Minimizing Posting List Cost[*]

Edward Hills
University of Otago
Dunedin, New Zealand
ehills@cs.otago.ac.nz

ABSTRACT

Information Retrieval is primarily concerned with searching through a document collection given a query, and returning a set of documents which could be relevant to the users request.

This task usually requires searching through the entire document collection to find which documents may be relevant. In large document collections this can be time consuming and costly. This paper looks at ways to minimize the number of documents that are examined but still being effective in returning the documents most likely to be relevant.

1. INTRODUCTION

Searching through document collections which are large can mean when it comes to query time that all documents that are indexed by one or more of the search terms will be examined. When you have a large document collection such as the TREC terabyte collection, searching every document that is indexed by even a relatively rare term can mean searching through thousands or even millions of documents.

By reducing the amount of documents searched by a particular query you do run the risk of limiting the amount of documents returned which are relevant to the user. Keeping the accuracy of your list returned is a top priority when searching through a minimal amount of documents.

There are a number of techniques in which the document list traversal can be minimized while still maintaining high accuracy. This paper looks at three techniques:

1. Index ordering by query-independent measures, Ferguson and Smeaton [2]
2. The Nearest Neighbour Problem In Information Retrieval, van Rijsbergeni and Smeaton [6]
3. Filtered Document Retrieval with Frequency-Sorted Indexes, Persin, Zobel and Sacks-Davis [5]

I will examine document minimization further and discuss the main research questions and contributions of each paper in 2., talk about how these techniques are linked in 3., then propose two new ideas in 4. and discuss my final findings in the Conclusion.

2. DOCUMENT MINIMIZATION

By minimizing the document collection that must be evaluated we can avoid unnecessary calculation which slows down performance, lengthens postings list and increases the inverted index file size.

This area of research has been looked into since the early ’80s and a multitude of techniques have arisen ranging from Query caching [3] and two-tier architectures [1] to nearest neighbour searching [6] which we talk about later.

These different techniques are not entirely mutually exclusive as some may think but in some cases can be built upon layer by layer such as the *quit* and *continue* strategies [4] which are used in a variety of different techniques.

With todays web based document collection growing larger and larger the need to weed out documents which are spam or of poor quality to the user needs to be addressed. By removing documents which are of poor quality not only increases the performance of the search engine (as there are less documents to search through given a certain query) but also increases the likelihood that only relevant documents are returned to the user. Removing web documents from a document collection is discussed in the next section.

2.1 Research Questions & Contributions

2.1.1 Index ordering by query-independent measures

This paper from Ferguson et al. aims to remove html documents from its index in a query-independent manner. This means that instead of at query-time and having to wade through all documents for each query and decide at query-time that you do not want that document, you instead check to see if you want that document in the first place.

[*] A summary of previous papers and a suggestion of new ideas

Ferguson et al. have come up with a range of heuristics for determining which documents are likely to be spam or of poor quality. A common formula for finding good quality documents is PageRank which gives a high priority to documents it thinks is best. PageRank is used as a query-independent document filter but this paper hopes to do better in the quality of documents returned.

Some heuristics developed include:

- Access counts
- Information-to-noise ratio
- Document cohesiveness
- Document structure and layout
- Term-specific sorting
- Global BM25 sorting

See Ferguson et al. [2] for details about all heuristics.

The two main new contributions of this paper are *term-specific sorting* and *global BM25 sorting.*

Term-specific sorting uses the term weighting approach of BM25 and applies that to each document in each postings list separately and provides a pre-calculated BM25 score for each posting list. This is relatively close to what would be used at query-time however is lacking the summation of each query term score. It is also important to note that there is no need to maintain information such as *df(t).*

Global BM25 sorting is similar to that above but keeps track of the importance of each term. This does cause problems however as it doesn't optimize for Zipf's Law as it gives a higher rating for terms which are very infrequent rather than moderately infrequent. To get around this they imposed a threshold in which terms which do not appear in a document more times than the threshold will not be added. To calculate the threshold they used a large query log from the *Excite* search engine from 1999. They generated two alternative global BM25 measures:

Global BM25 Query Log Terms (QLTs): global BM25 scores by including only terms used in the query log.

Global BM25 Query Log Term Frequency (QLTF): considers the frequency that each term occurs within the query log (i.e. idf).

