COMPGI15 Group Report

Option 1: Web Search - Group 3

Zhengyi Yang
University College London
Gower Street
London, United Kingdom WC1E 6BT
zhengyi.yang.13@ucl.ac.uk

Yuan Wei
University College London
Gower Street
London, United Kingdom WC1E 6BT
yuan.wei.13@ucl.ac.uk

Cem Ozgur
University College London
Gower Street
London, United Kingdom WC1E 6BT
cem.ozgur.16@ucl.ac.uk

ABSTRACT

In this paper, we implemented a web search engine using Apache Nutch[3] and Apache Solr[4] for the UCL CS domain. Different ranking algorithms, including BM25, TF-IDF and PageRank, have been applied to rank the search results. PageRank is implemented by our team and attached into Apache Nutch - Solr tools. In addition, evaluations have been done by comparing our search result with Google and UCL search engine of 56 manually selected queries under a variety of metrics. The results shows that ranking with BM25 will give the most similar results with Google among these 3 method whereas PageRank will give the most similar results with UCL search engine. A GitHub repository[1] was created for this project and stored all of our codes.

1 INTRODUCTION OF THE PROBLEM

Information retrieval (IR) has always been a major topic for scientists. Applying the knowledge to search the web is one of the great achievements of IR where IR companies have changed everyone's daily lives.[9] Before Google came out, although there were some options for web search engines, they were not used like they are right now because the results represented to the user were not adequate. The reason behind this fact lies upon the methodologies and algorithms used for retrieval.

For this reason, many different ranking algorithms have been proposed. Google is claimed to be used PageRank with many other features integrated to it.[12] Different scoring functions like BM25, DFR, Jelinke Mercer are also other options for scoring calculation for a better ranking. The current trend and the positive results from different areas of machine learning made it possible to be applied to web search by Learning to Rank.

Other than ranking algorithms, the methodologies for crawling, indexing and parsing are also other key factors that effect the performance of the search engines.

After applying different ranking/indexing/parsing etc. algorithms, there is another issue that needs to be resolved, which is how to evaluate the adequacy/relevancy of the results returned to the user. The evaluation techniques are divided into two in the literature by online and offline. In offline evaluation, domain experts are used for confirming the relevancy and lab generated searches are performed during analysis. However, online evaluation requires some properties of the system used to check the user actions. These actions can be click-trough rate, queries per user or the probability user skips over results they have considered. In addition to all of that, the time that a query is answered is also another main evaluation method.

[1] https://github.com/zhengyi-yang/ucl-search-engine

2 BACKGROUND

The main players in the search engine industry are Google, Bing, Yandex and DuckDuckGo. Billions of people use these tools for their daily search. However, there is a need for searching specific domains, data types, problem context. In addition, for example, websites desire to have an internal search engine to provide their users, which searches only the websites' domain. Therefore, there is a need for implementing one's own product. However, a broad technical domain knowledge is needed to build search engines and this also requires a great deal of time. Thanks to the great advantages of open source software, it is much easier to implement a search engine and tune it to the desired needs by changing ranking function etc.

To implement a search engine with open source software; there are many different packages and frameworks in many languages. Since Java is known by all our team members, we focused on researching Java based libraries. The options that have been considered were Apache Nutch, Apache Solr, Apache Nutch, Indri, Elastic and Terrier.

However, there are also open source libraries for other languages. Sphinx and Xapian[14] are written in C++ and supports multiple ranking algorithms. Sphinx also supports SQL database indexing.[1] In addition, a search engine that provides HTML indexing and searching is Zettair. It is programmed in C with simple and straightforward ranking scoring.

2.1 Apache Nutch

Nutch is an information retrieval package developed with Apache Hadoop[2]. It supports a highly extensible and highly scalable Web crawler. Other reasons that it differentiates itself from other packages is that its interface is open to pluggable parsing, protocols, storage and indexing. Thus, it provides extensible interfaces like Parse, Index and etc. Moreover, it supports pluggable indexing for Apache Solr, which is one of the main reasons that it is used by our team.

