Search Engine

Shashank Maithani
Computer Science
University of Illinois, Chicago
Chicago, Illinois, USA
Shanks2999@outlook.com

ABSTRACT

The project aims to implement all the theoretical concepts that were taught as a part of the CS582 course structure. The one practical application which would bring all those concepts to life was to build a search engine (similar to google) but on a much smaller scale.

This engine would consist of a web Crawler/Spider which would start from a base URL which in this case would be www.cs.uic.edu. The search would be limited to UIC domain and would incorporate a smart component either HITS/PageRank/Clustering.

This engine would then traverse all the subsequent links from that page in order to form a web graph. Cosine similarity of the documents would be used to find the most relevant documents. Page Rank would be used to rank all the retrieved pages in relative order.

REQUIREMENT ANALYSIS

1. Humans

- The system will be open to public use.
- Will be used by thousands of people within UIC as well as outside.
- Periodic usage using PCs or laptops
- At any time (Must be available 24x7)

2. Tasks

High Level

- Provide a system to which provides search facility by keyword.
- The system is strictly web based and is limited to UIC domain only.
- Must incorporate a smart component in order to complete the search.

Activities and Tasks

- User search could be a single word or a phrase of multiple words.
- Search must find pages which most similar to the query
- At certain staged during execution data is stored in intermediary files.

3. Data

Data is collected in form of text from the pages which are crawled by the spider. The crawl is in BFS fashion and is limited to first 3000 working web pages which are hit.

During the crawl the web pages hit are stored in intermediate files in order from 1 to 3000 for faster search later. The text stored in those files is already subjected to cleansing and is transformed before the search. Operations performed on the pages include.

- Stop words removal (pre-stemming)
- Stemming
- Stop words removal (post-stemming)
- Punctuation removal
- Digits removal
- Tokenization

4. Flow

- The user searches for a word or sentence.

- The Search Engine searches for most relevant pages within UIC domain.
- Sorts them according to the smart component mentioned (Page Rank).
- Displays top 200 results to the user.
- Overall search takes completed within 10 seconds.

5. Non-functional Requirements

- **Scalability**
  - ✓ System should accommodate high amounts of web traffic.
- **Privacy**
  - ✓ All data will be kept public.
- **Learnability**
  - ✓ System should be usable with minimum training.
- **Operating Environment**
  - ✓ The system would be online.
  - ✓ The system should have cross-browser support.
- **Availability**
  - ✓ The system must be online 24x7.
- **Resiliency**
  - ✓ The system must be highly stable, in order to achieve high uptime.

1.6. Probes

- Is this a visualization?
  - ✓ No. Even though a web graph is formed, there is no visualization necessary.
- Other existing tools?
  - ✓ Currently, there are other web search engines available like google, Yahoo and Bing.
  - ✓ Although they are very large scale covering the entirety of web. Our search coverage is very limited compared to those.
- Possibilities/Other ideas
  - ✓ A web app could be developed similar to google having similar search capabilities.

COMPONENTS

- **Web Crawler/Spider**

The crawler starts from www.cs.uic.edu. The UIC website allows spider both to crawl its webpages. This was checked beforehand in the Robots.txt file within the UIC root folder i.e. https://www.cs.uic.edu/robots.txt.

The crawler then hits all subsequent links using BFS approach. There is a hit count limit of 3000 web pages. The text is extracted from each hit webpage and is send to the preprocessor.

- **Preprocessor**

The main task of the preprocessor is the cleansing and transformation of data. The data fetched from the web page is unstructured and sometimes have Unicode samples from images etc.

The data is cleansed by removing stop words, punctuations, digits. The data is then stemmed and stored locally. The same operations are applied to the query as well.

- **File Handler (Pickle/JSON/TXT)**

During the entire process of crawling, data is stored intermittently in various file formats. This makes the actual search easy and extremely fast i.e. ~6 seconds. Here are the file formats in which data is stored:

TXT: The crawled web pages are stored in txt files in 2 lines. 1st line contains the URL and the 2nd contain the tokenized web page.

