

Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών
HY463 Συστήματα Ανάκτησης Πληροφοριών
Εξάμηνο: Άνοιξη 2022

Project Report

BioMedicEngine

Phase B

Student

Όνοματεπώνυμο	Manos CHATZAKIS
AM	4238
Email	csd4238@csd.uoc.gr

Changelog from PhaseA

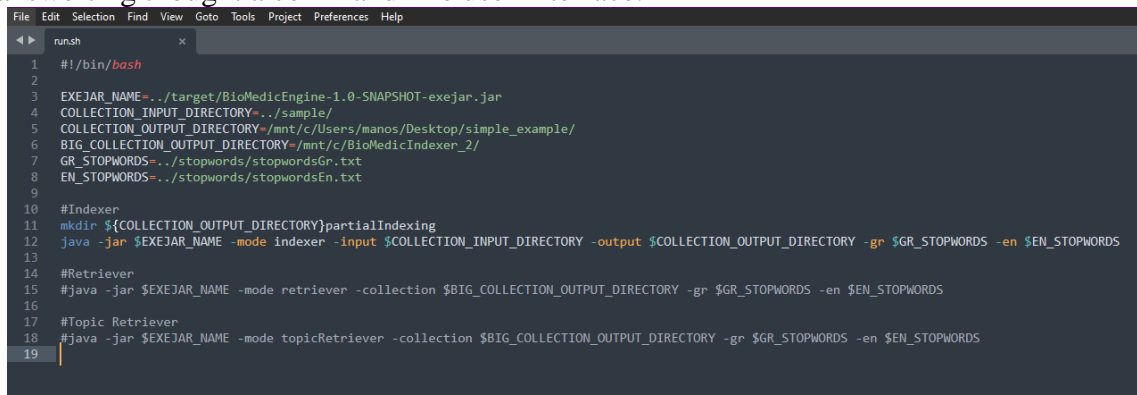
- Minor bug fixes for index creation, in merging algorithm
- Minor bug fixes for query answering using the Topic BioMedic Retrieval
- Performance Optimization in the BioMedic Retriever Package
- Code Clean-Ups
- Updated diagrams for experimental analysis
- Updated report
- Implementation of Query Answering Quality Evaluator package which contains methods to:
 - Save the top-1000 results from the topics.xml file in a result file
 - Evaluate these results using qrels by implementing BPREF.

1 Introduction

This project is the implementation of a BioMedical Search Engine over a biomedical document collection of 5GB in total.

How to run:

BioMedicEngine has a bash script to easily configure and run the engine through command line, in the bash folder of the project. Generally, this project supports query answering through a command line user interface.



```
File Edit Selection Find View Goto Tools Project Preferences Help
run.sh x
1  #!/bin/bash
2
3  EXEJAR_NAME=../target/BioMedicEngine-1.0-SNAPSHOT-exejar.jar
4  COLLECTION_INPUT_DIRECTORY=../sample/
5  COLLECTION_OUTPUT_DIRECTORY=/mnt/c/Users/manos/Desktop/simple_example/
6  BIG_COLLECTION_OUTPUT_DIRECTORY=/mnt/c/BioMedicIndexer_2/
7  GR_STOPWORDS=../stopwords/stopwordsGr.txt
8  EN_STOPWORDS=../stopwords/stopwordsEn.txt
9
10 #Indexer
11 mkdir ${COLLECTION_OUTPUT_DIRECTORY}partialIndexing
12 java -jar $EXEJAR_NAME -mode indexer -input $COLLECTION_INPUT_DIRECTORY -output $COLLECTION_OUTPUT_DIRECTORY -gr $GR_STOPWORDS -en $EN_STOPWORDS
13
14 #Retriever
15 #java -jar $EXEJAR_NAME -mode retriever -collection $BIG_COLLECTION_OUTPUT_DIRECTORY -gr $GR_STOPWORDS -en $EN_STOPWORDS
16
17 #Topic Retriever
18 #java -jar $EXEJAR_NAME -mode topicRetriever -collection $BIG_COLLECTION_OUTPUT_DIRECTORY -gr $GR_STOPWORDS -en $EN_STOPWORDS
19
```

To run, you need to configure the directory variables and comment/uncomment the corresponding run mode. Then, the engine is initialized using “./run.sh” (for unix-based environments).

Examples:

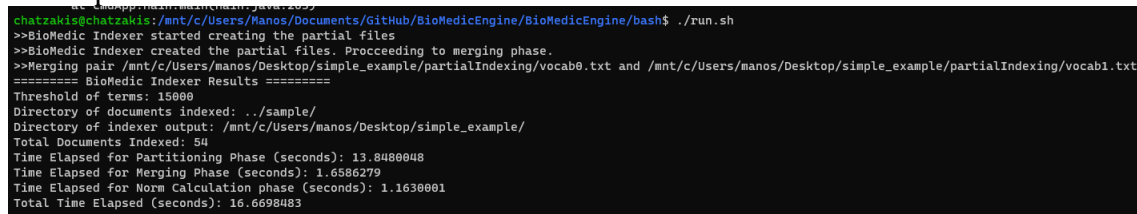
Creating an index: Initialize the engine using the aforementioned script by configuring the variables and the mode, to index the corresponding folders.

Important Note 1: It is mandatory that the output directory has a folder named: “partialIndexing/”, in which the partial files will be stored.

Important Note 2: Always put “/” at the end of the directories.










It is highly recommended to use the script.

