Intelligent Search Engine

With Page Rank Algorithm

Bhargav Arisetty
Computer Science
University of Illinois at Chicago
Chicago Illinois United States
marise2@uic.edu

ABSTRACT

Searching our website can be a challenging task due to the large volume of information it contains. By default, the website searches for records which contain any of the words you type. This can yield a surprising number of results which only amplifies with the more words you type.

Information about a subject is usually searched on the first page of search engines. After examining first 5 result pages, other remaining pages are not evaluated by users. Because of this, it is important to move a web page to top lists of search engines in order to introduce it better. In order to achieve this, the search engine optimization must be used by the developers. Because, it is possible to move a web page to the first page of a search engine by using only some necessary optimization rules.

Intelligent Search Engine is a software which is specifically 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. 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. The crawler phase includes the BFS technique to crawl the links which are confined/restricted to specific domains. Then, an inverted index is constructed based on the indexed information in the spider. TF-IDF structure is computed using the inverted index. A link analysis is performed by constructing a graph using the in/out links of the URL's and computes page rank scores for each URL. For a given query, the engine retrieves the relevant documents based on the cosine similarity of the query and document using the TF-IDF vectors. In our intelligent search engine, we combine the TF-IDF vector model with an extended page rank algorithm to improve the search by retrieving the most popular pages among the retrieved relevant documents and are displayed as ranked list of URLs' to the user. This would enhance the user experience with the search engine as the result yields to be in the best interest with the user expectations. Details about the algorithms and implementation are studied in detail below.

WEB CRAWLER

The web crawler job is to gather pages from the web and index them to support the search engine. They are one of the main components of web search engines, systems that assemble a corpus of web pages, index them, and allow users to issue queries against the index and find the web pages that match the queries. In this software, we use BFS spidering algorithm to crawl and index the URL's. While indexing the URL, we use URL filtering to restrict the undesired domains from being indexed. (i.e In this case, we restrict our spider to the "UIC" domain.) I also used Link Canonicalization by normalizing the urls by removing the slashes. All the urls from the current page are indexed using the anchor tags while the content is obtained from various tags like <SPAN>, <TEXT>, <P>, <H> after analyzing the general structed of the HTML page for that domain. The page duplication is avoided by maintaining the set of visited URL's and we do not spider the page which is already spidered. Once the crawler has successfully spidered the required URL's(Until the stopping criterion. 3000 url's in this case). The crawler is implemented using multi-threaded model for the spider to run faster. I have used 100 threads for this project.

Vector Space Model:

We build an inverted index from the entire vocabulary from the spider. We perform tokenization , stopword removal, porter stemming while constructing the inverted index. The data structure indexes the word as its key with all its occurances in each of the document along with its frequency in that document(Term Frequency). We also store the document frequency along with the inverted index. Complexity of creating vector and indexing a document of n tokens is O(n). So indexing m such documents is O(m n). Computing token IDFs can be done during the same first pass Computing vector lengths is also O(m n). Complete process is O(m n), which is also the complexity of just reading in the corpus. We then calculate the TF-IDF score of each word in that document and store it as another vector model.

Querying using Cosine Similarity

We 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. We

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 of their relevance

Cosine Similarity

$$cos(q,d) = \frac{\sum_{t \in q \cap d} q_t d_t}{\sqrt{\sum_{t \in V} q_t^2} * \sqrt{\sum_{t \in V} d_t^2}}$$

q_t is the tf-idf weight of term t in the query
d_t is the tf-idf weight of term t in the document

INTELLIGENT COMPONENT

The intelligent component used in this search engine is the page rank algorithm which is proved to be the successful model used by google. The link analysis is actually performed using an extended page rank algorithm to compare the results with regular query using cosine similarity. We iteratively run the page rank algorithm based on the in links of a page until it converges. As we have finite set of finite set of documents (3000), the standard formulae to compute the page rank algorithm is used and run for 10 iterations for the convergence. The intelligent component is integrated with the cosine similarity model by taking the harmonic mean of both the scores(i.e cosine similarity and page rank score of that URL)This has shown decent results when compared with the results in obtained from google.

$$s(v_i) = \alpha \sum_{v_j \in Adj(v_i)} \frac{w_{ji}}{\sum_{v_k \in Adj(v_j)} w_{jk}} s(v_j) + (1-\alpha)\tilde{p}_i,$$

