library EasyGUI.This GUI includes basic functionalities for displaying results.

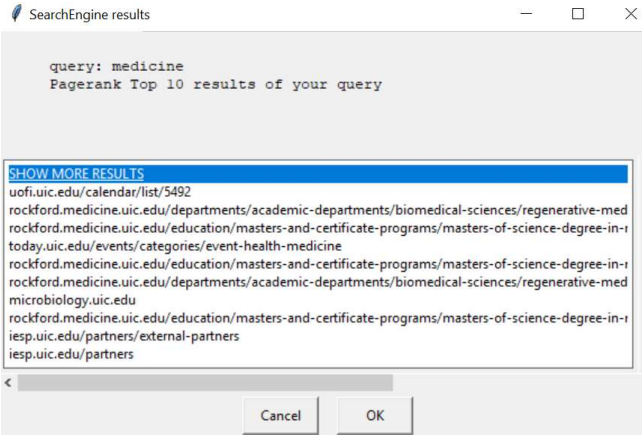
Initial state of the application is choice of either to start or quit the application and this functional module is used after every query results, so that an user can opt to quit as and when required.Below is the snapshot of the application.



An option to choose the type of algorithm to use while getting results is implemented. Below is the snapshot of the process.



Search more option in the GUI is given with the list of results.When the choice of more result is opted then list of top 20 results are displayed.



5. WEIGHTING and SIMILARITY

5.1 Weighting scheme

TF-IDF weighting scheme was used for the words in documents since most of the URLs in the UIC domain has lengthy documents with high use of common words such as UIC, edu, department, students etc. It's better to use TFIDF instead of TF alone.

TFIDF formula is given by:

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

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

5.2 Similarity measure

I use the inverted index constructed above to find the limited set of documents by computing cosine similarity of each indexed document as query words are processed one by one. I accumulate a total score for each retrieved document, store retrieved documents in a dictionary, where the url is the key and the accumulated score is the value. We then Sort the dictionary including the retrieved documents based on the value of cosine similarity and return the documents in descending order by its relevance.
Cosine similarity is given by :

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

6. INTELLIGENT COMPONENT

PageRank assigns to a page a score proportional to the number of times a random surfer would visit that page, if it surfed indefinitely from page to page, following all out links from a page with equal probability. To improve Page-Rank by using a more intelligent surfer, one that is guided by a probabilistic model of the relevance of a page to a query. Efficient execution of our algorithm at query time is made possible by pre-computing at crawl time (and thus once for all queries) the necessary terms

6.1 Page Rank: The Random surfer

Here a web surfer who jumps from web page to web page, choosing with uniform probability which link to follow at each step. In order to reduce the effect of dead-ends or endless cycles the surfer will occasionally jump to a random page with some small probability d, or when on a page with no out-links

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 graph. In this way, rank which is lost due to pages with no out links is redistributed uniformly to all pages. If averaged over a enough steps, the probability the surfer is on page j at some point in time is given by the formula:

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

6.2 Query Dependent Page rank:

A problem common to both PageRank and HITS is topic drift. Because they give the same weight to all edges, the pages with the most inlinks in the network being considered (either at crawl or query time) tend to dominate.To give importance to query , Query dependent page is implemented here.

Query dependent page rank is a more intelligent surfer, who probabilistically hops from page to page, depending on the content of the pages and the query terms the surfer is looking for. The resulting probability distribution over pages is:

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

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-PageRank(j) ≡Pq(j)).

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 depend on calculating Rq(j), a measure of relevance of page j to query q. The choice of relevance function Rq(j) is arbitrary. In the simplest case, Rq(j)=R is independent of the query term and the document, and QD-PageRank reduces to Page-Rank. One simple content-dependent function could be Rq(j)=1 if the term q appears on page j, and 0 otherwise.

If Rq(j)=0 for some document j, the directed surfer will never arrive at that page. In this case, we know QD-PageRank q(j)=0, and thus when calculating QD-PageRank q, we need only consider the subset of nodes for which Rq(j)>0. We add the reasonable constraint that Rq(j)=0 if term q does not appear in document j, which is common for most information retrieval relevance metrics, such as TFIDF. The computation for term q then only needs to consider dq documents. Because it is proportional to the number of documents in the graph, the computation of QD-PageRank q for all q in W will require O(S) time, a factor of S/N times the computation of the query independent PageRank alone.

7. EXPERIMENTS AND RESULTS

The non-intelligent method outputs the list of pages based on TFIDF of each page according to the query. The intelligent method performs better; at least in most of the queries, as it takes both context of the query and importance of page.table 1 shows the precision for top results using Query dependent page rank and TFIDF.