2.2 Apache Solr

Apache Solr is another open source package for information retrieval. It is built on Apache Lucene that provides a search interface for users. Solr is also adaptable for users' needs like implementing different ranking functions and/or preferences like using stemming. In addition to these, Solr also handles high traffic needs with its scalable architecture.

2.3 Google Site Search

Google is the most used search engine. The main reason behind its success lies upon the fact that user needs are satisfied by the returned results. This implies that the ranking function of Google performs well.[8] In this experiment, we use a special feature provided by Google, which allows us to search web-pages within any domain defined.

2.4 UCL Search Engine

UCL uses Funnelback for its search requirements. Funnelback is a corporation that provides search engines not only to universities but also to governments, NGO's and etc.[6] Its relevancy ranking system uses a variant of the Okapi BM25[7] algorithm.

3 EXPERIMENTS

3.1 Ranking Algorithms Studied

For our project, we chose Apache Nutch and Apache Solr. The reason behind the decision lies upon the fact that they are highly compatible with each other and the combined product results in a fully implemented web search engine starting from web crawling to providing an interface for queries. Moreover, the product from their combination has the property of providing 7 different scoring functions for ranking. This gave us the option to test different ranking algorithms easily. Moreover, an interface is also provided for Apache Nutch for implementation of other ranking functions.

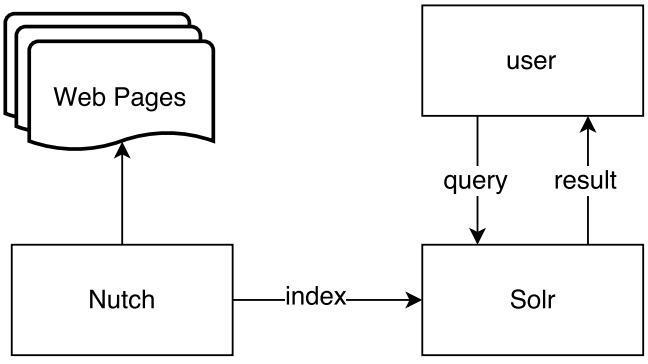

Figure 1: Nutch-Solr System

Figure 1 shows the relation between Nutch and Solr in the Nutch-Solr system. Nutch is an information retrieval package which fetches web-pages, Solr does the indexing, ranking job. However, Nutch also has a scoring system which calculates scores for web-pages, while feeding data to Solr, these scores will be passed to Solr as well and affect Solr's ranking.

Apache Solr is also open to extension of Learning to Rank. The necessary interface is provided within the Apache Solr interface. The main work that needs to be done is to find out specific features to be adopted and used in Learning to Rank with the domain knowledge of us. This opportunity is another major factor that have been considered for the decision of the packages because the group desired to implement Learning to Rank if we had time.

For different ranking options, we have used the great advantage of Apache Nutch and Solr's openness to multiple ranking options. The first one used is the default scoring algorithm of Apache Solr, which is TF-IDF. For the second option, the configuration of Solr is changed to BM25 scoring system since it is one of the base line in information retrieval. [15] Then, we have implemented PageRank in Nutch and adapted it to the Apache Nutch-Solr system to be used by the search engine.

To use the great advantage of machine learning in the area of IR, we have planned to implement Learning to Rank and found out how to implement it in Apache Solr and investigated and learned every detail of it. Although, our plan for LTR was ready to implement, we didn't have time to train the features because we found out that we need to implement PageRank from scratch 10 days before the submission. That's why all our efforts on LTR couldn't be reflected on our project but all that was needed was time for that.

TF-IDF
TF-IDF(Term Frequencyfi?!Inverse Document Frequency) is a widely used ranking algorithm in search engines. Search engines uses this algorithm to calculate the importance of query words in a document. It's the default ranking algorithm in Solr.

