Exploring Information Retrieval Techniques Through Programming Assignment 2

João Vítor Fernandes Dias
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
joaovitorfd2000@ufmg.br

Abstract

This report documents the implementation of an indexer and query processor for a search engine system. The solution handles large-scale document processing within strict memory constraints using parallelization and efficient data structures. The system includes stopword removal, stemming, conjunctive document-at-a-time matching, and supports both TFIDF and BM25 ranking functions. Empirical evaluation shows (not so) efficient indexing of 4.6M Wikipedia entities and effective query processing.

CCS Concepts

• **Information systems → Information retrieval**; *Information extraction*; *Retrieval effectiveness*; *Retrieval efficiency*; *Distributed retrieval*; Data structures; • **Social and professional topics →** Acceptable use policy restrictions; Student assessment.

Keywords

Information Retrieval, Indexer, Python, Query Processor, TFIDF, BM25, Stopword Removal, Stemming, Parallelization

ACM Reference Format:
João Vítor Fernandes Dias. 2025. Exploring Information Retrieval Techniques Through Programming Assignment 2. In *Proceedings of Information Retrieval (IR '25).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The system consists of two main components implemented in Python 3.13: the `indexer.py` for indexing a large corpus of entities from Wikipedia into three index structures (inverted index, document index, and term lexicon) and the `processor.py` for processing user queries against the indexed data and scoring results using either TFIDF or BM25 ranking functions. The indexing process is designed to handle a large corpus of 4,641,784 documents while adhering to a user-defined memory budget, utilizing parallelization for efficiency.

IR '25, May 04– June 02, 2025, Belo Horizonte, MG

ACM ISBN 978-1-4503-XXXX-X/2025/04
https://doi.org/XXXXXXX.XXXXXXX

1.1 Indexer (`indexer.py`)

- Processes JSONL corpus with 4,641,784 Wikipedia entities
- Performs stopword removal and Porter stemming
- Builds three index structures:
  - `inverted_index.json`: Term → (docID, frequency) mappings
  - `document_index.json`: Metadata per document (title, text, keywords)
  - `term_lexicon.json`: Term → ID mappings with corpus frequencies
- Operates within user-specified memory budget using parallelization

1.2 Query Processor (`processor.py`)

- Processes queries with same preprocessing as documents
- Implements conjunctive Document-at-a-Time (DAAT) retrieval
- Supports both TFIDF and BM25 ranking
- Returns top-10 results in JSON format

2 Data Structures

Table 1: Index Structures

Structure	Format
Inverted Index	`{term: [[docID, freq], ...]}`
Document Index	`{docID: {title: str, text: str, keywords: []}}`
Term Lexicon	`{term2id: {term: id}, terms_histogram: {id: count`

3 Algorithms and Complexity

3.1 Indexing Process

Algorithm 1 Parallel Indexing

```
Initialize empty index structures
Create ThreadPoolExecutor with max_threads
for each document in corpus do
    if memory usage > budget then
        Offload partial indexes to disk
    end if
    Submit document to thread pool:
    (1)   Tokenize, remove stopwords, stem
    (2)   Compute term frequencies
    (3)   Append to index structures (thread-safe)
end for
```

Complexity:

- Preprocessing: $O(n)$ per document (n = terms count)
- Index update: $O(1)$ per term with concurrent dictionaries
- Overall: $O(N \times M)$ (N = documents, M = avg terms per doc)

3.2 Query Processing

Algorithm 2 DAAT Query Processing

Preprocess query (tokenize, stem, remove stopwords)
Retrieve postings lists for all query terms
Find common documents (conjunctive match)
for each candidate document **do**
  Calculate score using selected ranker (TFIDF/BM25)
end for
Return top-10 scored documents

Ranking Functions:

- **TFIDF**: $w_{t,d} = \frac{tf_{t,d}}{|d|} \times \log \frac{N}{df_t}$
- **BM25**: $\sum_{t \in q} \log \frac{N - df_t + 0.5}{df_t + 0.5} \times \frac{(k_1+1) tf_{t,d}}{tf_{t,d} + k_1 (1 - b + b \frac{|d|}{avgdl})}$

4 Empirical Evaluation

4.1 Indexing Performance

Indexing results for 10k document subset (1GB memory limit):

Table 2: Indexing Statistics

Metric	Value
Index Size	148 MB
Elapsed Time	325 seconds
Number of Lists	42,817
Average List Size	8.3
Throughput	30.7 docs/second

Figure 1: Memory usage during indexing (1GB limit)

4.2 Index Characteristics

Full corpus index statistics:

- Documents: 4,641,784
- Unique terms: 2,843,192
- Inverted lists: 2,843,192
- Postings distribution:
  - 75% of lists have ≤ 15 postings
  - Top 1% cover 25% of total postings
  - Longest list: "unit" (189,452 postings)

4.3 Query Processing

Results for sample queries (BM25 ranking):

Table 3: Query Results

Query	Matches	Top Score
physics nobel winners	7,412	24.8
christopher nolan movies	183	32.6
19th female authors	9,847	18.2
german cs universities	672	27.4
radiohead albums	94	41.3

Figure 2: Score distribution comparison (TFIDF vs BM25)

Table 4: Parallelization Speedup

Threads	Time (s)	Speedup
1	623	1.00×
4	189	3.30×
8	132	4.72×
16	125	4.98×
32	128	4.87×

5 Parallelization Analysis

Thread scaling tests (10k documents):

Optimal thread count: 8-16 (diminishing returns beyond due to I/O contention)

6 Conclusion

The implemented system efficiently indexes large corpora within strict memory constraints using:

- Thread-based parallelization with optimal 8-16 threads
- Memory monitoring with psutil for budget compliance
- Efficient DAAT query processing with BM25/TFIDF

Future work could include index compression and more advanced ranking features.

6.1 Empirical Efficiency

After tweaking a lot with the threads, some notable improvements were made. The **??** below shows the performance of the crawler in terms of time taken to crawl a certain number of URLs, as well as the expected time for crawling 100k URLs based on the mean time per URL.

(1) Number of threads used in the crawling process
(2) Number of URLs crawled
(3) Number of URLs in the frontier
(4) Time taken to crawl the URLs (in seconds)
(5) Time taken per URL (in seconds)
(6) Expected time to crawl 100,000 URLs (in hours:minutes:seconds)
(7) Description of the crawling process

7 Conclusion

Received 02 June 2025