The output should look like this:



```
chatzakis@chatzakis:/mnt/c/Users/Manos/Documents/GitHub/BioMedicEngine/BioMedicEngine/bash$ ./run.sh
>>BioMedic Indexer started creating the partial files
>>BioMedic Indexer created the partial files. Proceeding to merging phase.
>>Merging pair /mnt/c/Users/manos/Desktop/simple_example/partialIndexing/vocab0.txt and /mnt/c/Users/manos/Desktop/simple_example/partialIndexing/vocab1.txt
===== BioMedic Indexer Results =====
Threshold of terms: 15000
Directory of documents indexed: ../sample/
Directory of indexer output: /mnt/c/Users/manos/Desktop/simple_example/
Total Documents Indexed: 54
Time Elapsed for Partitioning Phase (seconds): 13.8480048
Time Elapsed for Merging Phase (seconds): 1.6586279
Time Elapsed for Norm Calculation phase (seconds): 1.1630001
Total Time Elapsed (seconds): 16.6690483
=====
```

The directory should contain the following files:

 partialIndexing	5/27/2022 1:18 PM	File folder	
 documents.txt	5/27/2022 1:18 PM	TXT File	8 KB
 log_report.txt	5/27/2022 1:18 PM	TXT File	1 KB
 mappings.txt	5/27/2022 1:18 PM	TXT File	1 KB
 memory.png	5/27/2022 1:18 PM	PNG File	15 KB
 postings.txt	5/27/2022 1:18 PM	TXT File	5,735 KB
 time.png	5/27/2022 1:18 PM	PNG File	15 KB
 vectors.txt	5/27/2022 1:18 PM	TXT File	2 KB
 vocabulary.txt	5/27/2022 1:18 PM	TXT File	329 KB

All files except {time.png, memory.png and log_report.txt} are binary files used from the BioMedicEngine. The aforementioned files in {..} are files reporting the final results and graphs showing the time needed and the memory used by the indexer during the whole process.

Querying the engine: Initialize the engine using the script. The output is this below.

```
chatzakis@chatzakis:/mnt/c/Users/Manos/Documents/GitHub/BioMedicEngine/BioMedicEngine/bash$ ./run.sh
>>Total terms loaded: 3027032
>>> Type a query
|
```

Then, you can start applying queries and their topic:

```
>>> Type a query
58 year old woman with back pain
>>> Type a medical type (diagnosis, test, treatment)
treatment
|
```

BioMedic Engine responds:

```

26684. {42863,C:\MedicalCollection\13\2172078.xml} score: 1.0309951072189073E-4
26685. {30283,C:\MedicalCollection\09\2279262.xml} score: 1.0279416568612216E-4
26686. {31720,C:\MedicalCollection\10\2671137.xml} score: 1.0263197502578247E-4
26687. {10519,C:\MedicalCollection\02\2241853.xml} score: 1.0239498409294707E-4
26688. {44015,C:\MedicalCollection\14\2064391.xml} score: 1.0238604904141323E-4
26689. {756,C:\MedicalCollection\00\2632755.xml} score: 1.0156142015563931E-4
26690. {75842,C:\MedicalCollection\23\2625443.xml} score: 1.0093934572241439E-4
26691. {60910,C:\MedicalCollection\19\1078522.xml} score: 1.0084323702647972E-4
26692. {20846,C:\MedicalCollection\05\2585811.xml} score: 1.0035918020652174E-4
26693. {69883,C:\MedicalCollection\22\2396798.xml} score: 1.0017298144046235E-4
26694. {66876,C:\MedicalCollection\21\2719092.xml} score: 9.91698600329229E-5
26695. {51644,C:\MedicalCollection\16\2515195.xml} score: 9.898749195741921E-5
26696. {20962,C:\MedicalCollection\05\2586335.xml} score: 9.84045522118616E-5
26697. {68882,C:\MedicalCollection\22\2391251.xml} score: 9.779585960642803E-5
26698. {91406,C:\MedicalCollection\27\2696597.xml} score: 9.75751290072206E-5
26699. {19288,C:\MedicalCollection\05\2577362.xml} score: 9.727386535392472E-5
26700. {36645,C:\MedicalCollection\11\2570670.xml} score: 9.703240013257641E-5
26701. {67464,C:\MedicalCollection\21\2721414.xml} score: 9.683029370041125E-5
26702. {19161,C:\MedicalCollection\05\2677007.xml} score: 9.67240920802817E-5
26703. {75906,C:\MedicalCollection\23\2626601.xml} score: 9.558476339920932E-5
26704. {98680,C:\MedicalCollection\29\1820610.xml} score: 9.508196198055498E-5
26705. {31266,C:\MedicalCollection\10\2669286.xml} score: 9.4791007454049795E-5
26706. {18322,C:\MedicalCollection\04\2130882.xml} score: 9.400006716781246E-5
26707. {70127,C:\MedicalCollection\22\2390788.xml} score: 9.3904424305640887E-5
26708. {100013,C:\MedicalCollection\29\1851974.xml} score: 9.384001924329795E-5
26709. {18338,C:\MedicalCollection\04\2139941.xml} score: 9.380612752779598E-5
26710. {20870,C:\MedicalCollection\05\2585847.xml} score: 9.260809021098914E-5
26711. {19967,C:\MedicalCollection\05\2581893.xml} score: 9.236227193440695E-5
26712. {6283,C:\MedicalCollection\01\2654016.xml} score: 9.229637145546446E-5
26713. {89612,C:\MedicalCollection\27\2690687.xml} score: 9.187641538317283E-5
26714. {6732,C:\MedicalCollection\01\2654960.xml} score: 8.958509714172324E-5
26715. {92381,C:\MedicalCollection\27\2699538.xml} score: 8.922460125751009E-5
26716. {42439,C:\MedicalCollection\13\2171266.xml} score: 8.729954952212799E-5
26717. {4834,C:\MedicalCollection\00\2648954.xml} score: 8.724956816240038E-5
26718. {50581,C:\MedicalCollection\16\2490746.xml} score: 8.52248875908364E-5
26719. {70216,C:\MedicalCollection\22\2405950.xml} score: 8.419016981392001E-5
26720. {87269,C:\MedicalCollection\26\2048531.xml} score: 8.051396973872002E-5
26721. {10672,C:\MedicalCollection\02\2242833.xml} score: 7.977193967664569E-5
26722. {13199,C:\MedicalCollection\02\2254332.xml} score: 7.8774833931235118E-5
26723. {36030,C:\MedicalCollection\11\2567903.xml} score: 7.825990555472952E-5
26724. {90535,C:\MedicalCollection\27\2694275.xml} score: 7.811465352807527E-5
26725. {31542,C:\MedicalCollection\10\2670432.xml} score: 7.51875017984743E-5
26726. {98493,C:\MedicalCollection\29\1810395.xml} score: 7.511612323035171E-5
26727. {52185,C:\MedicalCollection\16\2518093.xml} score: 7.497921294187754E-5
26728. {44382,C:\MedicalCollection\14\2064762.xml} score: 7.44479855535981E-5
26729. {105855,C:\MedicalCollection\30\194663.xml} score: 6.171390641672286E-5
Response Time: 12.6361613 seconds
Total documents retrieved: 26729
>>> Type a query

