A Web Search Engine

CS 582 Term Project

Sandeep Rane
Department of Computer Science
University of Illinois at Chicago
Chicago, IL, USA
srane3@uic.edu

PROJECT DESCRIPTION

The goal of this project was to design and implement a web search engine for the UIC (University of Illinois at Chicago) web domain. A typical web search engine has several main components such as:

1. a web-crawling agent a.k.a. a spider,
2. a webpage processor or the parser which reads the documents fetched by the crawler, parses it and saves whatever metadata that would be required for the search engine and throws away rest of the junk data
3. an indexer which would create an inverted index on the words present in the web pages to make your querying process faster
4. a ranking engine which is responsible for ranking the webpages using some techniques so as to provide the user with the most relevant pages given a text query
5. a GUI which would allow the users of the system to interact with the application in an efficient manner

This is what a basic web search engine would consist of.

For this purpose, the first part was to create and deploy a web crawler which would crawl over at least 3000 web pages within the specified domain which would enact as a corpus for the search engine. Once we have the desired number of web pages (documents) available, we would integrate it with an information retrieval system that would implement a vector-space model. To improvise further on the results obtained through the vector space model, we were advised to add an 'intelligent aspect' to the search engine. The 'intelligent aspect' that was chosen for this project was PageRank. The entire implementation of this project has been explained thoroughly in this report.

MAIN COMPONENTS

As mentioned earlier, there are five main components of this project. The project has been implemented entirely using Python programming language. Each component of the project has been described below:

1 Web Crawler

A Web Crawler, also referred to as a Spider, is a bot that traverses through the world wide web in a systematic manner. It starts at a specific URL, scans the contents of that web page to see if there are any additional URLs. It then visits the newly discovered URLs in a specific manner. One can either use a DFS or a BFS approach to crawl the weblinks. Since the world wide web is a graph of web pages, we must also keep a track of the URLs that have already been visited. The Crawler would stop once it has parsed about 3000 web pages within the specified domain.

The web crawler has been implemented with the help of a Python library called BeautifulSoup. The urllib package consists of a request module that allows to send request and receive response over the web. While requesting to read a page over the web, one might encounter several types of exceptions such as HTTP Error, URL Error or SSL certification error. One way to handle SSL certificate error is to set the 'verify = false' while opening the url. In this project, we have rather handled them by using the ssl library to ignore the web pages that cause such an exception. The HTTP and URL Error have been handled by using the urllib library.

We are using a BFS technique to parse the webpages, by storing the unvisited URLs in a queue data structure. In order to speed up the crawling process, multiple sub crawlers are spawned through multithreading by using the threading library. Each webpage consists of links to other pages i.e. URLs and textual data.

The URLs that are encountered need to be processed: They must belong to the 'uic.edu' domain, 'https' URLs have been converted to 'http', it has been ensured that all URLs end with a '/' character, URLs have been normalized for '#' characters and all URLs that have relative path have been converted to absolute paths.

The textual data has been processed as explained below.

2 Webpage Parser

The textual data cannot be obtained directly from a webpage mainly because of its syntax. An HTML page consists of several markup tags that need to be processed in order to obtain the textual content that is embedded within it. Again, the BeautifulSoup library comes in handy as it has a method that allows us to find out all the objects that contain a specific tag. We are interested in the 'a', 'p', 'h1', 'h2', 'h3', 'h4', 'h5' and 'h6' tags as these tags contain the textual information that is rendered on a browser. Once we have captured those objects, we can find the text that lies within.

The obtained text needs to be preprocessed before we can proceed with building the inverted index.

We apply a tokenizer that converts the text to lower case and splits it on blank spaces. Once we have the individual words, we get rid of the stop words and then we perform stemming on it. The stemmer that has been used here is the Porter Stemmer provided by the NLTK library. Once this is done, we once again check if the stemmed word is a stop word and remove it. Also, words that are smaller than 3 characters have been ignored.

3 Inverted Index

An inverted index is a mapping of the words to its location in the documents. In this case, the inverted index is a dictionary with keys as processed words or the tokens and values as a URL dictionary. The URL dictionary itself contains URLs as the keys and the number of occurrences of that given token in that URL as values. So basically, we have a mapping of the tokens to the URLs in which they occur along with the number of times that they occur in them.

After a token has been processed, it would be added to the inverted index if it is not already part of it. This process is carried out over all the encountered tokens, adding all of them to the inverted index.

