Assignment 2: Querying

Nicolai Dahl Blicher-Petersen and Daniel Jonathan Smith
{s3441163, s3361789}@student.rmit.edu.au

RMIT

1 Ranked Retrieval

Building on the inverted index created in assignment one, we have implemented a simplified BM25 similarity function to efficiently rank documents with respect to an input query. The SimpleQueryEngine implemented in assignment one has been extended in a way that allows us to reuse loading of the inverted index and the document map. Furthermore, looking up the list of postings for a term is available through the getSearchResult method [2]. The class that carries out the ranked retrieval is called BM25RankedQueryEngine.

Indexing

The document weight component of the BM25 function, $K = k_1 \cdot \left((1-b) + \frac{b \cdot L_d}{AL}\right)$, is calculated at indexing time to improve query speed.

As each document is indexed its length in bytes is stored. Immediately before writing the full Document ID map to disk, the average of these document lengths is calculated. K is then calculated for each document and written to the document map on the same line as the document's raw Document ID, to be read in with the Document ID by the query program.

Processing a Query

The lexicon and inverted list in the query program work in an identical manner to the SimpleQueryEngine implemented in assignment one. To minimise code duplication this code is implemented in a QueryEngine class, that is subclassed by the various classes implementing each querying method. The only difference is that the document weight calculated at the indexing stage is read in to the DocIdHandler. Similarly to the raw Document ID, the document weight is stored in an array, where it resides in the array index of its corresponding internal Document ID to allow constant time lookup.

We also reuse the parser and stopper modules designed in the first assignment when first processing an input query. This way we ensure that the query terms are treated the same way as when building the index [2].

The query engine first steps through each query term as returned by the parser. It first retrieves a list of documents that the term appears in from the inverted index. For each of these documents the simplified BM25 similarity score

$$BM25(t, D_d) = \log\left(\frac{N - f_t + 0.5}{f_t + 0.5}\right) \cdot \frac{(k_1+1) f_{d,t}}{K + f_{d,t}}$$

is calculated for the current query term, where K is the document weight calculated at the indexing stage and retrieved from the DocIdHandler as described above. If an accumulator already exists for this document the similarity score is added to the accumulator value, otherwise a new accumulator is created with the calculated similarity score as its value.

Once all terms in the query have been processed in this way the top n accumulators are found by stepping over each and attempting to add it to a MinHeap of size n, which is explained in the section below. Lastly, an array sorted by the accumulator value is then created from the MinHeap, and the Document IDs and accumulated similarity scores are returned as the query result.

Data Structures

For the ranking algorithm two essential data structures are used. First of all we use a java HashMap to store our accumulators with the document ID as key and the accumulated ranking score as value. This allows for constant-time read and put operations when updating the accumulator values, which is essential for queries containing terms that appear in large amounts of documents [4].

The second data structure we take advantage of is the Min-heap [10]. It is used in the final stage of the ranking procedure to retrieve the n accumulators with the highest similarity scores.

The Min-heap is specified to only allow n elements in it, and every time a new accumulator is added it checks if the limit has been reached. If the heap is not yet full, the accumulator is added to the heap and the heap is heapified. If the heap is full, the accumulator's value is checked against that of the heap's root element and the accumulator is only added to the heap if it is of greater value than the root element. This allows us to check if an accumulator needs to be added in constant time, avoiding the extra cost that would be involved if all elements had to be checked.

Heapifying the heap is a linear-time operation, as the heapify routine, performed on insertion, is bound by the height of the heap, which is constant. In total, the asymptotic complexity of finding the n highest-valued elements is $O(n)$.

2 Advanced IR Feature

As our advanced information retrieval feature we selected automatic query expansion, also known as pseudo-relevance feedback. Queries might be hard to specify for users and there is a danger that the user searches for a synonym to a word that is more significant in the document collection [1]. The idea is to expand the initial query with E additional terms that are statistically related to the query to mitigate this vocabulary mismatch [1]. The steps to automatic query expansion are [9]:

1. Perform ranked retrieval on the initial query with a good similarity measure and assume that the top R ranked documents are relevant.
2. Parse through these R documents and mark all terms in these candidate terms for query expansion.
3. Select the best E of these candidate terms for the query expansion by evaluating them with some statistical method.
4. Append the E terms to the initial query and run the ranked retrieval procedure again. This is the final result.

