Custom Web Search Engine

Emre Can Kucukoglu E1746239@CENG.METU.EDU.TR
Middle East Technical University

Fatih Hafizoglu E1746049@CENG.METU.EDU.TR
Middle East Technical University

Yigit Ilguner E1560762@CENG.METU.EDU.TR
Middle East Technical University

Abstract

When a user looks for a specific topic under a certain domain via the popular web search engines, less authorized and lower relevant results can be displayed at a higher rank than the pages which include more related information. Mostly, decision of the first page to look at and to start the search is not left to end user. Thus, user has no other option but get the same results for the same query. In order to search in a smaller but more relevant collection of documents, a web search engine that asks its user's opinion for crawled pages and provides alternative methods for searching is developed. Besides the basics of web search like crawling, indexing and query processing, this system includes higher level operations such as result diversification and query optimization.

1. Introduction

Popular web search engines commonly do not provide customization feature for parameters like search domain, initial web page to crawl, maximum number of distant pages reach from the starting point. To illustrate, when a user wants to look for recipes from such masterchefs like Gordon Ramsey, Anthony Bourdain and Jamie Oliver, it is required to give their names explicitly in the search query. Although this kind of query increases the relevance, it causes a tackle for subtopic coverage. In order to overcome this obstacle, a user-friendly and customizable search engine is designed and developed. This engine is a web-based application which gives user flexibility in terms of collection domain, limit for the amount for visited pages and traverse depth. Moreover, it gives options for faster query processing and diversifying the search results.

.

In this project, basics of the web search such as crawling, creating an inverted index, query processing, and search result diversification are investigated and applied thoroughly. Because of the resource restrictions, we are able to constract a single site web search engine.

2. Background

2.1. Web Crawling

The basic web crawling(Manning et al., 2009) operation (whether for the Web, an intranet or other hypertext document collection) is handled as follows. The crawler begins with one or more URLs that constitute a seed set. It picks a URL from this seed set, then fetches the web page at that URL. The retrieved page is then parsed, to extract both the text and the links from the page (each of which points to another URL). The extracted text is fed to a text indexer. The extracted links (URLs) are then added to a URL frontier, which at all times consists of URLs whose corresponding pages have yet to be fetched by the crawler. Initially, the URL frontier contains the seed set; as pages are fetched, the corresponding URLs are deleted from the URL frontier. The entire process may be viewed as traversing the web graph. In continuous crawling, the URL of a fetched page is added back to the frontier for fetching again in the future.

2.2. Indexing

Downloaded documents are case folded, stemmed and tokenized into their terms by the system. Secondly, the inverted index is constructed incrementally as the collection is expanded. An inverted index consists of term dictionary and posting lists for every term. If a term occurs in a document in the collection, this document is added to the posting list of the term. Moreover, term frequency and document frequency of the term is updated. Inverted index speeds up the query processing by giving a rapid access over the number of term occurances for each document(in the posting list) in the entire collection.

2.3. Query Processing

When a user initiates a search, query is divided into its terms by using query splitting and term normalization methods. Secondly, postings belong to the query terms are retrieved. Each posting includes document identifier and number of term occurences in that document. Relevance score between the query and the document in which query term appears is calculated. Query processing can be done as either disjunctive(OR) or conjuctive(AND) way through query terms. Aggregation of the similarity scores between a document and the terms in the query provides the query-document relevance score for that document. After each document is scored, the highest top-k documents are chosen. Multiple ordering approach (i.e., re-ranking the top m retrievals of the previous ranking) can be preferred. Since computing document-query scores for millions, even for thousands of document utilizes high level of process power, dynamic pruning algorithms for postings are commonly used. Document-at-a-time and term-at-a-time(Fontoura et al., 2011) are the highly preferred approaches among web search engine architectures.

2.3.1. Document-At-A-Time

In Document-At-A-Time(DAAT), the query term postings lists are processed in parallel, such that all postings of document d_j are considered before scoring commences on d_{j+1} . There are two main factors in evaluating the performance of the DAAT algorithms:

- The index access cost
- The scoring cost.

In the case of the naive DAAT algorithm, every posting for every query term must be accessed. The index access cost is then proportional to the sum of the sizes of the postings list for all query terms. The scoring cost includes computing the scoring function and updating the result heap. As most of the DAAT algorithms have been designed for disk-based indexes, they try to minimize the index access cost by skipping parts of the postings list.

