CS 582 Term Project: Search Engine

A web search engine

Shiva Reddy KJ
MSCS Computer science

University of Illinois at Chicago
Chicago, Illinois,USA

skokil2@uic.edu

ABSTRACT

The goal of the project was to build a simple proof of concept search engine system. The search engine would comprise of both the IR system and a web crawler that fetches the documents and other information required to compose the result. The tool would be able to crawl and fetch documents from a default or configurable sub domain, and provide a list of the closest links that contain relevant data for a user query.

The search engine was built as a java package with an associated command line tool to run and test it. The report discusses the process followed in building this project, the system and its implementation detail, the performance of the tool and comments on what could be done to improve it.

CCS CONCEPTS

• Computer science • Information retrieval • Search engine

KEYWORDS

Web crawling, Page rank, Cosine similarity, Stemming

1 Components

The project has been built as a java package with 3 components:-
1. The web crawler
2. The Indexing engine
3. The Search engine

The goal is to divide the responsibility into more easily manageable and interchangeable components. Each component has its own independent responsibilities and outputs.

The project has been built such that it can be used to crawl and query from other sources. Though, it has been mostly tested and tuned using the seed https://www.cs.uic.edu/ , it can be used to index and query for web pages from other similar sources. The options provided and instructions on how to use will be discussed in following sections.

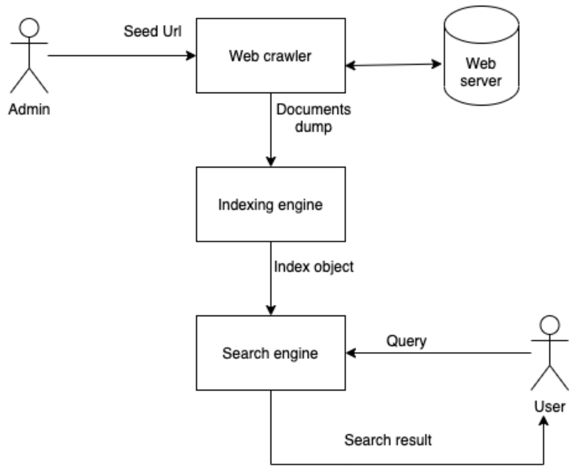


Fig 1.1 *A model representing the design*

1.1 The web crawler

A web crawler is a tool that can be used to fetch a document typically a web page and recursively traverse and fetch related web pages by using the links on the document.

The web crawler in this project behaves similarly using a breadth first search strategy to fetch a collection of pages starting from a seed url. The component has been built using multithreading capabilities in Java allowing it to fetch documents in parallel, improving the speed of download.

Having a high number of threads is of course not always advisable as we might end up putting a load too high on the target server. This can be handled by decreasing the number of threads or slowing our fetch by pausing our operation after every request. The user can also set a limit on the number of web pages they want to download.

The configuration requirements for the crawler are :-
1. The seed URL
2. The destination directory to store downloaded documents
3. Number of threads
4. Wait time for each request.
5. Maximum number of resources to fetch

The crawler outputs each page as a custom serializable object stored as a file in the destination directory which holds the relevant text content, the web page link and its outward pointing links.

1.1.2 Web crawler rules

It is crucial to set up rules on the web crawler to prevent the crawler from downloading pages that are irrelevant or outside the scope of the intended search area. The crawler has been configured to:-

- Read only http(s)://.. links, this filters out alternative protocols like mailto:
- Filter links that point to resources like pdfs, jpgs,..etc. For the current project we are only interested in the content on a HTML page.
- Filter links that are not on UIC subdomain. For the current project, we are only interested in links that are on the 'uic.edu' domain. This can be additionally configured if the project is repurposed for other contexts.

Additionally, to prevent duplicates the URLs need to be normalized. This prevents URLs which point to the same web page but have different addresses to be retrieved only once. The steps taken to normalize URLs on the web crawler are:-

- All URLs stored will be treated as https
- All URLs will have "www" prefix removed
- All URLs that are not GET queries will have '/' appended in the end.
- All URLs will have anchor tag information stripped out.