BM25
BM25(Best Matching 25) is a commonly used ranking algorithm by search engines to rank the documents by their relevance to the search query. In Solr, change the ranking algorithm to BM25 is easy. All we need to do is just modify the configuration. For this experiment, we've set k to 1.2 and b to 0.76.

PageRank
PageRank is a ranking algorithm that ranks the documents according to the relationships between web pages. According to Google, "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."

For this project, we implemented PageRank algorithm on our own and validated the algorithm with Nutch's built-in LinkRank algorithm. In the Nutch-Solr system, Nutch stores all the crawled web-pages and their scores in the Hadoop Sequence File format. While indexing, Nutch reads these stored data and feeds them to

Solr along with their scores. These passed scores will overwrite the ranked query responses in Solr. In other words, the built-in ranking algorithm in Solr will be overwrite by our PageRank.

To implement PageRank, we need the relations for all fetch web-pages. In the Nutch-Solr system, these data in only available in Nutch in the format of webgraph. Hence, we need to manipulate Nutch to implement PageRank. Fortunately, Nutch is a plug-in based system, it allows us to create additional class or plug-in and run it individually. Therefore, we created a new Java class for our PageRank and compiled it with Nutch to make it compatible with Nutch and Solr.

As mentioned earlier, the PageRank algorithm we implemented was based on Nutch's webgraph. In Nutch, webgraph is stored in few different parts: nodes, outlinks and inlinks. Each of them is an individual Hadoop Sequence file. Hence, we need to read all these file, transfer the stored data to usable format and run PageRank score calculation. Below is the pseudo code of our PageRank algorithm.

```
def invertLink(nodes, outlinks){
  inlinks = new HashMap()
  for node in nodes{
    // get all outlinks for a certain node
    nodeOutlinks = outlinks.get(node.url)
    for outlink in nodeOutlinks{
      inlink = copy(outlink)
      inlink.url = node.url
      inlink.score = node.score/nodeOutlinks.size
      inlinks[link.url].add
    }
  }
  return inlinks
}

def calculateScore(nodes, inlinks){
  for node in nodes{
    node.score = (1 - 0.85)/nodes.size
        // get all inlinks for a certain node
        nodeInlinks = inlinks.get(node.url)
    for link in nodeInlinks{
      node.score += 0.85 * inlink.score
    }
  }
}

def PageRank(webgraphdb){
  nodes = readNodes(webgraphdb)
  outlinks = readOutlinks(webgraphdb)
  setInitialScore(nodes, 1/nodes.size)
  for(i=0; i<10; i++){
    inlinks = invertLink(nodes, outlinks)
    calculateScore(nodes, inlinks)
  }
}
```

As the code shown, each node represents a webpage, outlinks is the set of all links grouped by web-pages. Inlinks is the inverted outlinks that represents the set of all inlinks grouped by web-pages as well. In outlinks, links in each group come from the same node. However, in inlinks, each group of inlinks points is to the same node. We updated the score for inlinks during the inverting and then calculate the score for each node based all its inlinks. The formula behind our algorithm is

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

.

We have compared out RageRank result with a modification of Nutch's LinkRank. According to Nutch's official document[5], LinkRank is similar to PageRank but ignores links from the same domain and same page by default. It calculate the score using the variant of PageRank formula

$$PR(p_i) = 1 - d + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

Our modification removed the ignorance of same domain or same pages. The formula remained the same as it won't affect the ranking although produces a different score. After the comparison, we believe our PageRank implementation is correct.