Now, the Inverse Document Frequency (IDF) is computed for each token and multiplied with the precomputed term frequencies of the tokens which can be found in the inverted index. This would give us the TF-IDF values of each token, so the inverted index then consists of the TF-IDF values for each term. Recall, that the TF-IDF value of a word indicates how important that word is to each document in the entire collection or corpus.

4 Vector-Space Model

The query submitted by the user through the Graphical User Interface would be processed using the same preprocessing technique that took place in the webpage parser component. So, the query is tokenized, stemmed, checked for stop words and so on, to get the tokenized version of the words in the query.

The TF-IDF is calculated for each token of the query, using the IDF that was computed over the entire collection/ corpus and the TF of the tokens in the query.

Next, we compute the similarity of each word of the query with all the crawled web pages, using the cosine similarity metrics. Cosine similarity of each webpage with the query is calculated and then the web pages are sorted in a descending manner using this cosine similarity score.

5 PageRank Algorithm

The search results for a user submitted query can be optimized further using a link analysis algorithm. Several link analysis algorithms are available for this purpose like the Hypertext Induced Topic Search algorithm (HITS), PageRank algorithm, etc.

PageRank algorithm has been found to be very effective for ranking of web pages and has been used by the Google Search engine as well. Although today, that isn't the only algorithm used by google to rank pages, yet it proves to be a good 'intelligent aspect' to improvise our search results.

The algorithm is implemented as follows:

While crawling the web pages, extract the URL of the current webpage. This URL acts as a source and hence we add a node for it in the page rank graph. Next, we find out all the links that are present in the webpage. These are the destinations and hence we create a node for each URL if it doesn't already exist in the graph. An edge is added from the current URL node (source) to the nodes of the destination URLs, if an edge doesn't already exist between them.

Once all the crawled webpages have been parsed, we'll have a graph ready for implementing the page rank algorithm. Initially, all the nodes have been assigned a score of 1/n where n is the total nodes in the graph. The page rank algorithm iteratively computes the scores for all nodes in the graph by considering the incoming edges to that node. A damping factor of 0.85 has been used and max iteration limit of 10 has been given to the algorithm. Dangling nodes – nodes with no outgoing edges have been handled separately by assigning a dangling score to these nodes.

Once we have PageRank scores, we sort the web pages in a descending order on the basis of their PageRank scores.

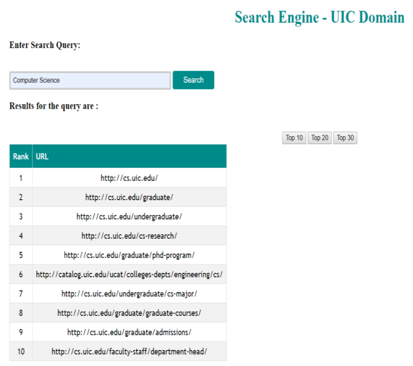
6 Integration of Intelligent Aspect

The PageRank algorithm has been combined with the vector space model. We perform a selection on the ranked web pages returned by the vector space model. We consider only the top 30 pages returned after the cosine similarity computation. For these 30 pages, we rank these pages using the rankings provided by the PageRank algorithm.

7 Graphical User Interface

The Flask library has been used to develop the GUI of the search engine. It consists of a single web page with a textbox where the users can type in their query and a submit button to submit the results to the search engine implemented using Python. All the computations for are performed by the python program and the top 30 results are returned as an array.

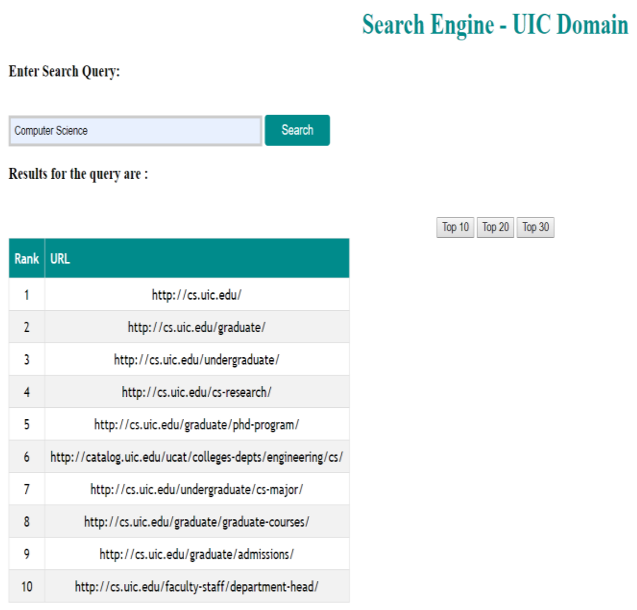
The user can view only the Top 10 queries in a tabular format.