As the statistical method in step three, we use the Okapi Term Selection Value (TSV) approach to select a set of E terms to extend the initial query with [1][7].

Implementation

We have extended the BM25RankedQueryEngine presented in Section 1 to handle the query expansion. The class is called QueryExpansionBM25QueryEngine and is automatically used as query engine if the $-QEBM25$ input flag is specified. The getResults method of the class follows the approach described above with the most difficult step being step 2, as the current invertedIndex does not allow us to retrieve all terms for a particular document. For step 1 of the algorithm we simply call the getResults method of the BM25RankedQueryEngine.

To get part 2 working we have had to do some extra work at indexing time. We save an uninverted index of documents, where document IDs in the term lexicon (termLex) point to a term list in the termIndex. We also save a term map that maps a term ID to a specific term, called termMap. Keeping track of this extra termMap allows us to reuse our indexing and compression code from assignment one, as we are once again only saving numerical data in the termIndex file [2].

So when the list of candidate terms is requested, each document's term list is found by first looking up the byte offset (into the termIndex) of the list of terms, which is listed in the termLex (lexicon). The findCandidateTerms method performs this procedure for each of the R documents and uses a HashMap [4] from term IDs to frequencies, to make sure that terms occurring in multiple documents are only saved as candidate terms once. The frequencies mentioned are not the within-document frequencies but the "number of documents in the initially retrieved pool that contain the term" [9], as this measure is later to be used in the TSV calculation.

When the candidate terms have been retrieved, the next job is to compute the TSV scores and provide the E lowest results. We again apply a Minheap in the form of a Java PriorityQueue [5] to efficiently sort and retrieve the lowest-valued candidate terms.

Lastly, the initial query is expanded with the E lowest-valued candidate terms, the getResults method of BM25RankedQueryEngine is run again with this new query, and the results of this process are returned as the final results of the ranked retrieval procedure.

3 Evaluation

3.1 Precision at 10 (P@10) Evaluation

The *Precision at 10* metric (P@10) describes the precision of a query after 10 answers have been seen. The P@10 score is equal to the number of relevant results divided by the number of answers, or $\frac{R}{10}$. This metric is often used since it reflects the relevance of the first page of search results returned by a web search engine, given the general default display of 10 answers per page. Since, as studies have shown, most users do not look past the first page of search results, this is a good general reflection of the relevance of the query result [8][3, p. 161].

The provided queries were run using the BM25 and BM25 with query expansion (hereafter referred to as *BM25QE*) querying methods.

The BM25QE function can be run with varying values for R (the number of top-ranked documents that are assumed to be relevant)and E (the number of terms that should be appended to the original query). We first ran the given queries with several different values of R and E to choose optimal values for comparison.

The average relevance score using the P@10 metric over the five sample queries with each combination of R and E values is shown in Table 1.

	$E = 5$	$E = 10$	$E = 15$	$E = 20$	$E = 25$	$E = 30$
$R = 5$	0.22	0.26	0.20	0.22	0.18	0.22
$R = 10$	0.28	0.32	0.26	0.28	0.30	0.30
$R = 15$	0.26	0.24	0.24	0.22	0.22	0.24
$R = 20$	0.24	0.26	0.20	0.20	0.20	0.20
$R = 25$	0.26	0.24	0.18	0.18	0.16	0.16
$R = 30$	0.20	0.22	0.20	0.16	0.16	0.16

Table 1. Average P@10 score for BM25QE with different combinations of R and E

We can see that the P@10 relevance score is maximised when $R = 10$, but there is not as definite a frontrunner when setting a value for E. As a result we chose to evaluate the BM25QE method with $R = 10$ for the three optimal values of E, $E = 10$, $E = 25$ and $E = 30$, in the comparison between BM25 and BM25QE.