2.3.2. Term-At-A-Time

In Term-At-A-Time scoring, the query term posting lists are processed and scored in sequence, so that documents containing term t_i gain a partial score before scoring commences on term. The contributions from each query term to the final score of each document must be stored in vector of accumulators V_A. The size of the vector is the number of documents in the index (N). For dot product scoring, the score contributions for each term can be independently computed and added to the appropriate documents in the vector. Every posting for every term is required to be accessible. For each posting, its score contribution is calculated and added it to the vector V_A. When processing term t, a score contribution is found:

$$V_A[d] \leftarrow V_A[d] + d_t * q_t \tag{1}$$

for every posting in ts postings list, where d_t is the document weight in the posting for document d and q_t is the query weight for term t. The vector must be initialized to zero in the beginning of the query execution. After processing all terms in the query, the k entries in V_A with the highest value are top-k documents that should be returned as the query results

2.4. Result Diversification

Queries submitted to web search engines are often ambiguous or multi-faceted in the sense that they have multiple meanings or subtopics. For ambiguous queries, a typical example is the query *jaguar* that can refer to several interpretations including a kind of animal, a car brand, a type of cocktail, an operating system. Multi-faceted queries are even more commonly seen in practice; to illustrate, for the interpretation *jaguar car* of the query *jaguar*, a wide range of subtopics may be covered: models, prices, history of the company. For such queries what the searchers underlying information need is ambiguous, because of a lack of context.

3. System Architecture and Design

The organization of the whole system and task of each component is shown in Figure 1

3.1. Data

Document corpus is constructed dynamically by beginning at a seeded web page and traversing through user specific number of levels. Instead of looking at a fixed collection, resizable corpus is decided for obtain more precise findings. Crawler object traverse through web pages and extract the text content from page. We use JTidy[1] in order to parse the HTML tags and obtain the textual data from the document.

[1]http://jtidy.sourceforge.net

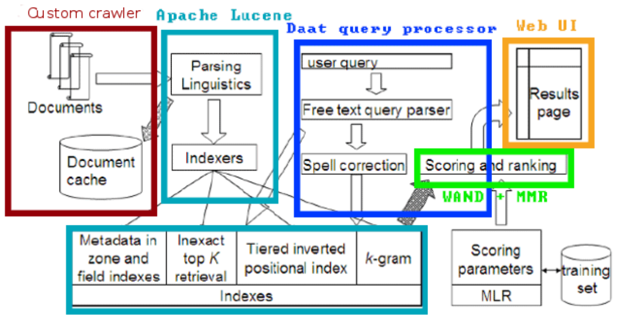


Figure 1. Complete organization of the components of Custom Web Search Engine

3.2. Index

After crawling phase has completed, crawled websites will be given as a collection of Documents to Apache Lucene[2] indexer. In Lucene, a Document is a collection of Fields. A Field is content along with metadata describing the content. These fields will be used for fielded searching (e.g. title, author) in Section 3.3. Fielded search will be presented to user as an query option. All cases will be lowered. To split the words, whitespace tokenizer will be used.

3.3. Query Processing

Vector space model is selected as query processing model. For ranking phase, we use WAND optimization over TFx-IDF weights of documents is applied. In WAND approach, documents that satisfy a minimum score are stored in a heap. Pointers are assigned for each posting list of the query terms. At a time t, document identifiers of currently analyzed postings from every pointer are ranked in an ascending order. We look for the pivot document whose score after adding the preceding ones exceeds the heap threshold. If a document includes enough query terms to predominate the limit of the heap, then the document is eligible to be picked. Afterwards, the element that has the minimum TFxIDF score in the heap is removed and ordering in the list is updated by scores. Unless, the document does not reach enough score, then current postings that show the smaller identifiers are iterated until their document IDs become greater than or equal to pivot documents. After processing every posting, documents in the heap give the most relevant result for the query.

3.3.1. WAND Processing

WAND(Broder et al., 2003) is a DAAT variation which aims to reduce the query-document score computation time by prununing posting list by a decision mechanism. WAND maintains the current top-k documents and a threshold equal to their minimum score. For any new document, WAND calculates an approximate score, summing up the upper bounds for the terms occurring in the document. If this approximate score is greater than the current threshold, then the document is fully scored. It is then inserted in the top-K candidate document set if this score is greater than the current threshold, and the current threshold is updated. If the approximate score check fails, the next document is processed.

The set of posting lists for the query terms are maintained in increasing order of the document identifier that each posting list currently refers to. Then a pivot term is computed, that is, the first term for which the accumulated sum of upper bounds of preceding terms and itself exceeds the current threshold. The corresponding document identifier in the posting list of the pivot term identifies the pivot document, that is, the smallest document identifier having a chance to overcome the current threshold. If the current document identifiers of the previous terms are equal to the pivot document document identifier, the document will be fully scored. Otherwise, the posting list of one of the preceding terms is moved to the pivot document document identifier, and the procedure is repeated. If a good candidate document is found, then full TFxIDF score for the (query,document) pair is calculated.