```

A typical BioMedic Indexer answer to a query is a set of documents logged in the following way:

“RANK. {DOCUMENT ID, PATH} score: SCORE”

```

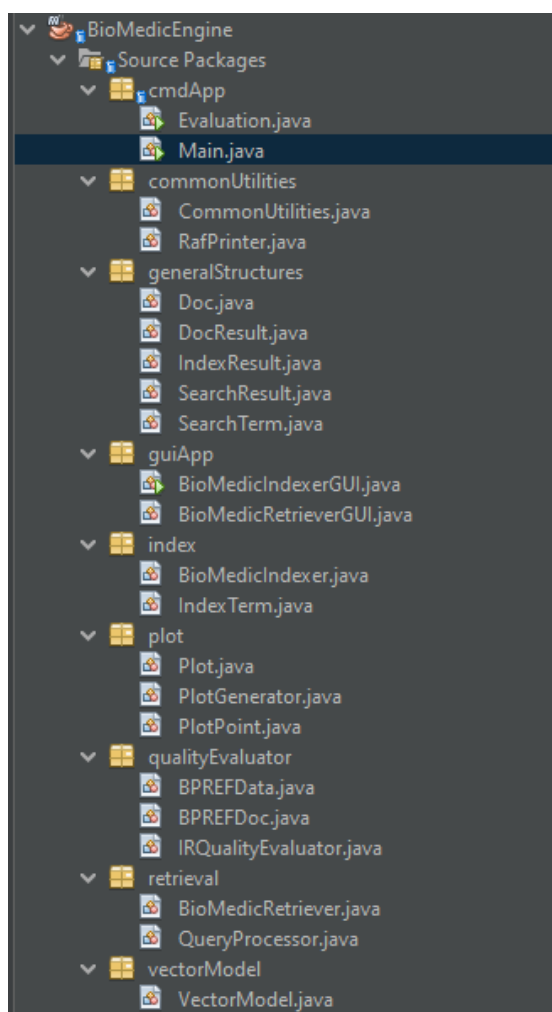
>>> Type a query
manos
>>> Type a medical type (diagnosis, test, treatment)
test
1. {65424,C:\MedicalCollection\20\2930172.xml} score: 0.0640647514177235
2. {12417,C:\MedicalCollection\02\2248289.xml} score: 0.0443578141493315
3. {53413,C:\MedicalCollection\17\2360582.xml} score: 0.03163012181263408
4. {1002,C:\MedicalCollection\00\2633327.xml} score: 0.03037159341804399
5. {7917,C:\MedicalCollection\01\2660293.xml} score: 0.02721603126087708
6. {64918,C:\MedicalCollection\20\2377076.xml} score: 0.026547488914530935
7. {106564,C:\MedicalCollection\30\1963477.xml} score: 0.023909794862741293
8. {34753,C:\MedicalCollection\11\2561144.xml} score: 0.020937570129309312
9. {64912,C:\MedicalCollection\20\2377070.xml} score: 0.019479788276633048
10. {81868,C:\MedicalCollection\25\2229563.xml} score: 0.01890936089050148
11. {52770,C:\MedicalCollection\17\2359936.xml} score: 0.017647126661477609
12. {32785,C:\MedicalCollection\10\2673979.xml} score: 0.016230558271091062
13. {6244,C:\MedicalCollection\01\2653750.xml} score: 0.015877300247975148
14. {52596,C:\MedicalCollection\17\2359638.xml} score: 0.01398410966057858
15. {10340,C:\MedicalCollection\02\2238956.xml} score: 0.013572435765708496
16. {3808,C:\MedicalCollection\00\2644764.xml} score: 0.013291919724824552
17. {100035,C:\MedicalCollection\29\1852093.xml} score: 0.013021618534209477
18. {53461,C:\MedicalCollection\17\2360631.xml} score: 0.0122549107056771
19. {6900,C:\MedicalCollection\01\2655708.xml} score: 0.012087606453622307
20. {53458,C:\MedicalCollection\17\2360627.xml} score: 0.011969336134703374
21. {92036,C:\MedicalCollection\27\2698849.xml} score: 0.011688108205183624
22. {13948,C:\MedicalCollection\03\2199029.xml} score: 0.011376730832689072
23. {71914,C:\MedicalCollection\22\2423659.xml} score: 0.01040554048328846
24. {65001,C:\MedicalCollection\20\2377161.xml} score: 0.010115453071802886
25. {64313,C:\MedicalCollection\20\2375316.xml} score: 0.009770398472389906
26. {54254,C:\MedicalCollection\17\2361774.xml} score: 0.008543933301543941
27. {32088,C:\MedicalCollection\10\2671936.xml} score: 0.007941647141263105
28. {1526,C:\MedicalCollection\00\2635728.xml} score: 0.007615260607781371
29. {3334,C:\MedicalCollection\00\2642781.xml} score: 0.0075730340835797605
30. {52793,C:\MedicalCollection\17\2359959.xml} score: 0.007043444110963338
31. {43463,C:\MedicalCollection\14\2063813.xml} score: 0.0066911651438861346
32. {53590,C:\MedicalCollection\17\2360760.xml} score: 0.006529358951569608
33. {9276,C:\MedicalCollection\01\2665263.xml} score: 0.005932117588551033
34. {54304,C:\MedicalCollection\17\2361825.xml} score: 0.005916252566793861
35. {79137,C:\MedicalCollection\24\3516685.xml} score: 0.005678830661881261
36. {81818,C:\MedicalCollection\25\2229513.xml} score: 0.00520912444973797675
37. {70999,C:\MedicalCollection\22\2400800.xml} score: 0.004712691981557766
38. {44462,C:\MedicalCollection\14\2064084.xml} score: 0.0038446673323846133
39. {55579,C:\MedicalCollection\17\2735164.xml} score: 0.0034324803876640892
Response Time: 4.3480928 seconds
Total documents retrieved: 39
>>> Type a query