3.2 Experiment Subject and Setting

Considering the time that have been given for us to analyse UCL domain, since we have given the option to use CS subdomain (www.cs.ucl.ac.uk) inside UCL domain, we have chosen to crawl only CS subdomain. This decision was made based on the fact that we had a little amount of time to crawl the UCL domain considering also the limited resources that we have by the server that we have hired from Digital Ocean[11]. Before switch to the CS subdomain, we've fetched about 200,000 web-pages in UCL domain, which took over a week of time and 23GB of disk space. According to Google, UCL domain has more then 2,000,00 pages. Hence, the cost of fetching the entire UCL domain is not acceptable for us. Using the incomplete set of web-pages with in a certain domain could makes out search engine performs extremely worse than Google and UCL search engine. Considering all these aspects, we decided to crawl only CS subdomain to complete crawling in the expected time and try to implement a search engine that is aimed to have a good performance considering the evaluation metrics.

For testing the search engines, we have written few python scripts to fetch the search result from different search engines. We've also created script under different evaluation metrics for understanding which engine outperformed the others. The evaluation metrics will be explained in Section 3.3. These files calculated metrics are analysed for all of the explained ranking functions and other aspects explained in Section 3.1. In addition, Google and UCL's search engine are the other aspects that have been considered by these metrics. This way, we created an extensive analysis

Google	UCL	Solr
4,605	3,340	4,843

Table 1: Dataset Statistics

Name	Version
Nutch	1.12
Solr	6.5.0
lxml	3.7.3
BeautifulSoup4	4.5.3
requests	2.13.0
tabulate	0.7.7

Table 2: Library Versions

on all of the engines with different aspects to conclude a decision which outperforms the others.

We've preformed other steps before the evaluation. We tried to compare all search results with Google in our evaluation, however, we then realized the results from Google didn't handle redirection, which means, some pages from Google may actually points to the same page as out search engine even the urls are not the same. In order to solve this issue, we create another script to check all urls returned from Google and filter all 404 or 403 pages.

Statistics of the Dataset
During the experiment, we have fetched the UCL CS domain. We've filtered all files that bigger than 256KB, all images, all audio/video files, executable files and some source code files during the fetch step. Some large code repos and file servers within the CS domain were also ignored. In the end, 60,422 web-pages from CS domain were fetched before we couldn't find any new pages. Then, after removing duplications, 404 pages, redirections and other error pages from the fetched pages, we have indexed 28,389 web-pages to Solr.

The dataset used directly in the analysis is a group of JSON files, which stores all filtered urls and their ranks for our selected 56 queries words. These query words is divided into 6 sub-parts, which are words, phrases, people names, sentences, words/sentences with typos and words/sentences with special characters. The reason that we have this diversity of sub-parts is to test the search engine from different perspectives. It is not sufficient to test it only by querying words and/or sentences. There needs to be special cases to look for if the engine is capable of handling them and returning a reasonable solution.

For each of these words, we fetched 100 pages from each search engine or all results for those have less than 100 results. Details of these statistics are listed in Table 1.

Setting
In order to make everyone work on the same platform and fetch web-pages continuously, we created a Virtual Machine and use it as our development server on Digital Ocean. The virtual machines we've used cost $40 per month and has 4GB memory, 2 Core of 1.80GHz CPU and 60GB SSD storage. The server runs Ubuntu 16.04.2 LTS. We used OpenJDK 8 for Nutch and Solr, Python 2.7.13 for the scripts. Table 2 shows the version of all Java and Python libraries/packages used in the experiment. Due to the time limit, data used in the evaluation does not have stopwords and stemming enabled.

3.3 Metrics

For the testing part, we have used many metrics in order to have a more unbiased analysis. Since we do not have manually annotated relevant scores of web pages, we assumed that the top 100 web pages retrieved from Google or UCL search engine are the sets of all relevant web pages and the relevant score of a web page is 100 minus its position(starting from 0) in the relevant set.

After that, as being one of the most used metric for IR evaluation, precision and recall values are calculated. To have observe the effect of both precision and recall, f-measure calculation is implemented. Thanks to this value, we are able to consider both of the values. However, precision, recall and f measures are for unranked sets. To turn these measures into measures of ranked lists; we compute these measurements for the top k results with different values of k as well as plot the precision-recall curve.