After the selection of top-k documents by WAND, those results are picked as their coverage of different query subtopics by exploiting a diversification method named Maximum Marginal Relevance.

3.3.2. Maximum Marginal Relevence

According to the Maximum Marginal Relevence(MMR) method (Carbonell & Goldstein, 1998), a document d is selected for inclusion in a ranked list of documents for a given query q such that where S is the set of documents that

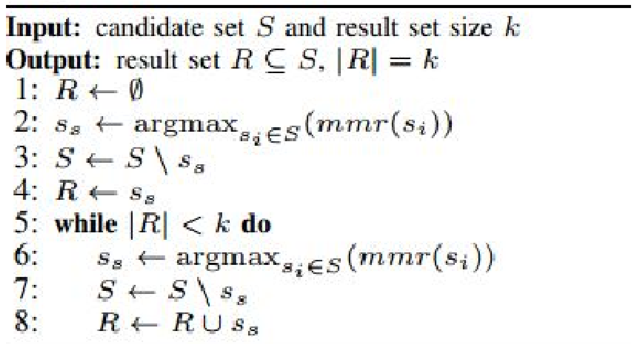
$$d = \arg\max_{d_i \in R} [\lambda \cdot sim_1(d_i, q) - (1-\lambda) \cdot \max_{d_j \in S} sim_2(d_i, d_j)]$$

Figure 2. MMR formula

have been selected so far and R is the set of candidate documents to be selected; sim_1 is the similarity between query and document and sim_2 is the similarity between two documents. For sim_1 and sim_2 cosine similarity is used as the similarity metric. MMR algorithm as a complete is given below.

Hence Maximal Marginal Relevance algorithm considers the tradeoff between relevance and diversity, it guarantees that result set is as diverse as possible. From the algorithm

[2]https://lucene.apache.org/



```
Input: candidate set $S$ and result set size $k$
Output: result set $R \subseteq S$, $|R| = k$
1: $R \leftarrow \emptyset$
2: $s_s \leftarrow \text{argmax}_{s_i \in S}(mmr(s_i))$
3: $S \leftarrow S \setminus s_s$
4: $R \leftarrow s_s$
5: while $|R| < k$ do
6:     $s_s \leftarrow \text{argmax}_{s_i \in S}(mmr(s_i))$
7:     $S \leftarrow S \setminus s_s$
8:     $R \leftarrow R \cup s_s$
```

Figure 3. Complete MMR Algorithm

3, in line 6, from each candidate set of results is chosen according to its MMR function result. Weight for the similarity with the query and weight for the diversification among the documents are shared equally, hence tradeoff parameter α is set to 0.5.

3.4. User Interface

Our web-based user interface gives opportunity to decide the initial web site and to adjust the parameters such as the maximum number of web pages investigated through, farthest page distance from the starting point for crawling and indexing. After the indexing stage is finished, search query is ready to enter. Besides being sorted, we show user the diverse and relevant results by exploiting the methods described in Section 3.3. Since supporting wildcard queries and snippets are not the aim of this project, these feature are not included. Result set will be sorted according to their rankings. Ruby on Rails framework is selected for the web based communication.

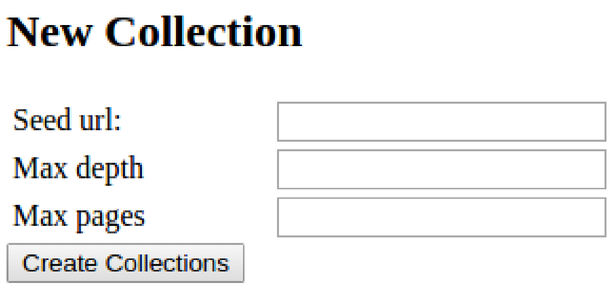
Workflow of the system for a custom search is described below:

1. User enters the seed URL, sets maximum depth and maximum number of pages that will be retrieved by the crawler (Figure 4).

2. After user initiates the search, crawler start to travel from the given web page and traverses the web as a breadth first-like manner.It downloads the available documents, get the textual information from them and stores as a raw text document in the server for indexing.

3. Lucene initiates the indexing procedure for the collection. Tokenizing the words, generating both the dictionary and the positing lists is handle by Lucene.