Fig1:Query dependent PR search on: "Job fair"

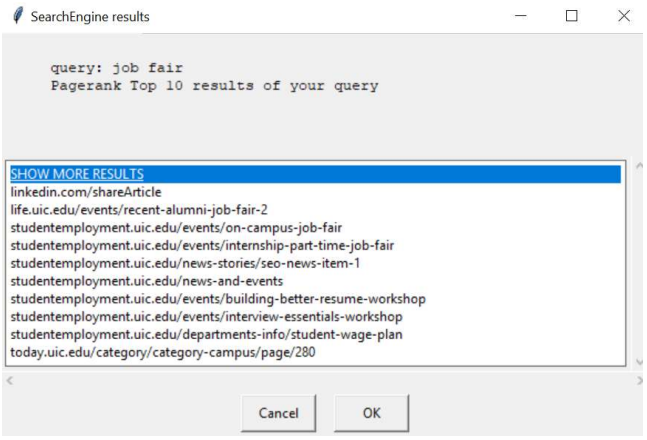
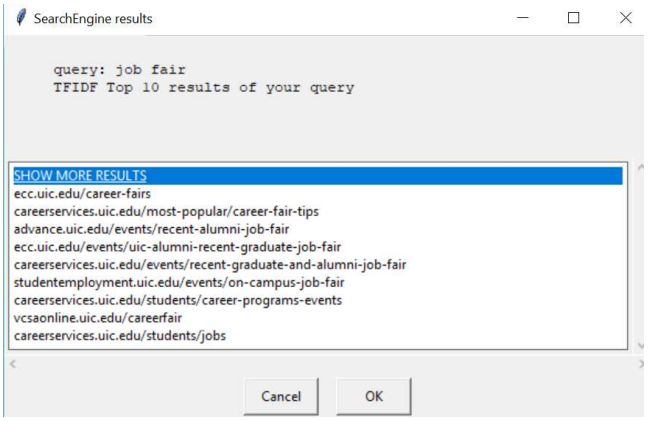


Fig2:TFIDF search on : "Job fair"



Comparing intelligent and non-intelligent we can see the results in intelligent search yields more sub domains of careers ,whereas non-intelligent search yields results where the term frequency in a domain is more.

Fig3:Query dependent PR on :"Artificial Intelligence"

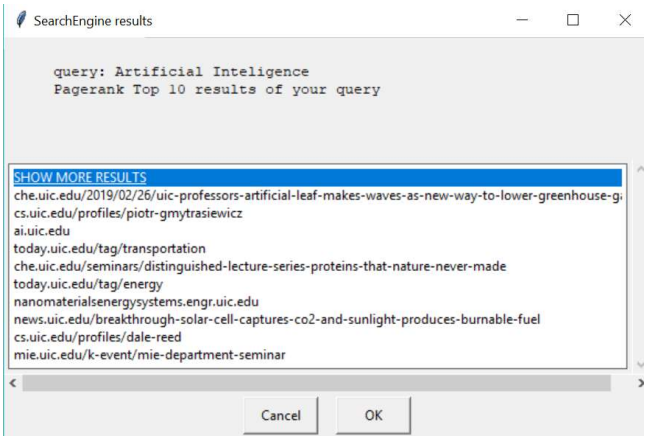
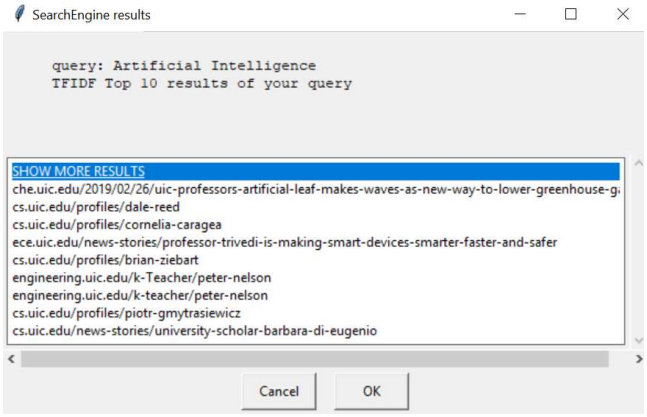


Fig4:TFIDF Query on :“Artificial Intelligence”



Similarly query based search here fetches articles more related to the query “Artificial Intelligence ” compared to non-intelligent technique.