Example:-
Pre normalization:-

- https://www.cs.uic.edu#faq
- https://cs.uic.edu/
- https://cs.uic.edu

Post normalization:-

- https://cs.uic.edu/

If the crawler is not able to fetch a URL due to an error, it has been configured to log the failure and skip this link.

1.1.3 Parsing the web page

As mentioned previously, each web page is stored as a document, with the relevant text data, links on the page and the web page's URL address.The project uses the java library jsoup to perform HTML parsing on the downloaded web page, to retrieve the required content.

1.2 The Indexing Engine

The goal of this component is to take a directory path as an input and create an object with the required indexing rules applied. This has been separated from the web crawler to remove dependencies between the indexing strategy and the web crawling functionality.

The output of this component is a file which represents a serializable Java object that has been called KnowledgeBase. This object would contain all the necessary indexes required to perform search engine queries.

Since the chosen indexing strategies for the project are cosine similarity and page rank scores, the indexing engine would read all documents on the provided input path and load the KnowledgeBase object with the page rank scores and TF-IDF metrics which can be used to calculate cosine similarity of the documents based on user queries.

Additional strategies for improving the performance of the TF-IDF vector indexes are :-

1. Including the tokenized URL content as part of the document's tokens with increased weight on the URL.
2. Extra weight added to HTML content stored in header tags (h1, h2,..etc)
3. Extra weight added to HTML content stored in bold tags (b).

These strategy was chosen to take advantage of the context of how the content is represented on the web page. The weight chosen on this project is '2'. Further tuning would help discover an appropriate weight.

The page rank scores for each URL has been calculated using a java library 'jgrapht'. The library was used to build a graph on the retrieved URLs and internal libraries to parse and retrieve the page rank scores on each link.

The text content is converted into words by:-

1. Replacing all special characters with empty
2. Splitting each line into tokens based on whitespace
3. Converting the tokens into lowercase

Each word is then filtered using a porter stemmer[1], and a stop word filter to remove words if they belong to a list of stop-words.

1.3 The Search Engine

The goal of this component is to use the 'KnowledgeBase' object created by the indexing engine, and use it to return a list of relevant URLs using the search strategy. This component can be treated as the runtime execution of the search engine with the crawler and indexer running passively running as preprocessing modules.

Since the strategy chosen for the project is to use a combination of cosine similarity and page rank, the component runs a cosine similarity test for the query against the TF-IDF indexes in the 'KnowledgeBase' object, and uses the score in conjunction with the page rank score to rank documents according to their relevancy.

cosine similarity and page rank are 2 very different measures, cosine similarity measures how similar a document is to a given query while page rank represents how authoritative a page is a source of information. Both values are useful in trying to determine how relevant a URL would be to the given query. The challenge is to find a function that is tuned best to the dataset.

The cosine similarity is split into multiple components. As mentioned previously, both text and header information on each page is set up to be parsed into different components. This allows us to match the query against header information on web pages and against page content independently. Based on user behavior and interest, the weighing of both of these parameters to find optimal results could vary.

By intuition, we can assume that the header would provide more information about what the web page's purpose is, and may provide better results for searches on the site. A distribution of 20-80 has been chosen to build the cosine similarity metric of a web page with the query as the header information is also part of the text information, and a higher weight on header score would impede the diversity of the search results.

Cosine similarity score = (Normalized header cosine similarity * 0.2) + (Normalized text cosine similarity * 0.8)

PageRank score for a web page, provides information about the "quality" of a web page, i.e how many peers consider it a good source of information. Combining cosine similarity with page rank can help provide better context of the relevancy of the page. For this project, I have chosen a distribution of 90 - 10, i.e the cosine similarity contributes to 90 percent of the relevancy score and the page rank contributes to 10 percent of the relevancy score.