Query	BM25	BM25QE $E = 10$	BM25QE $E = 25$	BM25QE $E = 30$
401	0.1	0.0	0.0	0.0
402	0.2	0.1	0.2	0.2
403	0.6	0.7	0.6	0.6
405	0.2	0.5	0.4	0.4
408	0.4	0.3	0.3	0.3
Avg	0.30	0.32	0.30	0.30

Table 2. P@10 relevance score for each query method

Table 2 lists the P@10 score for each query by querying method. These results don't seem to show any significant difference in effectiveness between the different query methods. There are hints of a slightly more even distribution of P@10 scores between the different queries when using larger values of E with the BM25QE method, but a larger sample size would be needed to confirm this.

3.2 Mean Average Precision (MAP) Evaluation

The *Mean Average Precision* metric (MAP) may provide a more useful comparison of our query methods. Since the MAP score takes account of both the ranking position of relevant documents (precision) and number of relevant documents retrieved (recall), where the P@10 metric measures only the recall of the query at 10 results, it may provide a more nuanced overview of the query results [3, pp. 159-162]. For example, it may prove to be the case that although recall is similar between the different query methods some return relevant documents at higher ranks.

The MAP is calculated by first taking the average of the precision obtained after each document is retrieved for each query then taking the mean of these average precisions over all sample queries [8, pp. 13-14].

We calculated the MAP over the 20 highest ranked query results for the BM25 ranking method as well as the BM25QE method with the same combinations of R and E evaluated with the P@10 metric. The results are recorded in Table 3.

Query	BM25	BM25QE $E = 10$	BM25QE $E = 25$	BM25QE $E = 30$
401	1.754	0.000	0.000	0.000
402	4.147	0.941	1.267	2.419
403	38.950	63.874	57.024	54.207
405	2.315	15.062	14.563	10.907
408	5.746	4.265	3.205	4.808
MAP	10.583	16.828	15.212	14.468

Table 3. Average Precision and Mean Average Precision % for each query method

We can see that the MAP is significantly improved when using query expansion, which suggests a more precise querying method despite the lack of improvement shown in recall levels with the P@10 metric. On closer inspection, however, this is not as straightforward as it seems.

Although there is an improvement in the MAP score for each variant of the BM25QE method, the average precision for individual queries did not consistently improve and indeed in some cases dramatically declined. As noted with the P@10 metric, a larger sample size would be needed to draw any meaningful conclusions from the data.

Interestingly, consistent with the P@10 scores recorded earlier there seems to be some correlation between higher values of E and a more even distribution of average precision across the different queries.