```

Also, the total documents retrieved and respond time is reported. When using the command line interface, the program can be terminated by typing “!exit” as query input.

```
>>> Type a query
!exit
chatzakis@chatzakis:/mnt/c/Users/Manos/Documents/GitHub/BioMedicEngine/BioMedicEngine/bash$ |
```

Project Layout:



About the architecture:

Package index: This package contains all classes needed to Index a directory.

Package retrieval: This package contains all classes needed to perform query answering (simple or topic).

Package cmdApp: This package contains the CLI application of the BioMedicIndexer. It also contains the automation methods for quality evaluation, in the Evaluation class file.

Package generalStructures: This package contains the general structures needed by (most of) all other packages, such as “Document” etc.

Package guiApp: This package contains the GUI applications (not finalized).

Package plot: This package contains methods to create the plots (used in the report of the index results).

Package vectorModel: This package contains all methods needed to calculate norms, TF*IDF arrays etc.

Package qualityEvaluator: This package contains all the methods needed to automate the quality evaluation process and calculate the per-topic-BPREF.

Package commonUtilities: All utility functions.

2 Implementation

In this section, the basic methods used to implement both parts of BioMedic Engine (Index Creation and Query Answering) are described.

2.1 Index Creation

BioMedic Indexer indexes a selected directory in three steps: (a) Partial Indexing (b) Partial Merging and (c) Document Norm Calculation.

Partial Indexing.

BioMedic Indexer uses a sorted <String, Term> map to store the terms. During the first phase of indexing, the terms and related information are stored into this map. This map stores their df their occurrences per tag etc. The document collection is read sequentially and the contents of each document are added to the map.

When the size of this map gets greater than a threshold TH, the contents of the map are flushed to the disk, creating a pair of partial files, with names “vocabX” and “postX”. These files are stored in a list. Every time these files are flushed to the disc, the map contents are cleared to maintain memory into a specific level. BioMedic Indexer uses a relatively small TH, to be able to run in systems with small memory capacity.

Merging.

After the partial indexing phase is completed, the files need to be merged. To merge the files, the tool removes two files from the list, and adds the output of merging to the list, till the list size is 1.

This list contains the names of the partial vocabularies files, and the corresponding posting file is found by replacing the part “vocab” with “post”. **This is why the directory path should not contain words such as “vocab” and “post” to avoid such mistakes when the program locates the corresponding posting file of the current partial vocabulary.** Then the remaining file is the vocabulary file, and the corresponding posting is the posting file. All of this files are maintained, traversed etc. using the Java *RandomAccessFile* API.

Merging Algorithm (Like merging two linked lists ☺): We create the new posting and vocab file keeping the the lexicographical order the same. For every term added to the file, we should also merge their postings, in an ordered way using the document IDs, as seen in the algorithms below.

```
int n_counter = 0;
while (vocabFileNames.size() > 1) {
    String vocab1 = partialFilesDirectory + vocabFileNames.remove(index: 0);
    String vocab2 = partialFilesDirectory + vocabFileNames.remove(index: 0);

    mergePartialFiles(vocabFileNames, vocab1, vocab2, dir: partialFilesDirectory, n_counter++);
}
```

```
String lineV1 = voc1.readUTF(), lineV2 = voc2.readUTF();
while (!lineV1.equals( anObject: "#end") && !lineV2.equals( anObject: "#end")){

    String[] contentsV1 = lineV1.split( regex: " ");
    String[] contentsV2 = lineV2.split( regex: " ");

    String currentTermV1 = contentsV1[0];
    String currentTermV2 = contentsV2[0];

    int comp = currentTermV1.compareTo( anotherString: currentTermV2);
    if (comp == 0) {
        mergeContentsAndCopyToRAF(contentsV1, contentsV2, pos1: post1, pos2: post2, vocNew: newVoc, postNew: newPost);
        lineV1 = voc1.readUTF();
        lineV2 = voc2.readUTF();
    } else if (comp < 0) {
        copyContentsToRAF( vcontents: contentsV1, pos1: post1, vocNew: newVoc, postNew: newPost);
        lineV1 = voc1.readUTF();
    } else {
        copyContentsToRAF( vcontents: contentsV2, pos1: post2, vocNew: newVoc, postNew: newPost);
        lineV2 = voc2.readUTF();
    }
}
```

```
while (!lineV1.equals( anObject: "#end")) {
    String[] contentsV1 = lineV1.split( regex: " ");
    copyContentsToRAF( vcontents: contentsV1, pos1: post1, vocNew: newVoc, postNew: newPost);
    lineV1 = voc1.readUTF();
}

while (!lineV2.equals( anObject: "#end")) {
    String[] contentsV2 = lineV2.split( regex: " ");
    copyContentsToRAF( vcontents: contentsV2, pos1: post2, vocNew: newVoc, postNew: newPost);
    lineV2 = voc2.readUTF();
}
```