Rank	URL
1	http://cs.uic.edu/
2	http://cs.uic.edu/graduate/
3	http://cs.uic.edu/undergraduate/
4	http://cs.uic.edu/cs-research/
5	http://cs.uic.edu/graduate/phd-program/
6	http://catalog.uic.edu/ucat/colleges-depts/engineering/cs/
7	http://cs.uic.edu/undergraduate/cs-major/
8	http://cs.uic.edu/graduate/graduate-courses/
9	http://cs.uic.edu/graduate/admissions/
10	http://cs.uic.edu/faculty-staff/department-head/

Figure 1: Graphical User Interface of the Search Engine

Figure 2: Search Bar



Search Engine - UIC Domain

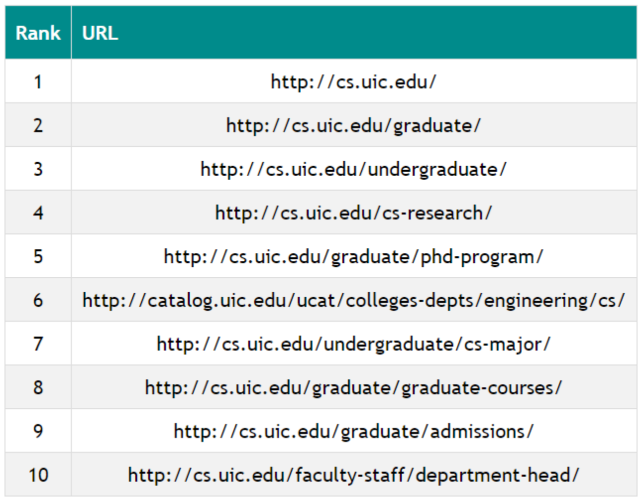
Enter Search Query:

Computer Science

Results for the query are :

Rank	URL
1	http://cs.uic.edu/
2	http://cs.uic.edu/graduate/
3	http://cs.uic.edu/undergraduate/
4	http://cs.uic.edu/cs-research/
5	http://cs.uic.edu/graduate/phd-program/
6	http://catalog.uic.edu/ucat/colleges-depts/engineering/cs/
7	http://cs.uic.edu/undergraduate/cs-major/
8	http://cs.uic.edu/graduate/graduate-courses/
9	http://cs.uic.edu/graduate/admissions/
10	http://cs.uic.edu/faculty-staff/department-head/

Figure 3: A Closer View of the GUI



Rank	URL
1	http://cs.uic.edu/
2	http://cs.uic.edu/graduate/
3	http://cs.uic.edu/undergraduate/
4	http://cs.uic.edu/cs-research/
5	http://cs.uic.edu/graduate/phd-program/
6	http://catalog.uic.edu/ucat/colleges-depts/engineering/cs/
7	http://cs.uic.edu/undergraduate/cs-major/
8	http://cs.uic.edu/graduate/graduate-courses/
9	http://cs.uic.edu/graduate/admissions/
10	http://cs.uic.edu/faculty-staff/department-head/

Figure 4: Results in tabular form

MAIN CHALLENGES ENCOUNTERED

Throughout the development cycle of the search engine, several challenges were encountered -

1. While implementing the crawler, initially I had deployed a single crawler that used to crawl and parse the web pages. This was very time consuming as a single crawler had to parse over 3000 web pages. A work around this method was to deploy multiple sub crawlers by making use of multithreading in python. Initially only a single crawler would be deployed from the 'http://cs.uic.edu' webpage. Later, I used about 75 sub crawlers at a time to begin crawling from 75 webpage URLs present in the queue.
2. Several types of errors were encountered while trying to request the web pages such as HTTP Error, URL Error, SSL certification error and so on. All these errors prevented crawling of those specific web pages and terminated the crawler activity. These were handled using exception handling libraries.
3. Integration of vector space model with page ranking algorithm was not straight forward. The vector space model assigns a rank by considering the query tokens where as the page rank score is independent of the query. A weighting scheme wasn't possible by integrating the two scores as both are based on completely different concepts. The approach used was to perform a selection using the vector space model and then perform selection using page rank model.
4. Integration of a GUI was also tricky. All of the term assignments were focused on the back-end. Several libraries such as Tkinter, Django and Flask are available. To make the results available immediately to the user, the TF-IDF and PageRank results were cached in json files.