Furthermore, to understand the number of non-relevant documents ranked before seeing the 1st relevant document, expected search length is calculated. In addition to that, reciprocal of the rank at which the first relevant document is retrieved is measured. This is calculated by means of expected search length. Then, average overlap value of two ranked list is implemented to measure the similarities of them. This calculation is enhanced by calculating ranked biased overlap(RBO)[13] of the lists.

Last but not least, to consider one of the most popular metric by having two assumptions of highly relevant documents are more useful than marginally relevant document and the lower the ranked position of a relevant document, the less useful it is for the user, since it is less likely to be examined; we have implemented normalised discounted cumulative gain(NDCG) of the lists.

3.4 Analysis of Results

The result of above mentioned metrics with different k value are presented in Figure 2, 3 and 4. For each metric, we compute the mean value among 56 selected query where vertical bars represent the standard error. For precision, as we can see, increasing the number of results will reduce the precision. As for recall, increasing the number of results will increase the precision. And comparing the results with Google and UCL search engine will result in a very similar recall curve when we employ TF-IDF. Comparing the result with Google will get better precision and recall when using BM25

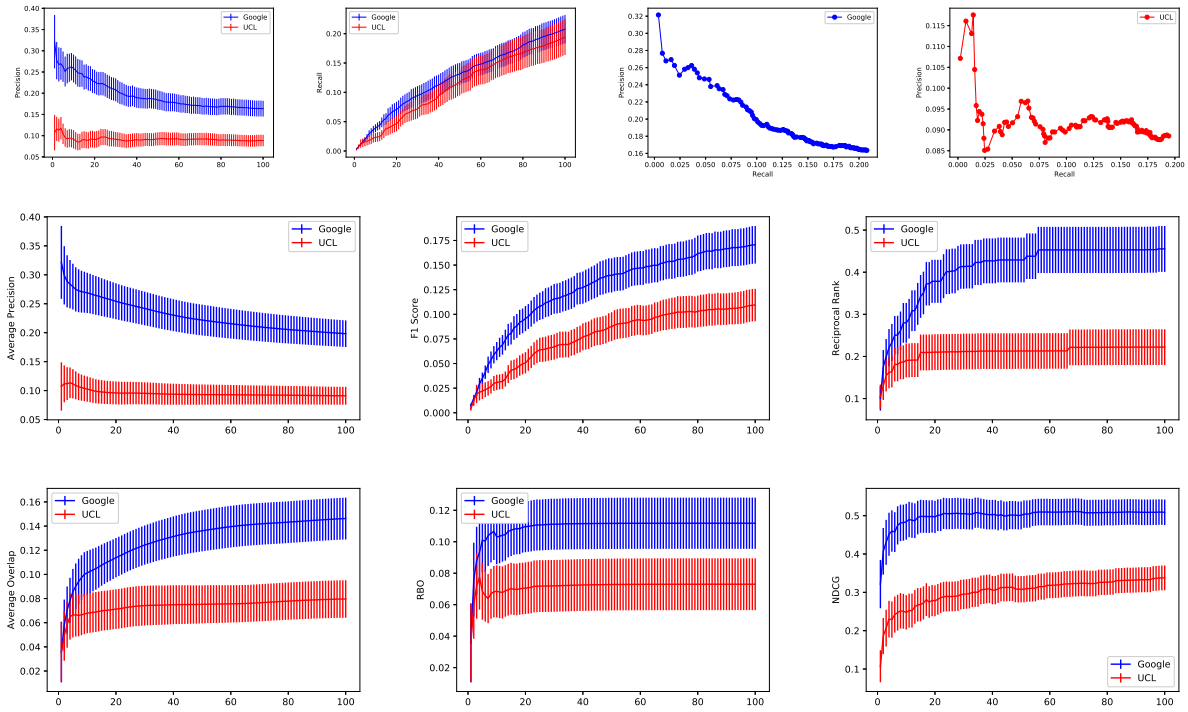

Figure 2: Evaluation of TF-IDF

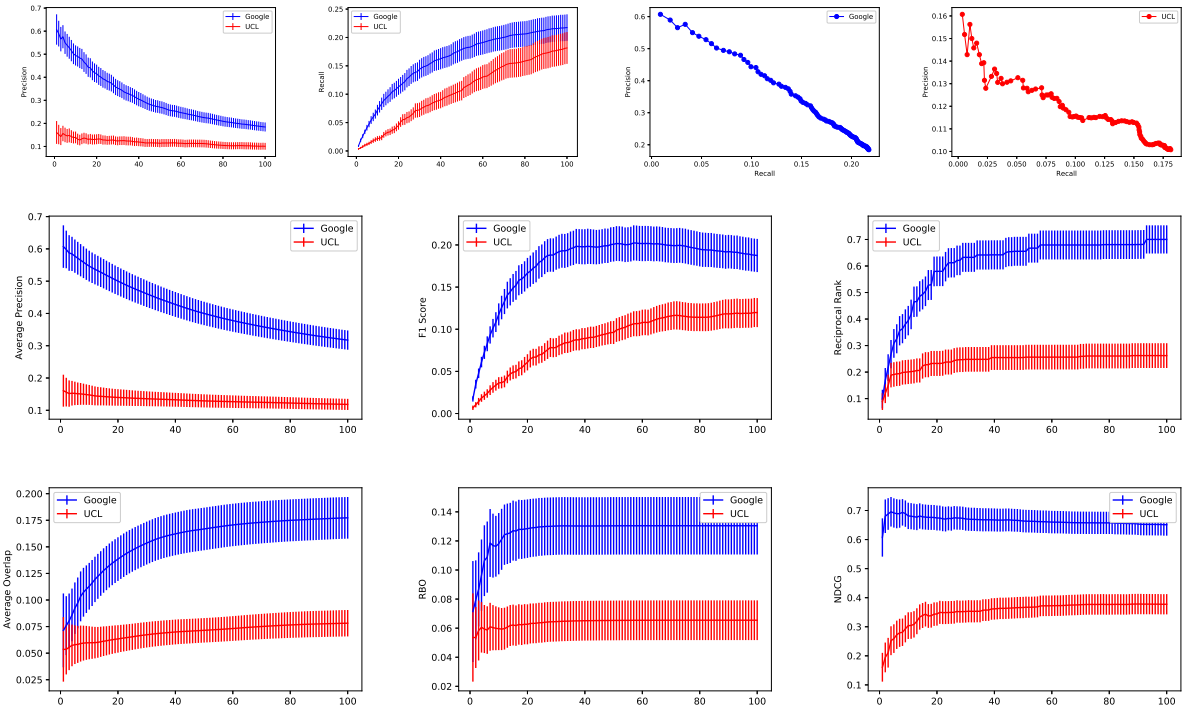

Figure 3: Evaluation of BM25(k1=1.2,b=0.76)