The algorithms above ensure that the order of both new vocabulary and posting files are right and no information is duplicated or lost during the merging process.

Document Norm Calculation.

The norms are calculated in a different file and stored separately. After the completion of the partitioning and merging, we initialize the vocabulary and we keep a map <Integer,Double> which stores the mappings of the document ID with it's norm. We traverse the terms one time and if a document contains the term we add $(TF*IDF)^2$ to the total current norm.

After the traversal, we write the \sqrt{map} in a new random access file called “norms” and we save the mappings of document IDs and the map pointers.

2.2 Query Answering

Here, the process of query answering is described.

Vector model.

[Step 1 – Initializing BioMedic Retriever]: Given a directory to index, the vocabulary is initialized and kept in memory, while we also load the pointers to the Random Access Files. Only the vocabulary and the pointer mappings are stored in-memory.

[Step 2 – Getting the relevant documents]: Given a query, the query processor parses the query using “[spaces...]” and finds its terms. Then, we traverse the terms one by one, and for every term present in the vocabulary, we traverse its postings and retrieve the documents in a list. This list contains the relevant documents. For performance improvements, we also keep their corresponding TFs per document.

[Step 3 – Finding the norm of the vector]: The query processor not only parses the query to its terms, but it returns a map of <Term, TF>. Thus, using the TF of the term inside the

query and the iDF as it comes from the model, we can calculate the norm of the query the same way we did for the documents. Indeed, terms that are not present in the vocabulary are removed.

[Step 4 – Find the dot product per vector and query]: For every relevant document, we do the following. For every term in the query, we have the queryTF and iDF, while we have also saved the TF of the term in the document so, we calculate the dot product as the sum of $(\text{queryTF} * \text{iDF}) * (\text{docTF} * \text{iDF})$ of every term that has a $\text{TF} > 0$. Previously, the postings were traversed again, but this lead to performance degregation for queries with vast results.

[Step 5 – Find the score of document]: Given that we have the dot product and the norms available, the score is $(\text{dot product}) / (\text{docNorm} * \text{queryNorm})$.

[Step 6 – Return the results]: The documents are stored in a sorted list based on their score. This list is returned, with the time needed to answer the query.

Vector model with Examination type support.

[Step 7 – Support Examination Type]: The goal of this part is to not only return the documents related to the query, but also try to return documents that also correspond to an examination type. The methods and the evaluation of theses ideas are presented in section 4.

Query Processor.

The query processor is a class that parses a query, removes the stop points (eg. “.”, “,” “()” ...) and returns a map that consists of the query terms and their TF. Also, it can load files of stopwords just like the index creation, using the same files and method calls, in order to remove the stopwords in greek and english.

2.3 Checking and Debugging

BioMedic Indexer contains classes that can print the vocabulary and the corresponding random access files. We also checked the program by applying a number of random queries, while the program contains assertions that ensure that the index creation and load is fine during query answering.

2.4 Technical Details

All algorithms, methods etc. are implemented in Java, openjdk version "11.0.11" 2021-04-20 and OpenJDK Runtime Environment (build 11.0.11+9-Ubuntu-0ubuntu2.20.04), but it also supports Java 8.

Also, maven is needed, version Apache Maven 3.6.3.