Fig5:Query Dependent PR on “Graduate commencement”

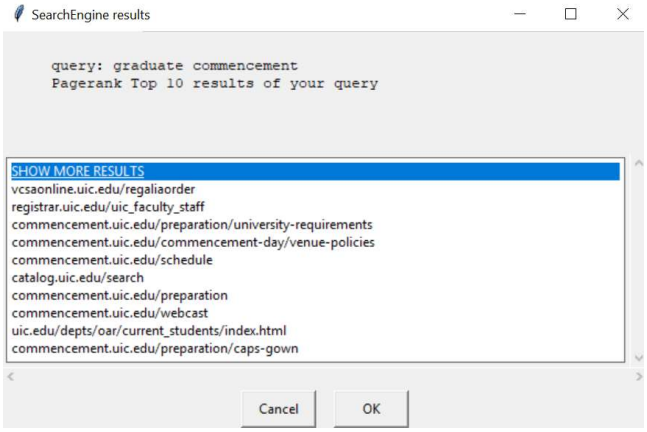
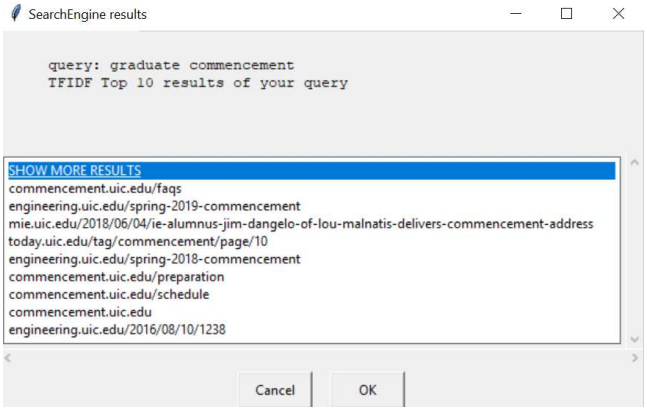


Fig6: TFIDF query on “Graduate commencement”



The results of “Graduate ceremony” is comparable in both the cases. Here most of the results are relevant to the search term.

Fig7:Query dependent PR search on “soccer clubs”

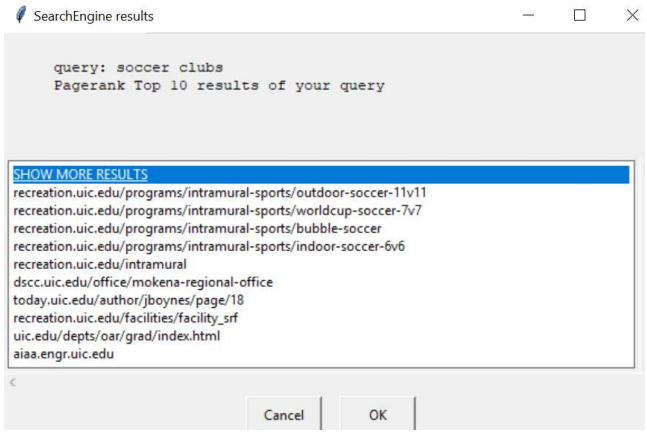


Fig8: TFIDF search on “soccer clubs”

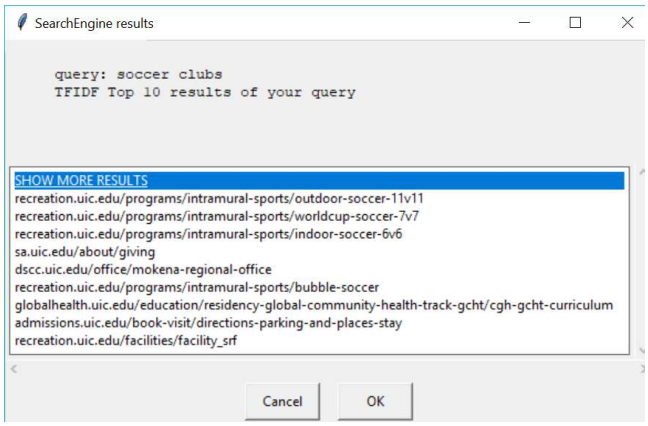


Fig9: query dependent PR on “hackathon”

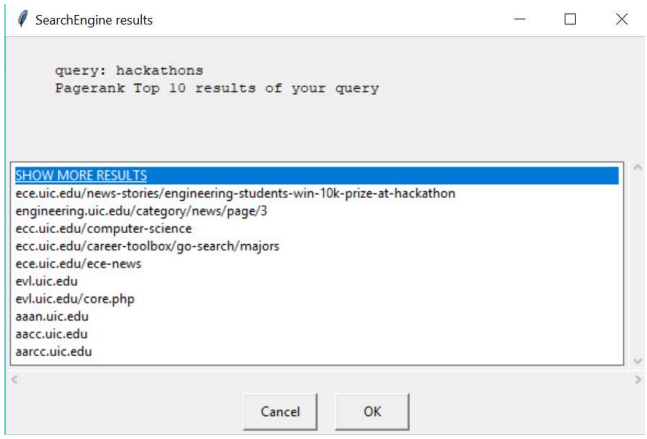
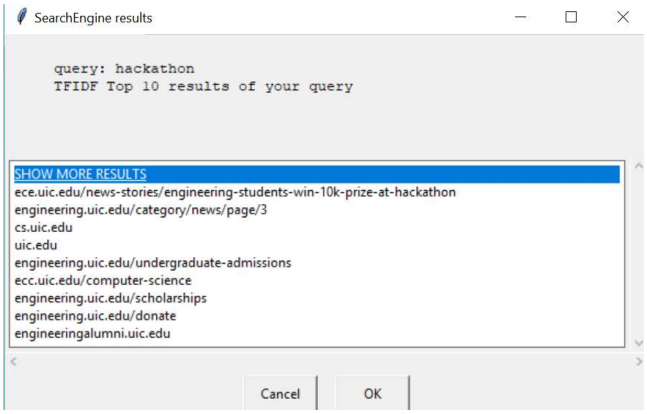