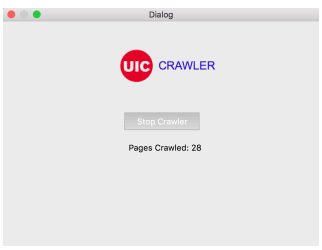
CHALLANGES

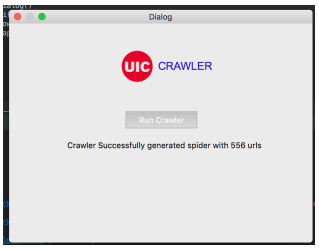
The main challenge for any search engine development is the choice of intelligent component. As the qurying using cosine similarity is a very straight forward approach to retrieve the relevant documents, many algorithms have been developed to improve the efficiency of the search by ranking the relevant documents. I have initially tried with relevance feedback but ended up shelving it I did not notice much of an improvement in comparison with a conventional TF-IDF method. I thought, the algorithm would require more data to perform better in the real world but as my URL's size is restricted to 3000 URL's for a single domain, it did not perform exceptionally well. I started of by implementing the extended page rank algorithm for the link analysis and found the results improving very quickly. Tuning the parameters like damping factor in the above equation and setting the right number of iterations for the algorithm to converge is a challenging factor while performing the link analysis. Even before this, I had to analyse the HTML for the UIC domain for the crawler to run smoothly. I had to check which tags yield more appropriate content and retrieve the text from those tags. The crawler was initially failing due to many reasons like redirects, SSL errors etc. I had to apply many conditions in order to avoid duplication like checking the URL scheme. Even after bypassing these hurdles, the crawler was running very slow and ultimately had to implement multithreading and managing the synchronization between the variables(Queue, Spider). Combining the intelligent component with the existing TF-IDF model has been challenging. I initially tried many methods like custom sorting the URLs obtained from cosine similarity using the page rank scores, later I tried multiplying the scores obtained by each weighting scheme. I finally found that harmonic mean between obtained using the cossim and page rank yielded better results.

SOFTWARE

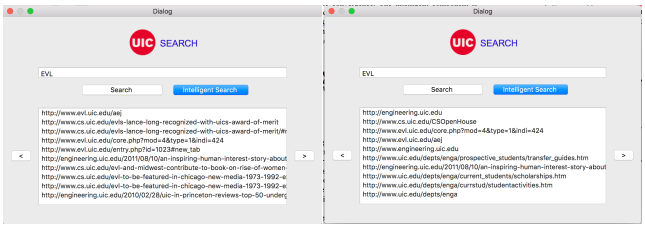
The software is developed in python. All the required packages that are used to run the software is detailed in the README.txt file. The search engine has 2 apps. One is CRAWLER_APP.py that implements the multi threaded crawler for the UIC domain. It has an option for the to start and stop the crawler, else the crawler will run for 3000 urls by default. The second one is the SEARCH_ENGINE_APP.py that performs the actual construction of page rank and tf-idf model. Once the application boots up, the user can enter the query to see the results on the window. The user has an option to navigate forth and back for more urls. Each page will show 10 pages. The application has two search options. One to do a regular search that implements a search based on the cosine similarity and the othe oprtion is the 'Intelligent search' which performs the search based on the page rank score and cossim (harmonic mean).

CRAWLER APP:





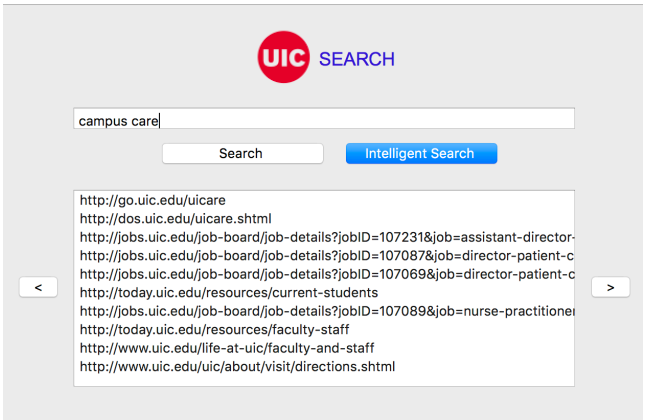
SEARCH ENGINE APP:



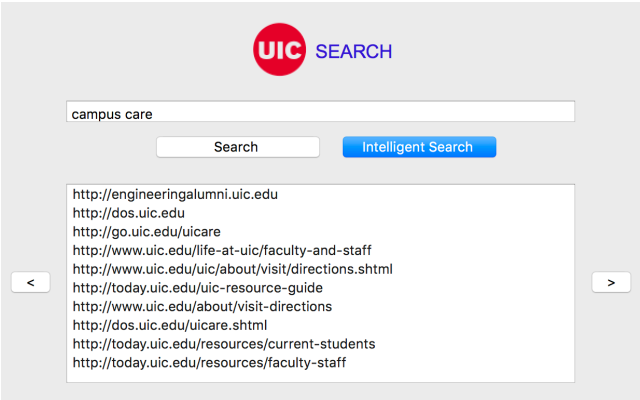
RESULTS:

The below are the comparative results when the search is performed using regular and intelligent search. The result of the left image is obtained using regular search and the result of the right image is obtained using intelligent search.