PICKLE: This '.p' file contains the web graph which is required for Page Rank. This contains ap packaged hash map of links of links outgoing and incoming. Requires pickle library of python to run.

JSON: This contains the key value pairs of the URL and its subsequent page rank scores.

- **Searching**

  After all the crawling and preprocessing the actual searching is done using TF-IDF weighing scheme for all the 3000 documents which are extracted locally.

  The search is narrowed down using cosine similarity mechanism followed by Page rank to sort all the retrieved documents in order from 1 to N.

- **UI**

  The UI is built using python Flask API. This resulted in a web app deployed on people.uic.edu website. It is a simple UI using POST to invoke python functions to perform the search.

CHALLENGES

- **Finding alive pages**

  One major challenge was finding links which exist when forming web graph without downloading the page. This was time consuming as each page takes 2-3 secs to load. And this has to be done for all 3000 web pages and their child links which resulted in more than 16000 web pages.

  This was tackled through using Http.Client to get only the status code for the page without loading the page. If status code is 200 then proceed else not

- **Handling Redirects**

  Some URL's are resolved at load time which then loads a different page altogether. Major challenge resided in finding the redirects.

- **Avoiding Cycles**

  This became a challenge when sites like:

  - ✓ https://www.cs.uic.edu
  - ✓ http://www.cs.uic.edu
  - ✓ cs.uic.edu
  - ✓ www.cs.uic.edu
  - ✓ www.cs.uic.edu/

  All resulting in the same web page.

- **Huge memory Leaks**

  This became apparent after 2-3 days of testing when errors of stack overflow became common. All functionality of crawling, forming the graph and Page Rank cannot be done on system memory. Not to mention the risk of error resulted in a huge waste of time.

  This was then resolved by storing the data in JSON/TXT files at various stages during the execution of the program.

- **Staying within UIC domain**

  This wasn't a problem initially but was trouble some in few cases involving redirects linking to social media websites like Twitter/Facebook/Google.

  The crawler would many times get lost in random webpages in the web. This was solved by creating a set of all visited websites. The set would only contain the URL after the 'www' part to avoid corner cases.

WEIGING SCHEME

Once all the cleansing and pre-processing was done, the data was stored in 3000 txt files. Once files were available locally, TF-IDF (Term Frequency – Inverse Document Frequency) was applied on the data. The TF-IDF weighing scheme assigns term *t* a weight in document *d* given by:

Tf-idf(t,d) = tf(t,d) x idf(t)

This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The 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.

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

SIMILARITY MEASURE

Once the query is passed and the weights are assigned to the corpus as well as to the query terms. A similarity measure is used to compare the query terms to the documents to fins the most relevant. In this case we use the cosine similarity measure to get the documents.

The cosine similarity is another similarity metric envisioning user preference as points in an n-dimensional space. This is based on a vector space model from each document we derive a vector. The set of documents in a collection then is viewed as a set of vectors in a vector space. Each term will have its own axis. Using the formula given below we can find out the similarity between any two documents.

Cosine Similarity (d1, d2) = Dot product (d1, d2) / ||d1|| * ||d2||

ALTERNATIVES

Word Embeddings

Feature learning method where words or phrases from vocabulary are mapped to vectors of real numbers.

Word2Vec: Created by team of researchers at google. In this word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space

More info here: https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

Earth Mover's Distance

Once the word embeddings are formed Earth Mover's Distance could be used to calculate the distance between the documents. This distance measure is rather slow (quadratic time) so it might not scale well if you have many pages to go through.

Jaccard Index

This similarity measure compares members for two sets to see which members are shared and which are distinct. It's a measure of similarity for the two sets of data, with a range from 0% to 100%. The higher the percentage, the more similar the two populations. Although it's easy to interpret, it is extremely sensitive to small samples sizes and may give erroneous results, especially with very small samples or data sets with missing observations.