The strategy chosen to explore for this project is to:-

1. Find the top 'x' relevant web pages closest to the query using cosine similarity
2. Assign a normalized page rank score to each web page in the top 'x' set.
3. Calculate a 'ranking' score using 90*(cosine similarity) + 10*(normalized page rank score)
4. Rearrange the top 'x' using the new ranking.

The top 'x' documents are essentially chosen using only the cosine similarity, and PageRank is used to influence the ranking of the x documents in the search result. This model was chosen based on the observation that page rank independently had little to no effect on the ranking of documents when added to the cosine score. This approach was taken to use page rank instead to rank the pages locally using cosine similarity score as the key metric for finding the top 'x' documents.

This metric was chosen based on manual checks of relevancy using the distribution. *A thorough research or a large scale test of different distributions (or multi parameter optimization strategies) might provide a better performing split up*.

2. Usage instructions

The project has been structured as a java package.The Main class in the jar can be used to run the project via the console menu. The below command should run the prebuilt project. In the path src/:-

sh run.sh

Optionally, the project can be rebuilt by running
sh build.sh

The console menu provides 2 options, Quick Run and Manual. Quick run allows the user to use the prebuilt indexes to be loaded and used to provide search query results. The prebuilt indexes included as part of the project, are used to discuss the testing and evaluation in this report.

The manual configuration allows the user to fetch the latest version of web pages and build a new index and use it for testing. The procedure to use the manual configuration is:-

1. Run the web crawler
    - Provide a seed URL
    - Provide the number of pages to download
    - Provide a destination directory to store documents
    - Provide the maximum number of threads to use
    - Provide a sleep time period to pause between requests
2. Run the indexing engine
    - Provide the directory with documents to index (from previous step)
    - Provide a name for the Index file
3. Run the search engine
    - Provide a name of the Index file (from previous step)
    - Provide the number of results to display per query.

Note: Once the new index is built, the user can skip the first 2 steps and choose option 3 to use the index to run search queries.

In any stage, typing in "q" and pressing Enter will return to the previous menu.

3. Testing and evaluation

The queries chosen to test and demonstrate the project, the results and a discussion of the results are as follows :-

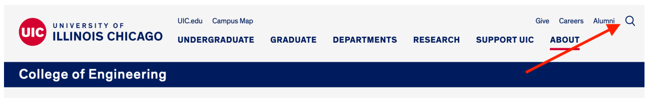
1. Machine learning
2. Course schedule
3. International students
4. Library
5. Cybersecurity
6. Maps

The queries have been chosen as they are represent typical searches on a university search engine. The results of the project can be evaluated under 2 methods:-

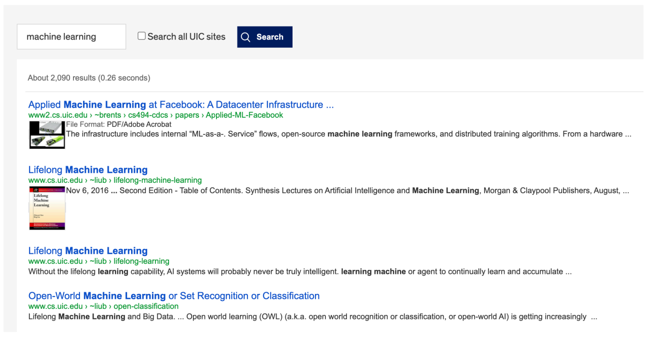
- Manual evaluation
- Comparison with other search engines

Manual evaluation can be done by evaluating if each of the link provided by the tool is relevant, and calculating an accuracy score. Calculating a precision score for manual evaluation is misleading, as it would require the user to find a list of pages that the user thinks is relevant.

Evaluation can also be performed using an alternate (more mature) search engine as a point of reference and a trustworthy source of truth. The alternate search engine chosen in this project, is the search bar present on the "cs.uic.edu" page. The function is powered by google search and can be considered as a fairly mature and advanced engine. This affords an advantage over the project's search engine as it would have done a much more comprehensive indexing as it indexes pdfs, and most web pages have SEO components build to be optimized for Google Search, and ranking is affected by page hits and multiple other complex metrics. The comparison is however useful to identify gaps in our search strategies.