WEIGHTING SCHEME, SIMILARITY MEASURE USED AND COMPARISONS

As mentioned earlier, a TF-IDF weighting scheme has been used in the vector space model. The TF-IDF matrix is computed for the entire crawled webpage collection. This is an m*n matrix where m represents the vocabulary size and n represent the total webpages crawled. TF-IDF value of a word indicates how important that word is to each document in the entire collection or corpus.

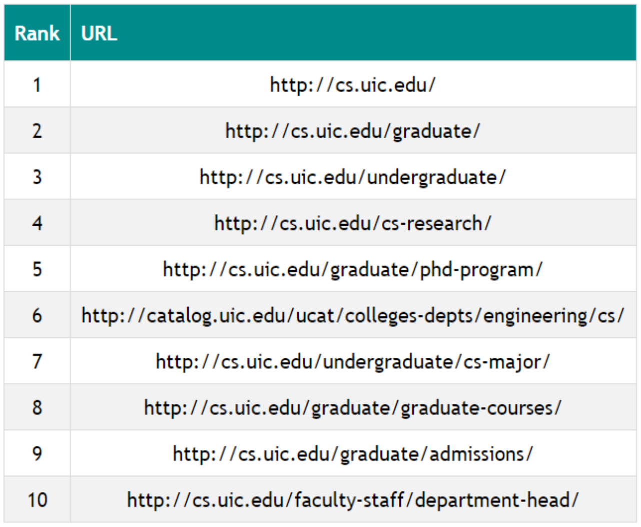
Similarity measure is used for finding out how closely related two entities are. Here, the Cosine Similarity metrics has been used to find the similarity of each word of the query with all the crawled web pages. The higher the score, the higher is the similarity between the two entities. Hence the webpages are sorted in a descending order of their similarity scores.

Comparison of the weighting scheme - The vector space model approach was initially tried as a standalone approach. The results found using the vector space model using cosine similarity are generally close to our expectations as we try to compute the similarity of the query with the documents. However, the precision scores have found to improve further using a PageRank algorithm. This is mainly because the PageRank algorithm has been applied on the Top 30 pages ranked using cosine similarity. This approach betters the original ranking scheme which was purely dependent on the vector space model.

MANUAL EVALUATION OF TEST RESULTS

Some of the queries that were tested on the search engine are listed below.

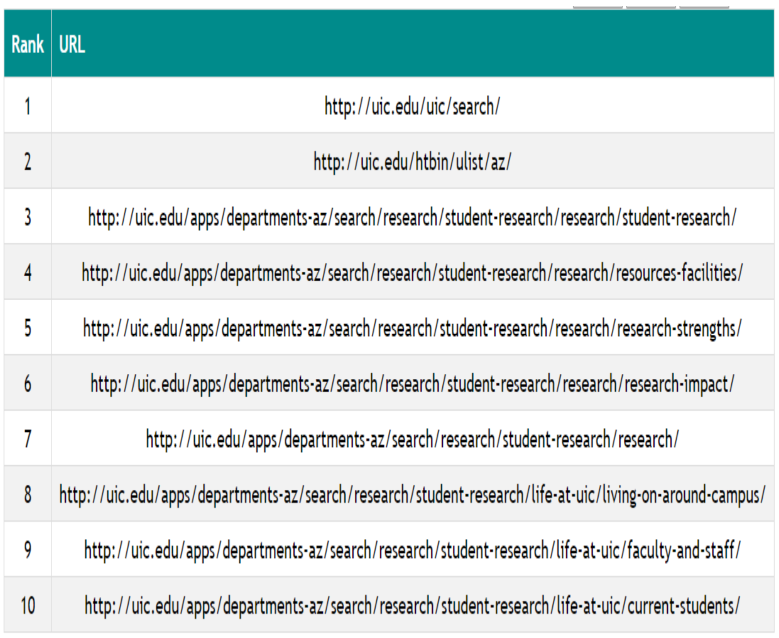
1. Computer Science



Rank	URL
1	http://cs.uic.edu/
2	http://cs.uic.edu/graduate/
3	http://cs.uic.edu/undergraduate/
4	http://cs.uic.edu/cs-research/
5	http://cs.uic.edu/graduate/phd-program/
6	http://catalog.uic.edu/ucat/colleges-depts/engineering/cs/
7	http://cs.uic.edu/undergraduate/cs-major/
8	http://cs.uic.edu/graduate/graduate-courses/
9	http://cs.uic.edu/graduate/admissions/
10	http://cs.uic.edu/faculty-staff/department-head/