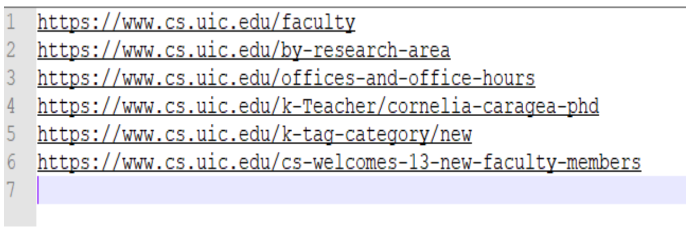
Other measures of similarity include:

- Inner product
- Document distance
- Dice coefficient

SAMPLE QUERIES

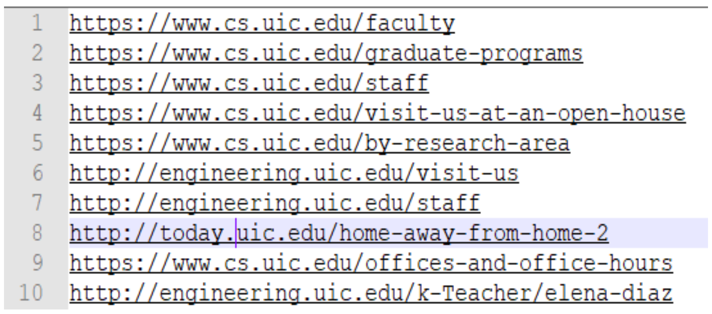
Here are a few sample queries which I have tested on the Search Engine and the corresponding search result. Here the data is all computed locally in the JSON/Pickle files so the query tine in all the cases is ~6 secs.

Query 1: "Cornelia Caragea"



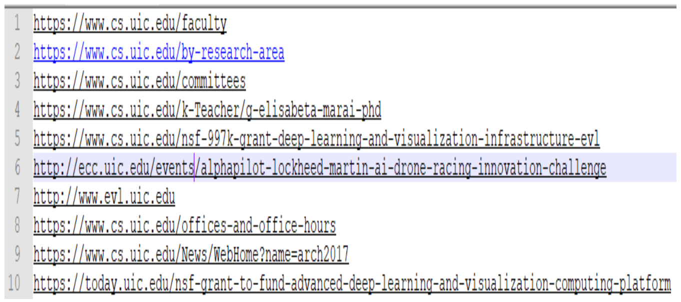
```
https://www.cs.uic.edu/faculty
https://www.cs.uic.edu/by-research-area
https://www.cs.uic.edu/offices-and-office-hours
https://www.cs.uic.edu/k-Teacher/cornelia-caragea-phd
https://www.cs.uic.edu/k-tag-category/new
https://www.cs.uic.edu/cs-welcomes-13-new-faculty-members
```

Query 2: "Elena Zheleva"



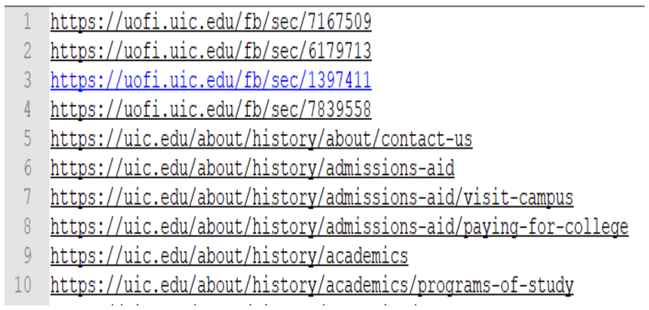
```
https://www.cs.uic.edu/faculty
https://www.cs.uic.edu/graduate-programs
https://www.cs.uic.edu/staff
https://www.cs.uic.edu/visit-us-at-an-open-house
https://www.cs.uic.edu/by-research-area
http://engineering.uic.edu/visit-us
http://engineering.uic.edu/staff
http://today.uic.edu/home-away-from-home-2
https://www.cs.uic.edu/offices-and-office-hours
http://engineering.uic.edu/k-Teacher/elena-diaz
```