A Appendix

A.1 Query Results

BM25	**BM25 with Query Expansion** $R = 10, E = 25$
401 LA101790-0075 1 13.372	401 LA021490-0049 1 55.848
401 LA021890-0100 2 12.814	401 LA050790-0042 2 49.818
401 LA100890-0131 3 12.278	401 LA020790-0156 3 48.857
401 LA050690-0109 4 12.244	401 LA061290-0074 4 47.232
401 LA040789-0015 5 12.017	401 LA083090-0247 5 47.085
401 LA021490-0049 6 11.921	401 LA031590-0102 6 46.062
401 LA031590-0102 7 11.784	401 LA050490-0055 7 45.205
401 LA111089-0188 8 11.724	401 LA021890-0100 8 45.202
401 LA071890-0073 9 11.698	401 LA120689-0034 9 44.923
401 LA020789-0133 10 11.580	401 LA112989-0054 10 44.872
401 LA050790-0042 11 11.532	401 LA050690-0109 11 44.430
401 LA060890-0011 12 11.513	401 LA040590-0157 12 44.096
401 LA082789-0152 13 11.419	401 LA091490-0080 13 43.455
401 LA021190-0168 14 11.383	401 LA100289-0097 14 42.147
401 LA040590-0157 15 11.245	401 LA031490-0073 15 41.553
401 LA021389-0098 16 11.223	401 LA071890-0073 16 41.182
401 LA030990-0189 17 11.218	401 LA021190-0168 17 41.022
401 LA050990-0043 18 11.215	401 LA030190-0112 18 39.656
401 LA062290-0172 19 11.214	401 LA020890-0148 19 39.258
401 LA050390-0176 20 11.209	401 LA081290-0159 20 38.434
Running time: 82 ms	Running time: 138 ms
402 LA101290-0115 1 20.222	402 LA101290-0115 1 46.490
402 LA052290-0110 2 14.065	402 LA020389-0077 2 45.898
402 LA020389-0077 3 13.371	402 LA080190-0099 3 42.962
402 LA121289-0055 4 13.183	402 LA052290-0110 4 42.430
402 LA082590-0108 5 12.642	402 LA042990-0032 5 39.944
402 LA080190-0099 6 12.091	402 LA121289-0055 6 39.569
402 LA042990-0032 7 11.330	402 LA042390-0048 7 39.202
402 LA020789-0112 8 11.097	402 LA110889-0005 8 33.748
402 LA071689-0143 9 11.026	402 LA072490-0082 9 33.422
402 LA110889-0005 10 10.771	402 LA082590-0108 10 31.852
402 LA071489-0085 11 10.685	402 LA021290-0061 11 31.419
402 LA042390-0048 12 10.571	402 LA060289-0090 12 31.093
402 LA021290-0061 13 10.498	402 LA020789-0113 13 30.307
402 LA030289-0084 14 10.467	402 LA020789-0112 14 29.973
402 LA012589-0063 15 10.464	402 LA012589-0063 15 29.805
402 LA051689-0102 16 10.396	402 LA071689-0143 16 28.876
402 LA040790-0127 17 10.384	402 LA011089-0045 17 28.795
402 LA060289-0090 18 10.301	402 LA073090-0057 18 28.778
402 LA020789-0113 19 10.179	402 LA110490-0092 19 27.652
402 LA051389-0010 20 10.141	402 LA120389-0120 20 26.729
Running time: 33 ms	Running time: 83 ms

BM25	**BM25 with Query Expansion** $R = 10, E = 25$
403 LA030689-0082 1 15.761	403 LA010790-0103 1 83.603
403 LA071290-0133 2 15.621	403 LA020490-0136 2 71.175
403 LA083089-0024 3 14.952	403 LA071290-0133 3 61.075
403 LA020490-0136 4 14.701	403 LA010390-0067 4 60.311
403 LA011389-0029 5 14.481	403 LA083089-0024 5 59.617
403 LA010790-0103 6 14.162	403 LA111589-0004 6 51.288
403 LA051490-0120 7 12.999	403 LA080289-0067 7 49.383
403 LA032290-0151 8 11.931	403 LA011389-0029 8 47.395
403 LA120689-0083 9 11.727	403 LA082890-0074 9 47.107
403 LA010390-0067 10 11.608	403 LA042589-0052 10 46.729
403 LA111589-0004 11 11.175	403 LA120689-0083 11 46.412
403 LA042589-0052 12 11.152	403 LA042689-0065 12 44.090
403 LA041990-0072 13 10.901	403 LA092890-0067 13 42.848
403 LA042189-0027 14 10.267	403 LA030689-0082 14 40.337
403 LA051889-0006 15 10.174	403 LA051490-0120 15 40.203
403 LA082390-0094 16 9.817	403 LA033089-0013 16 37.511
403 LA022790-0099 17 9.571	403 LA082390-0094 17 35.959
403 LA052290-0110 18 9.564	403 LA032290-0151 18 33.468
403 LA012990-0041 19 9.270	403 LA051889-0006 19 31.465
403 LA080190-0135 20 9.231	403 LA041990-0072 20 29.267
Running time: 33 ms	Running time: 56 ms
405 LA012289-0002 1 14.588	405 LA042089-0083 1 59.288
405 LA010889-0109 2 12.173	405 LA010889-0109 2 57.888
405 LA063089-0071 3 11.949	405 LA063089-0071 3 48.695
405 LA031290-0034 4 11.940	405 LA020190-0053 4 29.920
405 LA021690-0057 5 11.352	405 LA041689-0021 5 29.533
405 LA042089-0083 6 11.175	405 LA092489-0134 6 27.263
405 LA121690-0039 7 10.481	405 LA012289-0002 7 26.748
405 LA020190-0053 8 10.444	405 LA120189-0127 8 26.011
405 LA061490-0089 9 10.427	405 LA011590-0098 9 24.914
405 LA122989-0137 10 10.208	405 LA031290-0034 10 23.881
405 LA121589-0173 11 10.130	405 LA082489-0132 11 23.671
405 LA031389-0056 12 10.092	405 LA082290-0043 12 23.307
405 LA111789-0134 13 10.055	405 LA101989-0137 13 22.890
405 LA081190-0002 14 9.840	405 LA090889-0077 14 22.794
405 LA090190-0129 15 9.790	405 LA021690-0057 15 22.703
405 LA100690-0017 16 9.632	405 LA102189-0071 16 22.471
405 LA091789-0029 17 9.379	405 LA111190-0024 17 22.065
405 LA050990-0163 18 9.372	405 LA060290-0131 18 21.596
405 LA052090-0005 19 9.265	405 LA121690-0039 19 20.962
405 LA010790-0016 20 9.198	405 LA061490-0089 20 20.854
Running time: 64 ms	Running time: 116 ms