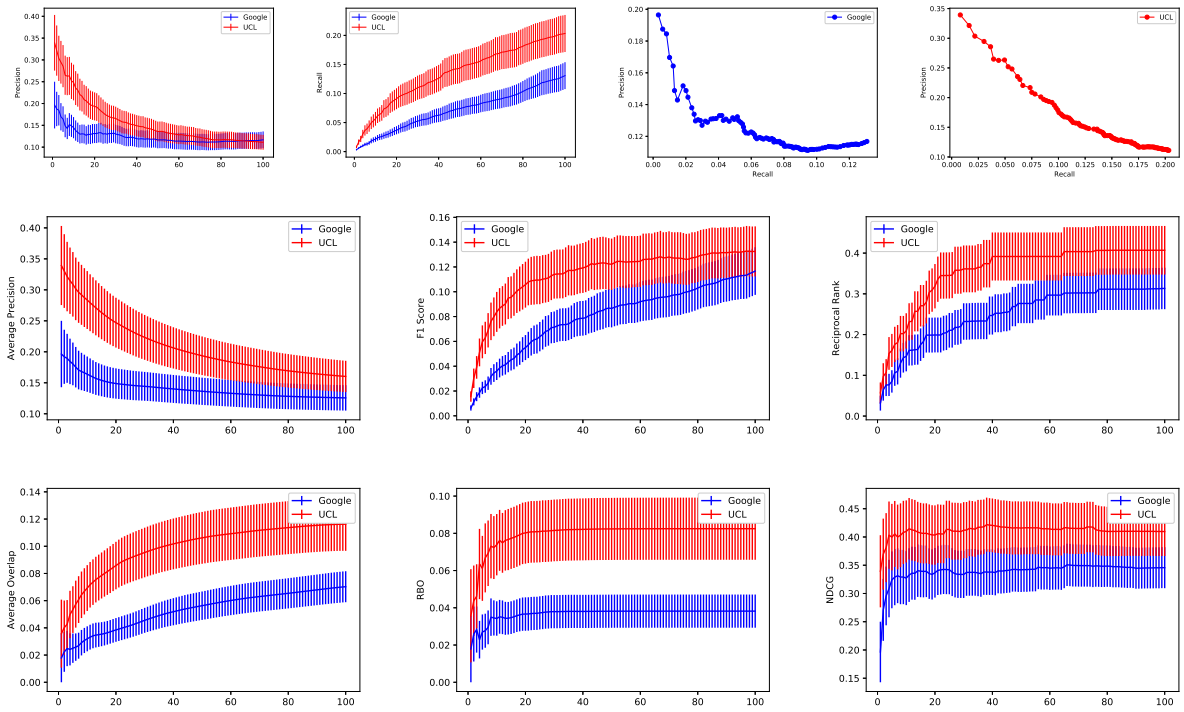

Figure 4: Evaluation of PageRank

or TF-IDF for ranking then comparing with UCL search engine whereas opposite results are obtained using PageRank. Among different ranking method, BM25 gives the best precision and recall curve when comparing the results with Google and PageRank gives the best precision and recall curve when comparing the results with UCL search engine. Hence, BM25 outperforms the other methods during comparisons with Google's results in both average precision and F1 score, however, PageRank outperforms the other methods during comparisons with results from UCL search engine. Similarly, Same result can be concluded for reciprocal rank, average overlap, RBO and NDCG. A summary of the average values for different curves is given in Table 3.

	BM25	PageRank	TF-IDF
Average Overlap (Google)	0.15363	0.05281	0.12831
Average Overlap (UCL)	0.08285	0.09902	0.07400
Average Precision (Google)	0.27870	0.14071	0.22929
Average Precision (UCL)	0.10817	0.20833	0.09499
F1 (Google)	0.14357	0.07971	0.12686
F1 (UCL)	0.08498	0.11438	0.07831
NDCG (Google)	0.55361	0.33899	0.49939
NDCG (UCL)	0.32778	0.41075	0.30154
Precision (Google)	0.23133	0.12369	0.196075
Precision (UCL)	0.10263	0.15728	0.09180
RBO (Google)	0.13217	0.03681	0.10906
RBO (UCL)	0.08145	0.07958	0.07155
Recall (Google)	0.13431	0.07161	0.12518
Recall (UCL)	0.11412	0.13505	0.11062
Reciprocal Rank (Google)	0.47919	0.24992	0.40749
Reciprocal Rank (UCL)	0.23941	0.35475	0.20978

Table 3: Summary of Results

4 DISCUSSION AND LIMITATIONS

According to the results of the metrics, it is reasonable to consider our search engine as a well performing web search tool. In the limited amount of time, it is not possible to analyse the satisfaction of the user when they are performing a search, which is the ultimate evaluation metric that should be used as a performance indicator. However, we have used the most commonly used evaluation metrics in IR community and the analysis of the results show that the search engine we have built performs considerably similar to Google when using BM25 ranking factor. Considering the fact that Google changes its ranking algorithm regularly, we believe that PageRank is not the main ranking factor in Google's existing ranking algorithm.