Query 3: "Liz Marai"



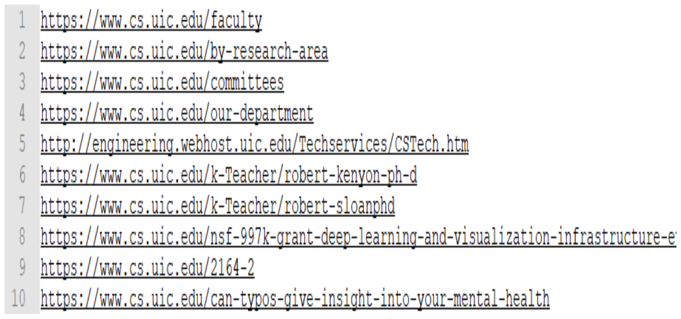
```
https://www.cs.uic.edu/faculty
https://www.cs.uic.edu/by-research-area
https://www.cs.uic.edu/committees
https://www.cs.uic.edu/k-Teacher/g-elisabeta-marai-phd
https://www.cs.uic.edu/nsf-997k-grant-deep-learning-and-visualization-infrastructure-evl
http://ecc.uic.edu/events/alphapilot-lockheed-martin-ai-drone-racing-innovation-challenge
http://www.evl.uic.edu
https://www.cs.uic.edu/offices-and-office-hours
https://www.cs.uic.edu/News/WebHome?name=arch2017
https://today.uic.edu/nsf-grant-to-fund-advanced-deep-learning-and-visualization-computing-platform
```

Query 4: "University Of Illinois"



```
https://uofi.uic.edu/fb/sec/7167509
https://uofi.uic.edu/fb/sec/6179713
https://uofi.uic.edu/fb/sec/1397411
https://uofi.uic.edu/fb/sec/7839558
https://uic.edu/about/history/about/contact-us
https://uic.edu/about/history/admissions-aid
https://uic.edu/about/history/admissions-aid/visit-campus
https://uic.edu/about/history/admissions-aid/paying-for-college
https://uic.edu/about/history/academics
https://uic.edu/about/history/academics/programs-of-study
```

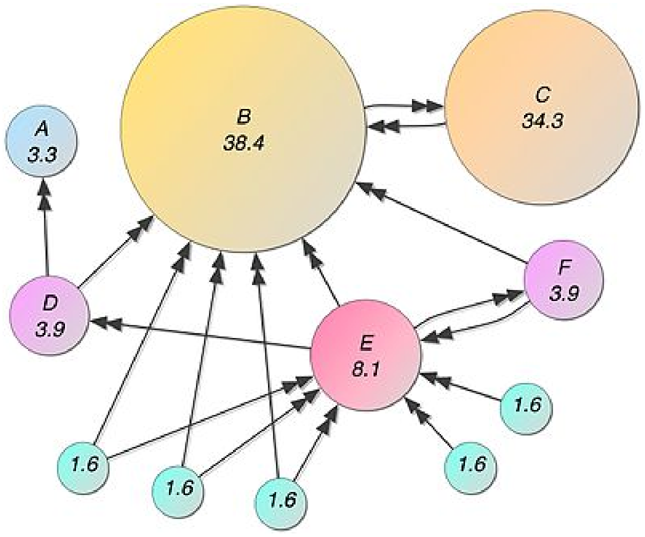
Query 5: "Robert Sloan"



```
https://www.cs.uic.edu/faculty
https://www.cs.uic.edu/by-research-area
https://www.cs.uic.edu/committees
https://www.cs.uic.edu/our-department
http://engineering.webhost.uic.edu/Techservices/CSTech.htm
https://www.cs.uic.edu/k-Teacher/robert-kenyon-ph-d
https://www.cs.uic.edu/k-Teacher/robert-sloanphd
https://www.cs.uic.edu/nsf-997k-grant-deep-learning-and-visualization-infrastructure-e
https://www.cs.uic.edu/2164-2
https://www.cs.uic.edu/can-typos-give-insight-into-your-mental-health
```

INTELIGENT/SMART COMPONENT

The smart component used in the search engine is Page Rank which is an algorithm used by Google Search to rank web pages in their search engine results. PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.