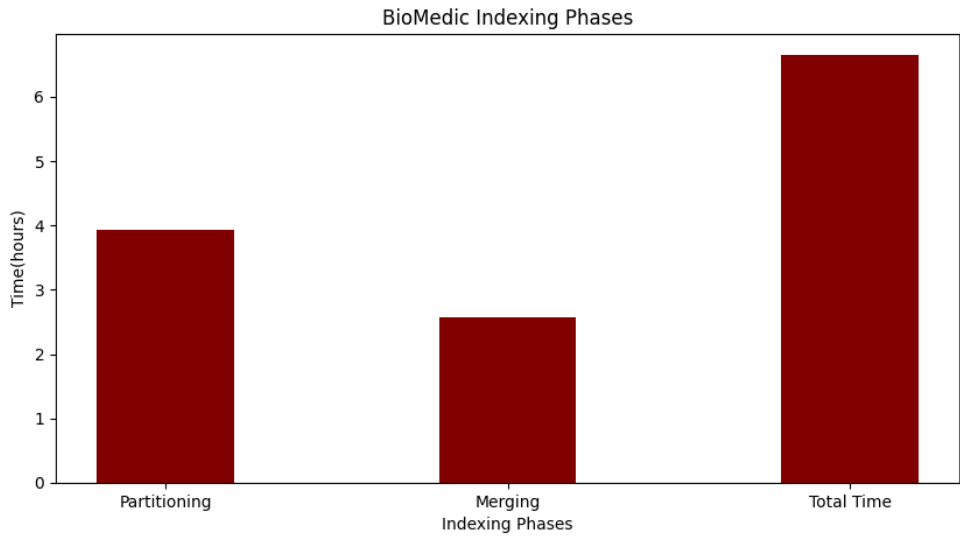
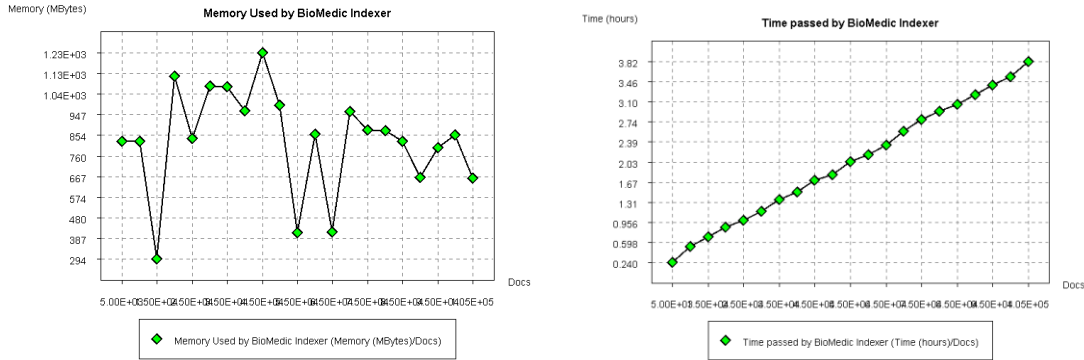
BioMedic Engine uses a number of frameworks to work, so in case it is loaded to an ide, a complete build should be made. Examples of these frameworks are “Lombok”, “biomedicReader”, “mitosStemmer”, “Apache Commons CLI” and a maven plugin to produce an exe-jar executable to be able to use the command line interface.

3 Experimental Evaluation

The experiments conducted on a machine of *16GB Memory, 1TB SSD NVMe Disc, and 8-Core (16 Hyperthreads) CPU, running Windows 11.*

3.1 Index Creation Evaluation

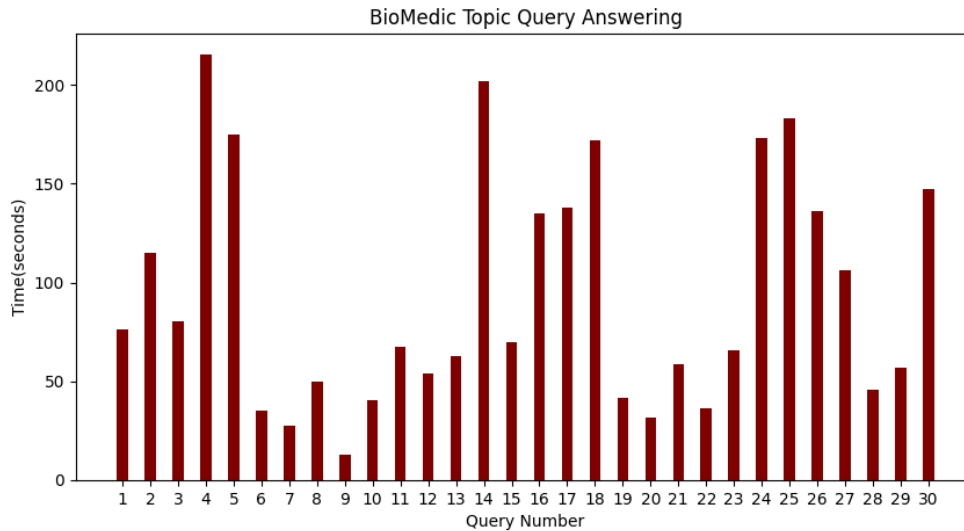
The evaluation of the indexing phase is presented in two graphs: A graph showing the memory usage correlated with the document count, and a graph showing the total time passed correlated with the document count for the partitioning phase of the algorithm.



Note that the total time needed to index a directory is also based on the threshold we choose. Choosing a smaller threshold to maintain the total memory used in small levels (see corresponding figures) could be beneficial as the program can run in any machine. With the current configuration, the total time for indexing is approximately 6 hours.

3.2 Query Answering Performance Evaluation

For query answering, we show the total response time needed for the queries created from the files of “topics.txt”, using the summaries of these queries. The results can be seen in the graph below.



This evaluation is performed automatically from the system, using statistics gathering and analysis, and can be reproduced easily.

4 Quality Evaluation

BioMedic Indexer comes with a Quality Evaluator package, that calculates the results of a set of queries, as given from the BioMedic Indexer, and then calculates some evaluation metrics, e.g. BPREF.

This process is automated and it consists of two phases: (a) creation of the result file and (b) the evaluation of the results.