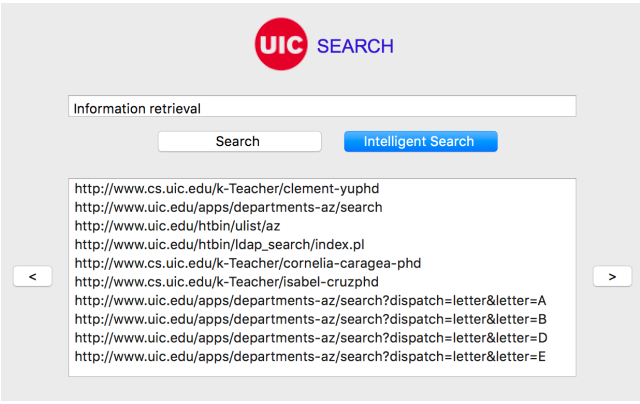
[Regular Search] Query: Campus Care



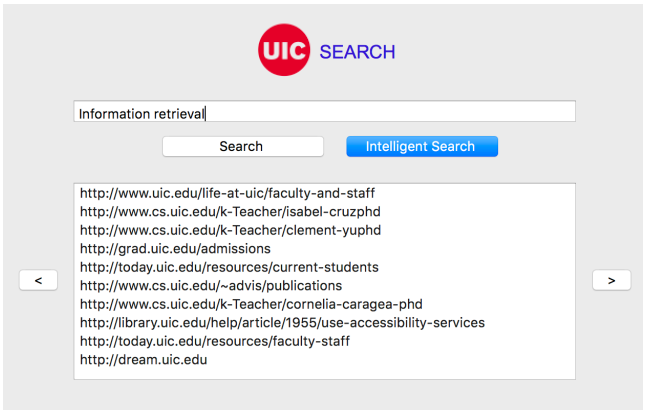
[Intelligent Search] Query: Campus Care



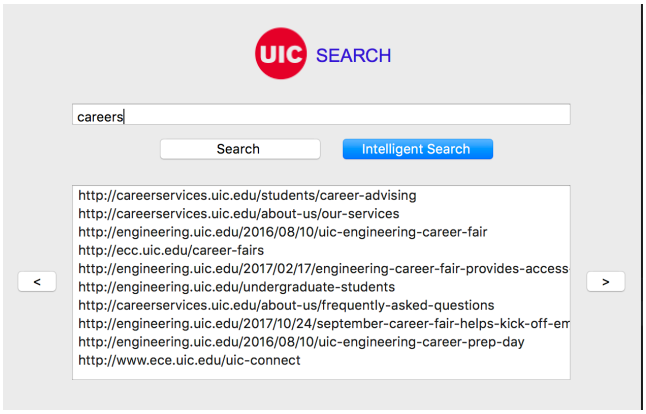
[Regular Search] Query: Information Retrieval



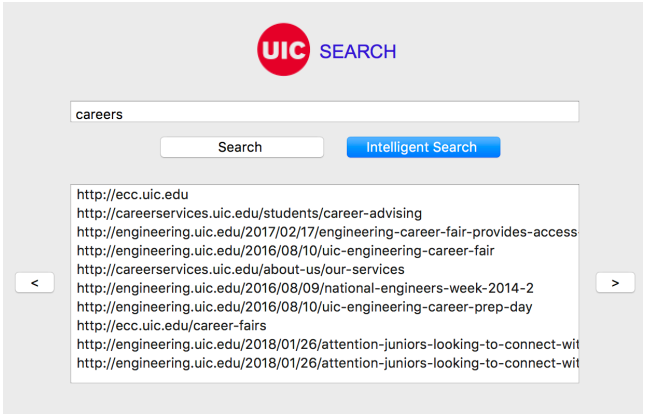
[Intelligent Search] Query: Information Retrieval



[Regular Search] Query: Careers



[Intelligent Search] Query: Careers



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 intelligent search.

FUTURE WORK

The search engine works better when combined with query expansion techniques. I would prefer implementing that along with auto corrections in the user query. Considering n-grams while calculating the TF-IDF also enhances the search engine.