Precision score = 10/10

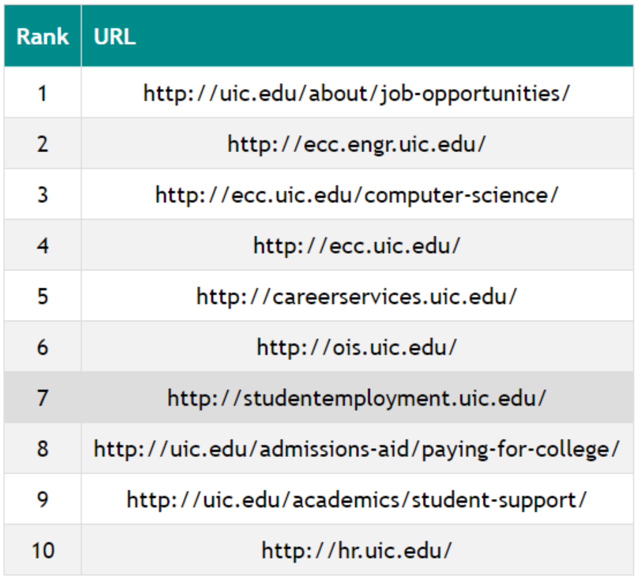
2. Student Research



Rank	URL
1	http://uic.edu/uic/search/
2	http://uic.edu/htbin/ulist/az/
3	http://uic.edu/apps/departments-az/search/research/student-research/research/student-research/
4	http://uic.edu/apps/departments-az/search/research/student-research/research/resources-facilities/
5	http://uic.edu/apps/departments-az/search/research/student-research/research/research-strengths/
6	http://uic.edu/apps/departments-az/search/research/student-research/research/research-impact/
7	http://uic.edu/apps/departments-az/search/research/student-research/research/
8	http://uic.edu/apps/departments-az/search/research/student-research/life-at-uic/living-on-around-campus/
9	http://uic.edu/apps/departments-az/search/research/student-research/life-at-uic/faculty-and-staff/
10	http://uic.edu/apps/departments-az/search/research/student-research/life-at-uic/current-students/

Precision score = 8/10

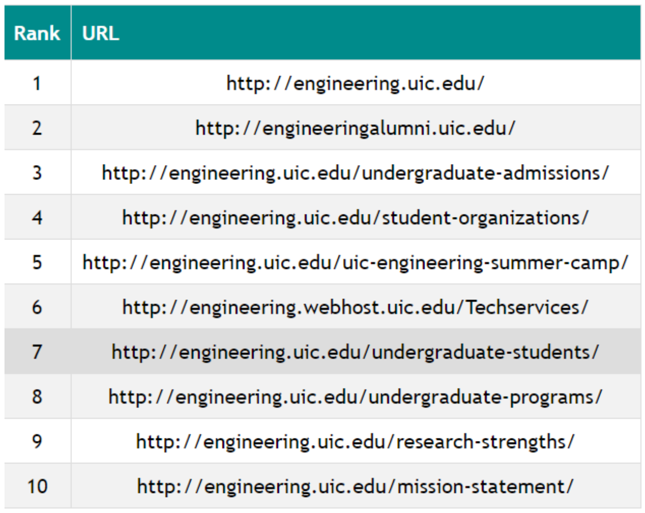
3. Employment



Rank	URL
1	http://uic.edu/about/job-opportunities/
2	http://ecc.engr.uic.edu/
3	http://ecc.uic.edu/computer-science/
4	http://ecc.uic.edu/
5	http://careerservices.uic.edu/
6	http://ois.uic.edu/
7	http://studentemployment.uic.edu/
8	http://uic.edu/admissions-aid/paying-for-college/
9	http://uic.edu/academics/student-support/
10	http://hr.uic.edu/