Search bar on https://cs.uic.edu



machine learning ☐ Search all UIC sites Search

About 2,090 results (0.26 seconds)

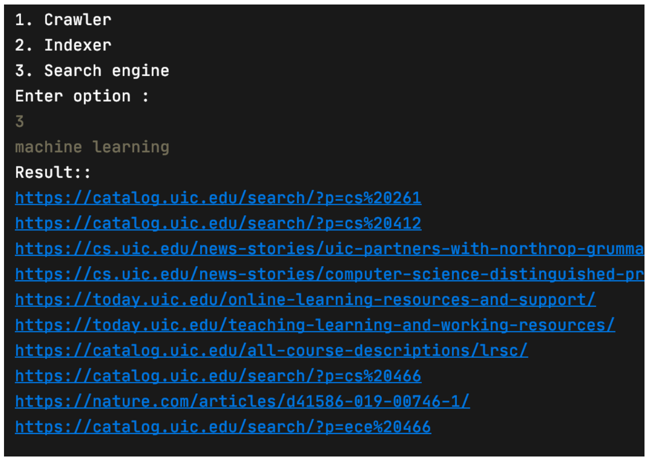
Applied **Machine Learning** at Facebook: A Datacenter Infrastructure ...
www2.cs.uic.edu › ~brents › cs494-cdcs › papers › Applied-ML-Facebook
File Format: PDF/Adobe Acrobat
The infrastructure includes internal "ML-as-a-. Service" flows, open-source **machine learning** frameworks, and distributed training algorithms. From a hardware ...

Lifelong **Machine Learning**
www.cs.uic.edu › ~liub › lifelong-machine-learning
Nov 6, 2016 **...** Second Edition - Table of Contents. Synthesis Lectures on Artificial Intelligence and **Machine Learning**, Morgan & Claypool Publishers, August, ...

Lifelong **Machine Learning**
www.cs.uic.edu › ~liub › lifelong-learning
Without the lifelong **learning** capability, AI systems will probably never be truly intelligent. **learning machine** or agent to continually learn and accumulate ...

Open-World **Machine Learning** or Set Recognition or Classification
www.cs.uic.edu › ~liub › open-classification
Lifelong **Machine Learning** and Big Data. ... Open world learning (OWL) (a.k.a. open world recognition or classification, or open-world AI) is getting increasingly ...

Fig 2.1 *Sample UIC search engine search results*



```
1. Crawler
2. Indexer
3. Search engine
Enter option :
3
machine learning
Result::
https://catalog.uic.edu/search/?p=cs%20261
https://catalog.uic.edu/search/?p=cs%20412
https://cs.uic.edu/news-stories/uic-partners-with-northrop-grumma
https://cs.uic.edu/news-stories/computer-science-distinguished-pr
https://today.uic.edu/online-learning-resources-and-support/
https://today.uic.edu/teaching-learning-and-working-resources/
https://catalog.uic.edu/all-course-descriptions/lrsc/
https://catalog.uic.edu/search/?p=cs%20466
https://nature.com/articles/d41586-019-00746-1/
https://catalog.uic.edu/search/?p=ece%20466
```

Fig 2.2 *Sample search result output*

Tool and web search result details for other query terms have been omitted in this report for brevity but can be found as part of the project in the directory :- *evaluations/*

On performing evaluation using the 2 strategies, the metrics gathered are as follows.

Search queries	Manual Precision	Search engine-Precision	Search engine - Recall
Computer science	0.7	0.2	0.2
Course schedule	0.3	0	0
Library	1.0	0.5	0.5
Cryptography	0.5	0.1	1.0
Maps	0.4	0.3	1.0

Fig 2.3 Precision and recall metrics

The precision for manual search is based on the number of documents that were verified to be relevant by checking the links and manually deciding if the link was what the participant expected to see in the result. The precision and recall for comparing with google search was calculated using the top 10 web pages that have been crawled by the project's crawler.