4.1 Creation of the result file

As a first step, a result file that holds the top-1000 answers from specific medical topics, sorted by their score, one per line:

```
IRQualityEvaluator iqe = new IRQualityEvaluator();

String basePath = "C://BioMedicIndexer_2/";
String resultsPath = ".corpus/results-topic.txt";
String versionRunName = "topic-biomedic-engine";
iqe.createResultFileOfBioMedicIndexer(basePath, resultsPath, versionRunName);
```

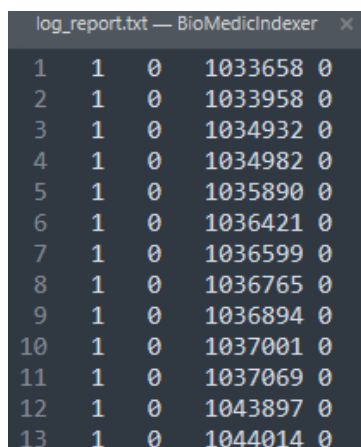
The output of this process is a file of this form:

```
1 1 0 2562018 1 0.1363858523436266 topic-biomedic-engine
2 1 0 2518278 2 0.12344087714193332 topic-biomedic-engine
3 1 0 2615777 3 0.11427938062542377 topic-biomedic-engine
4 1 0 2396976 4 0.10903621285752298 topic-biomedic-engine
5 1 0 2621386 5 0.09945024593852532 topic-biomedic-engine
6 1 0 1820749 6 0.09518714240688884 topic-biomedic-engine
7 1 0 2685385 7 0.09261849295810085 topic-biomedic-engine
8 1 0 2694417 8 0.0887261553303287 topic-biomedic-engine
9 1 0 1940093 9 0.08794436368517417 topic-biomedic-engine
10 1 0 1906857 10 0.08779413721361261 topic-biomedic-engine
11 1 0 2588883 11 0.08572795855992202 topic-biomedic-engine
12 1 0 3389947 12 0.08546462546185728 topic-biomedic-engine
13 1 0 2835690 13 0.08498666125228116 topic-biomedic-engine
14 1 0 2676268 14 0.08442509103795136 topic-biomedic-engine
15 1 0 1876464 15 0.08019539028897225 topic-biomedic-engine
```

This file is later used to evaluate these results.

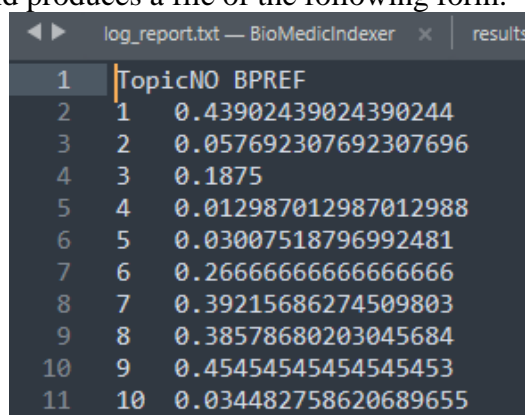
4.2 Evaluation of the results using external knowledge

The results are evaluated using an external file (“qrels.txt”) which is a TSV file of the form:



1	1	0	1033658	0
2	1	0	1033958	0
3	1	0	1034932	0
4	1	0	1034982	0
5	1	0	1035890	0
6	1	0	1036421	0
7	1	0	1036599	0
8	1	0	1036765	0
9	1	0	1036894	0
10	1	0	1037001	0
11	1	0	1037069	0
12	1	0	1043897	0
13	1	0	1044014	0

Holding information about which files are relevant to the medical topic and which are not. The data of this file are loaded in-memory, then the data saved in the result file of the previous phase are also loaded and BioMedic Indexer calculates the per-topic-BPREF based on these results and produces a file of the following form:



1	TopicNO	BPREF
2	1	0.43902439024390244
3	2	0.057692307692307696
4	3	0.1875
5	4	0.012987012987012988
6	5	0.03007518796992481
7	6	0.26666666666666666
8	7	0.39215686274509803
9	8	0.38578680203045684
10	9	0.45454545454545453
11	10	0.034482758620689655

Again, the process of the BPREF-calculation is automated and can be reproduced for different engines, algorithms etc. An outline of this process in the code is given in the following image:



```
public static void main(String[] args) throws Exception {
    IRQualityEvaluator iqe = new IRQualityEvaluator();

    String basePath = "C:/BioMedicIndexer_2/";
    String givenResultsFilepath = "./corpus/qrels.txt";

    double[] topicWeights = {0.0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0};
    boolean[] intersections = {false, true};

    for (double w : topicWeights) {
        for (boolean i : intersections) {
            String resultsPath = "./qualityEvaluation/results-topic-weighting-" + w + "-iset_" + i + ".txt";
            String versionRunName = "topic-biomedic-engine-weighting-" + w + "-iset_" + i;

            iqe.createResultFileOfBioMedicIndexer(basePath, resultsPath, versionRunName, topicWeight: w, inter: i);

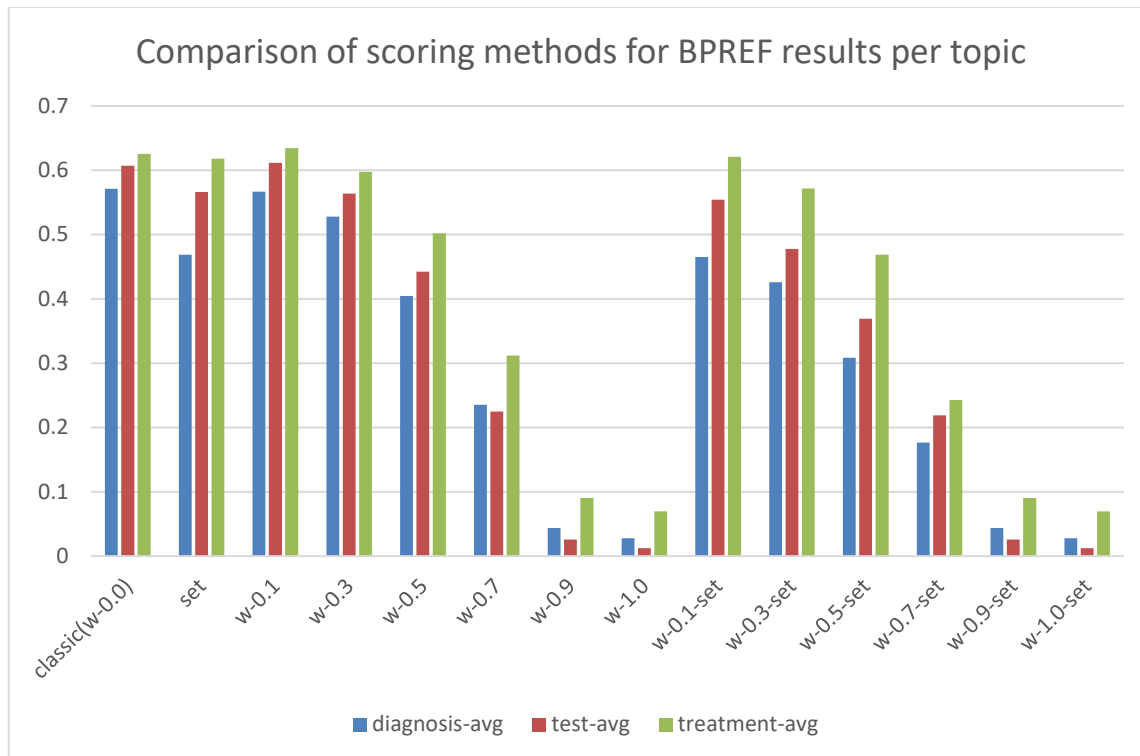
            String reportFilepath = "./qualityEvaluation/evaluation-results-topic-weighting-" + w + "-iset_" + i + ".txt";
            iqe.calculateBPREFinMemory(systemResultsFilepath: resultsPath, givenResultsFilepath, reportFilepath);
        }
    }
}
```