The results of this paper found that by combining the best static measure (which turned out to be access counts) and the global BM25 sorting technique at retrieval-time (as well as normal BM25 at query-time) that they can achieve a higher P10 value than a conventional IR index by processing only 15% of the postings list. This is a huge reduction of 85%, however the *Mean-average precision (MAP)* was lower than that of the conventional index (0.282 compared with 0.304). However the P10 and MAP trade-off is worth it as most uses are not likely to go past the first page of results so the lower MAP value is unlikely to have a real effect on the user.

2.1.2 The Nearest Neighbour Problem in Information Retrieval

This paper from Smeaton et al. [6] aims to reduce the number of documents that need to be scanned when scanning through the postings list for each term in the given query. It does this by using nearest neighbour classification and proposes a new algorithm for doing this.

The hopes of this paper was to develop an algorithm which would solve the nearest neighbour problem in IR. That is given a document it wishes to find all nearest neighbours of that up to a given threshold. This algorithm also has the added benefit of only checking the same document once, even if indexed multiple times by multiple terms in the search query.

I describe the algorithm below:

Maintain two sets, R and S. R contains all documents which are final candidates for the set of nearest neighbours. Where the maximum size of R is given by the user (lets assume one for now). S contains the set of all documents discovered so far whether in R or not.

Then given a query containing a set of search terms, order the terms based on the order of their increasing frequencies of occurrence within the entire document collection. Then add every document containing the term into S, the nearest neighbour will be found among this set.

By assuming that the number of co-occurrences of index terms between a document and a query is zero, then the similarity is zero, then it is possible to remove documents which are not indexed by at least one search term from the nearest neighbour search.

When checking each document found for a particular term, first check it is in S and if it is not then calculate a similarity using one of the formula below and add it to S.

Similarity Measures:

- Simple
- Ivie
- Dice
- Cosine
- Jaccard
- Overlap

If the value returned is greater than the best value so far then this document becomes a candidate and is added to R. After looking through each document for a given term it is then possible to calculate the maximum possible similarity value between among the documents which haven't been discovered yet. If this upper-bound that is calculated is less than the best value found so far then terminate.

This surprisingly fairly simple algorithm managed to save the number of document comparisons needed by about 50-60%. It is important the document collection does affect the results as explained in the paper, however an average amount of comparisons needed of about 40% was accepted.

2.1.3 *Filtered Document Retrieval with Frequency-Sorted Indexes*

Persin et al. [5] aimed to determine which documents are more likely to be highly ranked by organizing the inverted file by descending term frequency. By doing this the test data used 2% of memory that a standard implementation would, cpu and disk use was also reduced. This paper also showed that frequency sorting can decrease the index size regardless of compression used.

This paper discusses two points:

1. Limiting size of accumulators
2. Sorting postings list by term frequency

I will now describe the algorithm for limiting the number of documents in the accumulator:

The query terms are first sorted by weight so that important terms are processed first. An insertion s_{ins} and addition s_{add} threshold are then calculated where $s_{add} <= s_{ins}$. As the postings list is processed for each term the partial similarity $sim_{q,d,t}$ of the term is compared to the thresholds. If $sim_{q,d,t} >= s_{ins}$ then the document is likely to be important so $sim_{q,d,t}$ is added to the documents accumulator value (or one will be made). However if $s_{add} <= sim_{q,d,t} < s_{ins}$ then the document is probably not important but important enough to affect the outcome so $sim_{q,d,t}$ is added to the documents accumulator, or if no accumulator exists then nothing is done. And obviously if $sim_{q,d,t} < s_{add}$ then it is not important and nothing is done. Where $s_{ins} = c_{ins} \cdot S_{max}$ and $S_{add} = c_{add} \cdot S_{max}$.

The above basically means that if there a large number of documents with high similarity then looking at ones with low similarity is a waste of time and by having the S_{ins} threshold we can avoid looking at some documents all together. These allow us to save cpu usage (as we no longer need to need look at some documents) and memory usage (fewer documents in the accumulators).

This algorithm does however require that suitable constants are chosen, too higher values means there will be a lower amount of documents returned and too low means more processing will be required and the benefit will be lost.