It was observed that the major deviation in the results between google's search and this tool were caused by difference in the breadth of the number of pages crawled by google search being much higher than this project despite being configured to crawl unto 6000 pages. The webpages in the tool's results are also strongly associated compared to google's results. This might be due to a lack of breadth in the number of web pages parsed by the crawler or due to intentionally injected diversity in google's search algorithm.

Note: This method does not evaluate the ranking of the results within the response

4. Challenges

Each component in the project presents different challenges. Some of them are:

3.1 Web crawling

The main challenge with building the web crawler is ensuring that the normalization and filtering rules are adequate and prevent duplicate URLs while ensuring that relevant URLs are not skipped. The relevant text extraction for each page is dependent on how the information is structured in the web page. The process of discovering the rules on a domain takes iteration and testing.

Developing an efficient multithreaded script using concurrent blocking queues to allow for sharing objects between multiple threads and ensuring that all tasks are completed is also a relatively challenging exercise. Several open source crawlers like Crawler4j[2] were considered but building a simple web crawler turned out to be a more interesting challenge and learning experience. An open source repository, simple-java-web-crawler[3] turned out to be a great design guide for building multithreaded crawlers.

3.2 Indexing and search engine

The main challenge with building the indexing component is translating the TF-IDF vectors to an inverted index, and translating the indexing algorithm into code. The other challenge is validating that the code works correctly. The challenge with building the search engine component is testing and identifying a strategy for combining the page rank metrics with the cosine similarity measure. The confidence in the evaluation strategy would have been higher if the project was tested and used in a larger scale by unbiased participants.

5. Related work and research

There are several other open source projects which provide a much more comprehensive search engine functionalities like Eyers[4], which can be used to build better models and weighing parameters. Alternate ranking algorithms like HITs[5] were also considered, as it provides a ranking for every search instead of using globalized page rank scores in a local scope in this project. The differences and deviations in approaches can be explored deeply in further study.

An interesting idea that seems relevant to this project is Maximal Marginal Relevance[6], which helps rank web pages with a goal to improve the diversity of the results. This is an idea that can be further explored as one of the caveats of the project as evidenced by its comparison with google search results is that the results are strongly associated with each other.

6. Future improvements

The project can be improved in the following directions:

5.1 Web crawling

The crawler uses a very simple strategy to identifying relevant information on the web page. More advanced techniques like using meta tags and content's HTML classes can provide better stronger weights to relevant keywords. Modern web pages typically use tags to provide better SEO(Search Engine Optimization) on their pages. The project's web crawler does not use this information. Moreover, increasing the number of pages the crawler scrapes would improve the results.

5.2 Indexing and search engine strategies

Future versions of this project can implement additional strategies like HITs algorithm to provide alternate metrics to use for providing search engine results. Strategies like Maximal Marginal Relevance can also be explored to improve the diversity of the result set.

5.3 Testing and tuning improvements

An identified test set can be used to validate the project's performance. One way is to pick query sets and use a bot to run it on other search engines which read from the UIC domain and create a somewhat reliable test set and validate the accuracy and precision of the project. This test set can be further used to tune weighting parameters used in the project to provide more optimal results.

5.4 Sub domain level filtering

Allowing the users to choose the sub domain in which they want to look for results is an interesting challenge that can be explored in future iterations

REFERENCES

[1] https://sure.sunderland.ac.uk/id/eprint/3326/
[2] https://github.com/yasserg/crawler4j
[3] https://github.com/krishabhishek/simple-java-web-crawler
[4] https://github.com/Dhyeythumar/Search-Engine
[5] https://patents.google.com/patent/US6112202
[6] https://www.cs.bilkent.edu.tr/~canf/CS533/hwSpring14/eightMinPresentations/handoutMMR.pdf