Results:

Now, we reason about different scoring/retrieval methods to achieve better scores regarding that the type of the documents we want to retrieve is given by the user. Below we present a description of the methods and ideas that were implemented:

Method	Description	Reasoning
Classic Query Answering	Retrieving the documents using the classic vector model, without any modification.	Classic Retrieval, Baseline idea
Set Operations (Interestion)	Retrieving the relevant documents to a given query (set A), retrieve the relevant documents to a given type (set B), and retain all the items from set A that do not belong to set B. Then apply the vector model scoring.	We think that filtering out the documents that are not even relevant to the topic (when topic is given as a query) can lead to better results
Weighting (X-Y)	Changing the scoring function to assign weights. Every document has two scores: ScoreA is the score that has based on the given query, and ScoreB is the score that has based on the given medical topic. Then use a new weighted function (e.g.) $\text{finalScore} = X\% \text{ScoreA} + Y\% \text{ScoreB}$. Indeed, $X + Y = 1$, because the score should always be ≤ 1 .	We are trying to concurrently retrieve the documents that are relevant to the query but give better scores to the ones that are relevant to the topic also.
Set Operations & Weighting(X-Y)	Combination of the two aforementioned methods: Keep a set of files that result from the intersection of the files that are relevant to the query and the topic, and then apply the weighting method for the scoring.	Combination of the aforementioned ideas

The results of each method are presented in the following screenshot from the file "eval.xlsx", as we compared all 4 scoring methods for different configurations for the weightng for the topic (values of Y): [0.1, 0.3, 0.5, 0.7, 0.9, 1.0], with and without intersections etc., and we got the results shown in the graph below:



We see that the weighting method for $X=0.9$ and $Y=0.1$ gives the best results, so we chose this for the BioMedic Engine. The excel result file contains all the detailed results per query and the graph.

	classic(w-0.0)	set	w-0.1	w-0.3	w-0.5	w-0.7	w-0.9	w-1.0	w-0.1-set	w-0.3-set	w-0.5-set	w-0.7-set	w-0.9-set	w-1.0-set
diagnosis-avg	0.57103372	0.468603769	0.568522325	0.520785208	0.404335996	0.225933801	0.04397675	0.027970098	0.455295212	0.425837317	0.308427107	0.178532045	0.04357675	0.027970098
test-avg	0.6070785354	0.566393942	0.61850404	0.563541803	0.442468212	0.22474531	0.025823445	0.012580973	0.554435469	0.477570172	0.369368577	0.218856229	0.025823445	0.012580973
treatment-avg	0.625326555	0.617834161	0.634297533	0.59747597	0.501979144	0.312005095	0.090626019	0.069448782	0.620961742	0.571552248	0.468856876	0.242384044	0.090626019	0.069448782

Also this file contains results for every one of the 30 queries and other metrics from this evaluation, for example min-max values retrieved, for every configuration that we tried.

We see that generally, the intersection operation leads to worse results. This behavior is observed because in the resultset, there can be documents that do not have strong relation to the given topic, but could be really close to the basic query. Filtering out those documents leads to worse results.

We also notice that giving more weight to the topic leads to worse results. This happens because the score a document coming from the theme of the query is more important than the topic instead, and a scoring system that considers the topic-score of the document more important leads to such outcome.

5 Conclusion

This report presents the outline of the work for the creation of a BioMedical Search Engine. It contains basic information about how to use the engine and the architecture, while the algorithms used for index creation and query answering are presented and explained. Moreover, a modification of the vector-model is presented to be able to retrieve documents that are more relevant to a specific topic (treatment, diagnosis, test) apart from the classic vector model. Also, an experimental evaluation for both indexing and query answering is shown, while the project contains a quality evaluator module that is used to evaluate the results of the engine. Finally, a quality evaluation using the module is presented, comparing different implementations for document-scoring and their corresponding accuracy. We also provide a brief explanation of the results.