Precision Score = 8/10

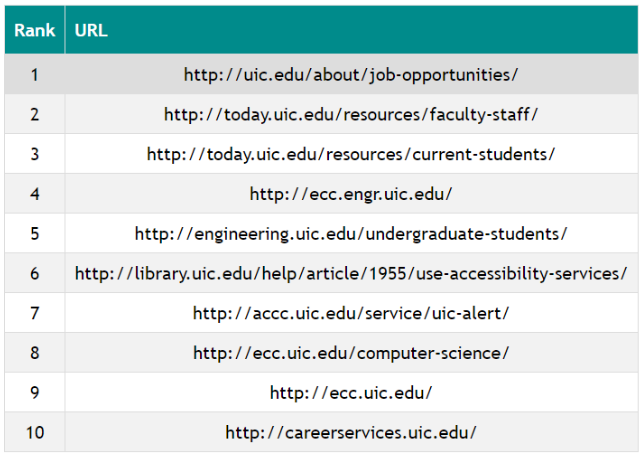
4. Engineering Department



Rank	URL
1	http://engineering.uic.edu/
2	http://engineeringalumni.uic.edu/
3	http://engineering.uic.edu/undergraduate-admissions/
4	http://engineering.uic.edu/student-organizations/
5	http://engineering.uic.edu/uic-engineering-summer-camp/
6	http://engineering.webhost.uic.edu/Techservices/
7	http://engineering.uic.edu/undergraduate-students/
8	http://engineering.uic.edu/undergraduate-programs/
9	http://engineering.uic.edu/research-strengths/
10	http://engineering.uic.edu/mission-statement/

Precision Score = 10/10

5. Career Services



Rank	URL
1	http://uic.edu/about/job-opportunities/
2	http://today.uic.edu/resources/faculty-staff/
3	http://today.uic.edu/resources/current-students/
4	http://ecc.engr.uic.edu/
5	http://engineering.uic.edu/undergraduate-students/
6	http://library.uic.edu/help/article/1955/use-accessibility-services/
7	http://accc.uic.edu/service/uic-alert/
8	http://ecc.uic.edu/computer-science/
9	http://ecc.uic.edu/
10	http://careerservices.uic.edu/

Precision Score = 5/10

A DISCUSSION OF THE RESULTS AND CONSIDERATIONS

As we can see from the results, the average precision scores over the top 10 results is quite astonishing in case of the PageRank combined Vector space model.

The average precision scores over the top 10 ranked webpages obtained for the five queries was computed manually. The scores are as follows:

Average Precision score = 41/50 = 82%

As seen from the average precision score, there is still plenty of room for improvement. The application can only be considered a mini version of Google had it been able to perform significantly well over the queries.

What can be improved? We are only parsing over a subset of web pages (only 3000 pages). Thus, the results obtained is only based on the corpus of webpages that was crawled. To allow support for wider array of queries, more number of pages need to be crawled.

Also, the number of selected pages after the two ranking can be tweaked to see what difference does it make to the overall results that are returned.

RELATED WORK

Developing a search engine is not just about applying a vector space model or page rank algorithm. A large scale web search engine is quite complex and there is continuous research going on in the field to improve the search quality.

There is a lot of work going on in the domains of Search Engine Optimization, scalability of a search engine for large number of web pages, efficient query caching, smart disk allocations and so on.

No system is complete which is why developers of popular search engines like Google, Baidu, Yahoo, Bing, etc. are conducting continuous research in this domain.

FUTURE SCOPE

As has been mentioned earlier, this is a relatively smaller search engine which crawls only about 3000 web pages that too within the specified web domain. An improvement over this method could be to crawl over more number of pages rather than just restricting the system to 3000 pages. This would allow us to serve varying user queries.

Additionally, if we want to improve further, we may extend this to different domains and not just restrict it to the UIC domain.

Also, the user should be provided with an option to view more and more number of weblinks as a link which is lower in ranking might be relevant to the user.

The precision of the search results need to be optimized further and the current approach might not be suitable if we are to further extend the search engine.

REFERENCES

[1] https://stackoverflow.com/questions/112248/building-a-web-search-engine
[2] https://en.wikipedia.org/wiki/Web_crawler
[3] https://en.wikipedia.org/wiki/Inverted_index
[4] https://en.wikipedia.org/wiki/Vector_space_model
[5] https://en.wikipedia.org/wiki/Cosine_similarity
[6] https://en.wikipedia.org/wiki/PageRank
[7] https://en.wikipedia.org/wiki/HITS_algorithm
[8] https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[9] https://2.python-requests.org//en/latest/user/advanced/#ssl-cert-verification
[10] https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[11] https://2.python-requests.org//en/latest/user/advanced/#ssl-cert-verification
[12] https://www.python.org/
[13] http://flask.pocoo.org/