BM25

408 LA101490-0142 1 20.220
408 LA062290-0070 2 18.324
408 LA101390-0102 3 17.619
408 LA101289-0148 4 17.057
408 LA103190-0052 5 16.222
408 LA091989-0049 6 16.147
408 LA120389-0130 7 15.910
408 LA021690-0167 8 15.579
408 LA030990-0199 9 14.464
408 LA092089-0027 10 12.280
408 LA110390-0071 11 11.762
408 LA082189-0033 12 10.969
408 LA030989-0189 13 10.808
408 LA102189-0071 14 10.800
408 LA051190-0106 15 10.685
408 LA040289-0192 16 10.659
408 LA060689-0099 17 10.635
408 LA020289-0156 18 10.547
408 LA050489-0061 19 10.495
408 LA101489-0074 20 10.342
Running time: 39 ms

BM25 with Query Expansion
$R = 10, E = 25$

408 LA101490-0142 1 64.878
408 LA101390-0102 2 61.602
408 LA062290-0070 3 58.849
408 LA103190-0052 4 56.881
408 LA120389-0130 5 52.349
408 LA101289-0148 6 49.630
408 LA082189-0033 7 47.840
408 LA061989-0095 8 47.024
408 LA101489-0074 9 46.639
408 LA091989-0049 10 46.637
408 LA102090-0035 11 46.591
408 LA082990-0118 12 46.044
408 LA092089-0027 13 45.441
408 LA080189-0042 14 44.945
408 LA100890-0072 15 44.270
408 LA101589-0180 16 41.045
408 LA100590-0178 17 40.897
408 LA080289-0005 18 40.362
408 LA060689-0099 19 39.340
408 LA101290-0181 20 39.138
Running time: 85 ms

Bibliography

[1] Bodo Billerbeck and Justin Zobel. Questioning query expansion: an examination of behaviour and parameters. In *ADC '04: Proceedings of the 15th Australasian database conference*, pages 69–76, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[2] Nicolai Dahl and Daniel Smith. Assignment 1: Indexing. Technical report, RMIT, September 2013.

[3] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, 2008.

[4] Oracle. Class hashmap<e>. `http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html`, 2011.

[5] Oracle. Class priorityqueue<e>. `http://docs.oracle.com/javase/6/docs/api/java/util/PriorityQueue.html`, 2011.

[6] S E Robertson and S Walker. Microsoft cambridge at trec-9: Filtering track.

[7] S E Robertson and S Walker. Okapi/keenbow at trec-8.

[8] Falk Scholer. Information retrieval evaluation, slideshow, 2013.

[9] Falk Scholer. Query expansion and relevance feedback, slideshow, 2013.

[10] Eric Weisstein. Heap. `http://mathworld.wolfram.com/Heap.html`, 2013.