4. If inverted index is updated successfully for the current collection, user is redirected to the search page.

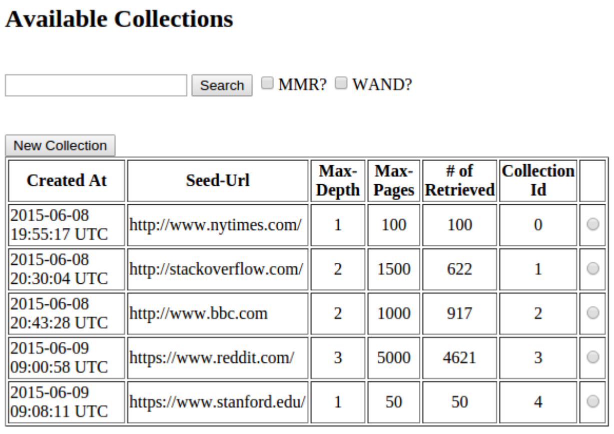


New Collection

Seed url:
Max depth
Max pages

Figure 4. Page for adding New Collection

In this page, user is able to select the query processing type either naive DAAT or WAND, and whether the diversified results is found or not as a result of the query (Figure 5).



Available Collections

Created At	Seed-Url	Max-Depth	Max-Pages	# of Retrieved	Collection Id	
2015-06-08 19:55:17 UTC	http://www.nytimes.com/	1	100	100	0	○
2015-06-08 20:30:04 UTC	http://stackoverflow.com/	2	1500	622	1	○
2015-06-08 20:43:28 UTC	http://www.bbc.com	2	1000	917	2	○
2015-06-09 09:00:58 UTC	https://www.reddit.com/	3	5000	4621	3	○
2015-06-09 09:08:11 UTC	https://www.stanford.edu/	1	50	50	4	○

Figure 5. Search Page

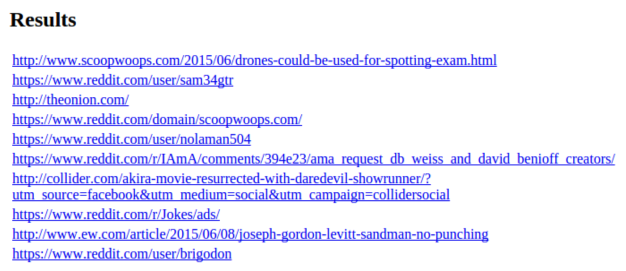
5. According to the search parameters determined by user, system handles the query processing and the diversification steps.

6. Top-10 ranked search results are shown with their URLs (Figure 6).

Components of custom web search engine, except the user interface are developed with Java programming language. Source codes can be download from `https://github.com/eckucukoglu/custom-web-se`.

4. Conclusion

Although increasing the precision is one of the major concerns of web search engine design. Providing the freedom



Results

http://www.scoopwoops.com/2015/06/drones-could-be-used-for-spotting-exam.html
https://www.reddit.com/user/sam34gtr
http://theonion.com/
https://www.reddit.com/domain/scoopwoops.com/
https://www.reddit.com/user/nolaman504
https://www.reddit.com/r/IAmA/comments/394e23/ama_request_db_weiss_and_david_benioff_creators/
http://collider.com/akira-movie-resurrected-with-daredevil-showrunner/?utm_source=facebook&utm_medium=social&utm_campaign=collidersocial
https://www.reddit.com/r/Jokes/ads/
http://www.ew.com/article/2015/06/08/joseph-gordon-levitt-sandman-no-punching
https://www.reddit.com/user/brigodon

Figure 6. Result Page

to user for configuring the search domain, subtopic coverage level, and relaxation of search results are unnegligible design choices. In this project, we implement a web search engine which gives user a configurable query search. Furthermore, it is capable of optimizing query processing and diversifing the search results. We apply state-of-the-art query processing and diversification methods. Due to financial restrictions, our system runs on a single server and uses single indexer and query processor. In order to increase the speed of the system, a distributed approach is required. Moreover, pruning the postings by according to some heuristics and caching the web search result pages and posting lists will reduce the latency. As a future work, these two design choices are going to be investigated. Additionally, dynamically loaded web pages are unable to crawl and to extract its content with our crawler. A solution for this issue is going to be provided in the following version.

References

Broder, A. Z., Carmel, D., Herscovici, M., Soffer, A., and Zien, J. Efficient query evaluation using a two-level retrieval process. *CIKM'03*, 3(11):426–434, 2003.

Carbonell, J. and Goldstein, J. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *In Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '98)*, pp. 335–336, New York, 1998. ACM Press.

Fontoura, M., Josifovski, V., Liu, J., Venkatesan, S., Zhu, X., and Zien, J. Evaluation strategies for top-k queries over memory-resident inverted indexes. *The 37th International Conference on Very Large Data Bases*, 4(12): 426–438, 2011.

Manning, C. D., P.Raghavan, and Schtze, H. *An Introduction to Information Retrieval*. Cambridge University Press, draft edition, 2009.