Another thing worth discussing is the PageRank result for our search engine is close to UCL search engine, which uses a variant of BM25 for its ranking. But our BM25 ranking results are not close to UCL search engine. A possible reason is UCL search engine uses the whole UCL domain, therefore BM25 won't produce same score.

The most significant two limitations that we have faced are time and resources. If we had more time and computing power, we might be able to implement better ranking functions and more optimised search functions for web search. In addition, we could easily use Learning to Rank(LTR) by training the features data to obtain a better performing ranking function. Since we didn't have a significant time to implement the search engine and we had small amount of funds to hire the server, our end product had the limitation of performing not as well as the compared search engines.

5 CONCLUSION

In conclusion, to make search results more similar to Google's results, BM25 should be applied for ranking while searching in the UCL CS domain using the settings in this paper. On the other hand, to make the results more similar to search results from the UCL search engine, PageRank should be applied. However, since we only calculate the PageRank scores within the UCL CS domain, link relations cannot be fully captured and the PageRank scores cannot reflect the precise importance of web pages. Similarly, the BM25 and TF-IDF score will also vary if we use a larger document collection such as the entire UCL domain. Moreover, since all web-pages we fetched are from the same domain, loads of internal link would exists. Hence, PageRank won't work very well in this environment.

In our future work, we will build the search engine for the entire UCL domain as well as implement LTR as mentioned before. Last but not the least, we will also investigate the impacts of stop-words removal and stemming in the future as we disabled both of them in our experiment.

REFERENCES

[1] Abbas Ali. 2011. *Sphinx Search Beginner's Guide.* Packt Publishing.
[2] Apache. *Apache Hadoop.* http://hadoop.apache.org/
[3] Apache. *Apache Nutch.* http://nutch.apache.org/
[4] Apache. *Apache Solr.* http://lucene.apache.org/solr/
[5] Apache. *NewScoring.* https://wiki.apache.org/nutch/NewScoring
[6] Funnelback. *Funnelback.* https://www.funnelback.com/
[7] Funnelback. *Funnelback Ranking Algorithms.* https://docs.funnelback.com/15.10/develop/reference-documents/funnelback-ranking-algorithms.html
[8] Google. *Google.* https://www.google.com/
[9] Ken Hillis, Michael Petit, and Kylie Jarrett. 2012. *Google and the Culture of Search* (1st ed.). Routledge, New York, NY, 10001.
[10] Shahin Mirhosseini, Guido Zuccon, Bevan Koopman, Anthony Nguyen, and Michael Lawley. 2014. Medical Free-Text to Concept Mapping As an Information Retrieval Problem. In *Proceedings of the 2014 Australasian Document Computing Symposium (ADCS '14).* ACM, New York, NY, USA, Article 93, 4 pages. DOI: http://dx.doi.org/10.1145/2682862.2682880
[11] Digital Ocean. *Digital Ocean.* https://www.digitalocean.com/
[12] Ian Rogers. 2002. The Google Pagerank algorithm and how it works. (2002).
[13] William Webber, Alistair Moffat, and Justin Zobel. 2010. A similarity measure for indefinite rankings. *Acm Transactions Information Syst Tois* 28, 4 (2010), 20. DOI: http://dx.doi.org/10.1145/1852102.1852106
[14] Xapian. *The Xapian Project.* https://xapian.org/
[15] Jun Xu and Hang Li. 2007. AdaRank: A Boosting Algorithm for Information Retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '07).* ACM, New York, NY, USA, 391–398. DOI: http://dx.doi.org/10.1145/1277741.1277809