Fig10: TFIDF search on “hackathon”



TFIDF search of the query ”hackathon” is not good as the result yielded here are more generic results compared to intelligent search.

Table1: Precision values for Top 10 results

No.	Search	Precision	
		QD-Pagerank	TFIDF
1	“Job fair”	0.8	0.7
2	“Artificial Intelligence”	0.9	0.7
3	“Graduate Commencement”	1	0.9
4	“soccer cubs”	0.7	0.4
5	“hackathon”	0.5	0.2

8. ERROR ANALYSIS

From the results, I observed that the search engine performs much better with a single word in the query. Though this did not apply for all the queries, I have found few which failed to work better with large query size when performing an query dependent page rank search.

9. CHALLENGES

The main challenge for search engine implementation is the choice of intelligent component. As the querying using cosine similarity is a very straight forward approach to retrieve the relevant documents, many algorithms have been developed to improve the efficiency of the search by ranking the relevant documents. I have initially tried with weighted page rank model but ended up shelving it as I could not notice much of an improvement in comparison with a naïve page rank method. Later implemented query-based page rank as it I thought since domain is fixed most of the queries are correlated.

Other Challenging part was web scraping UIC domain.The web scraped pages had lot black listed format such as ".docx", ".doc", ".avi", ".mp4", ".jpg", ".jpeg", ".png", ".gif", ".pdf", ".gz", ".rar", ".tar", ".tgz", ".zip", ".exe", ".js", ".css", ".ppt". Finding these formats consumed lot of time as it would require running crawler script multiple times.

In the GUI, I faced issues in setting up “SHOW MORE RESULTS” Options as the default options for any choice menu in easygui library was “OK” or “Cancel”. So in order to create a clickable option, appended “SHOW MORE RESULTS” to choice list and it was popped once user clicks on.

10. RELATED WORKS

Over the years improvising Page rank algorithm is steadily growing. Some of the key issues such dangling nodes are considered for improvising. Similar to Query dependent page rank,pro-pagerank algorithm is introduced here.The algorithm which does not simply weight the links equally but take the differences of weights into account. It can effectively solve the following problems: outlink pages with high PageRank values and users urgency for needed pages. According to Simulated experimental data, the improved algorithm, Pro-PageRank algorithm, compared with the traditional one, has more accuracy and further improves the quality of page rank to meet authority requirements of the page calculation results.

11. FUTURE WORK

The overall performance of the proposed search engine is decent enough. Currently the search engine works gives best result for single query term and the success of finding relevant URLs for sentence query is not good. I would prefer bigrams and trigrams for TFIDF vectorization. Crawling of web pages can be improved by using multiple threads. Also implementing pseudo relevance feedback with query dependent page rank and spelling correction for English words can improve results.

12. REFERENCES

[1] G. Salton and M. J. McGill (1983). Introduction to Modern Information Retrieval. McGraw-Hill, New York, NY.

[2] T. Haveliwala (1999). Efficient computation of PageRank. Technical report, Stanford University, Stanford, CA.

[3] D. Rafiei and A. Mendelzon (2000). What is this page known for? Computing web page reputations. Proceedings of the Ninth International World Wide Web Conference. Tavel, P. 2007. *Modeling and Simulation Design*. AK Peters Ltd., Natick, MA.

[4] K. Dai, "PageRank Lecture Note," 22 June 2009. [Online]. Available: http://www.ccs.neu.edu/home/daikeshi/notes/PageRank.pdf

[5] T. Qin, T.-Y. Liu, J. Xu and H. Li, "LETOR: A Benchmark Collection for Research on Learning to Rank for Information Retrieval," Information Retrieval, vol. 13, no. 4, pp. 346 - 374, August 2010.

[6] L. Page, S. Brin, R. Motwani, and T. Winograd (1998). The PageRank citation ranking: Bringing order to the web. Technical report, Stanford University, Stanford, CA.