Source: https://en.wikipedia.org/wiki/PageRank

Formula
PageRank for a given page = Initial PageRank + (total ranking power ÷ number of outbound links) + …

Damping Factor
PR(A) = (1-d) + d (PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))

This ranking measure was applied on our search engine after the relevant documents was retrieved by our cosine similarity measure. Once the Page Rank scores were calculated they were compared with the retrieved documents.

The relevant documents were then sorted according to their Page Rank scores and were displayed in their respective order.

Comparison
Referring to the memory leak challenge earlier the execution was divided in 2 parts:

- **Computing/Assembly**
  This part focuses on the crawling and web graph generation. Both of which are stored locally in the system.

- **Querying**
  This part focuses on searching the files locally instead of hitting the web pages again.

Overall Compute time (with page rank): **3 to 3.5 hrs**

Overall Query time (with page rank): **6 to 10 sec**

Overall Compute time (without page rank): **1.5 hrs**

Overall Query time(without page rank): **5 to 10 secs (negligible change. Only N extra comparisons as the scores stored are already in sorted order)**

ERROR ANALYSIS

The project encountered its fair share of errors, some of which were fixed, and some were not.

A major one was "Stack Overflow" which was caused because of huge memory leaks in the searching. This was due to the loading of web pages in the heap memory and almost brought the project development to a halt for a few days. This was resolved by storing the data in local files during various stages of execution.

Another one was handling dead pages within the UIC domain. Many pages points to links which are either dead (404) or are redirects to another page within UIC or altogether outside the domain (301). Such corner cases were handled by virtually opening the page in the except clause.

The final one was during the development of an additional feature which I was trying to build via integration of Flask API with the python method. The UI was built and deployed on "people.uic.edu" but the POST functionality is not associated with the python function. This communication is still buggy and will hopefully be resolved in future release.

The project as of now is fully functional and the results have improved through intermediary testing**.**

RELATED WORK

A web search engine is a long research topic which was studied well over a decade. Their search is not limited to text but expands to images, videos and infographics as well. Our search however scalable is limited to UIC domain. This is because of the gigantic amount of data available on the web which would be too much for a single machine to handle. For this major search engine companies have data centers available for this specific purpose.

There are still some content not capable of being searched by a web search engine, generally called as deep web. In current time there are a lot of search engines available which crawl the web, some of those are mentioned below:

- Google
- Bing
- Yahoo
- Ask.com
- AOL.com
- Baidu
- Duckduckgo

FUTURE WORK

Even though the search engine is fully functional there is still room for further improvement. Few features which can be added include:

- Keyword Extraction

Taking keywords from the "div" or "p" tags for the redirected links to increase the content of the web page.

- Query Expansion

Using Relevance feedback or synonyms / wordnet to add more keywords thus expanding the query to facilitate more accurate matches.

ACKNOWLEDGEMETS

I would like to thank Dr. Cornelia Caragea and University of Illinois Chicago (UIC) for the course "Information Retrieval". For me it was very novel and inspiring working with real data on the web and doing a scalable project.

It was really a gratifying experience realizing how the project was tied up to all the concepts taught since the beginning of the semester. The puzzle pieces tied up all together in the final project giving a humbling feeling of completion.

REFERENCES

[1] The original PageRank paper by Google's founders Sergey Brin and Lawrence Page - http://www.db.stanford.edu/~backrub/google.html

[2] Page Rank: https://en.wikipedia.org/wiki/PageRank

[3] Jaccard Index: https://www.statisticshowto.datasciencecentral.com/jaccard-index/

[4] Word2Vec: https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

https://en.wikipedia.org/wiki/Word2vec

[5] TF-IDF: http://www.tfidf.com/