It then works out with a bit of rearranging and by substituting the definitions of the term weighting the s_{ins} and s_{add} can be calculated as frequencies f_{ins} and f_{add}.

However no real improvement is available yet because we must still process each term in the query against the thresholds. This can be easily avoided by ordering the postings list by descending frequency and then checking the threshold and if it is below the threshold simply stop processing the postings list.

The above then imposes a new problem however which is compression. Because the document ids are no longer in order the compression will be less effective, but this can be overcome by organizing the document ids grouped by frequency.

This set up was then run on the Wall Street Journal database and vast improvements in all areas were found. Their experiment found they could reduce memory requirements from 173,000 to 4,000 accumulators; reduce the size of data requested from disk from 532 Kb to 157 Kb; and reduce cpu time from 3.18 to 1.20 seconds. These are all possible by the filtering algorithm described above and then by understanding that if we sort the list on term frequency then only the first part of each list will be examined saving in a lot of processing time.

2.2 Relationship

The relationship between the three papers described above is fairly obvious. The 'main' aim of these papers is to minimize the number of documents that need to be examined to fulfill a query.

The first paper does this by running query independent heuristics and limiting the number of documents put in the postings list to begin with, filtering out documents likely to be of poor quality and then sorting them on a global BM25 ranking.

The second does this by looking at the nearest neighbour problem in which it gets a threshold value and then stops processing the postings list if the next document is not greater than the value.

The third then limits the number of documents put into the accumulator and then sorts the postings list by frequency so that only the top part of the postings list needs to be examined and hence saves on a large amount of resources.

The relationship here is extremely strong and the papers ideas are built off one another.

Below I describe two new ideas for possibly limiting the number of documents needing to be processed.

3. FUTURE WORK

I have come up with two new possible solutions for minimizing the number of postings needing to be examined.

The first question I will pose is similar to Ferguson et al. in which I propose a new heuristic for removing or minimizing the ranking of web documents that have a high out-going-link-to-content ratio.

The second question that I will propose is another web document heuristic in which I believe it would be beneficial if we were able to filter documents which have been created for the sole purpose of boosting search-engine-optimization. By removing these documents we limit the number of poor quality documents and also punish those for using techniques that are usually discouraged.

3.1 Question 1

As has been discussed previously, removing documents as early as possible which are not likely to affect our search results but can speed up our search is ideal. This is why I propose that when looking at documents from the web and deciding whether they should be added or not (such as by using the heuristics discussed in Ferguson et al., lowering the rank of pages that have a high content-to-outgoing-link ratio could be beneficial.

The use case you could imagine is a page which contains a list of links to download a particular program. Due to some ranking functions these pages may get a higher ranking due to the number of out-going links in the document. I propose that having these documents ranked higher can affect what the user wants.

If you have a query such as ”Galaxy SII Ice Cream Sandwhich” You are probably looking for information about the new Android operating system, rather than a list of links of where to download or buy the product. I propose that there is a tipping point in which if the number of out-going links is higher by some threshold than the ranking value of the content itself, that the overall rank of this document should be lowered. There is of course many exceptions to this where a user may explicitly want this such as terms like ’buy’ or ’download’ etc.

To avoid the last point discussed above in which a user does want these links (based on select query terms), it is possible to look at query traces (say from Google) in which we can analyse which terms in a particular query had the users go to the page containing the high link-to-content ratio.

Once recognising to a certain degree, which terms the users used, we can then find where the exceptions occur and otherwise calculate a value which represents the link-to-content ratio.

By doing this you would be more likely to be returned documents containing more content than links on a particular page.

3.2 Question 2

Following on from the approach above my next approach is to look at a mixture of the term-frequency and the proximity of the term. My idea is this, if a term is found multiple times (above some threshold) and is found within some small number of words away, and this formula happens multiple times in itself, then there is the possibility that this page has been specifically designed for search-engine-optimisation (SEO). There is also the likely possibility that they do not repeat the same word over and over but instead use synonyms, this means that the system would have to look up a thesaurus to check (adding overhead). If the system detects this pattern has occurred more times than some threshold then the page should be detected as solely used for SEO and discarded.

Of course this approach can run into multiple problems, most search engines use these pages to detect good pages from the bad, *i.e.* if it contains the word (or similar word) multiple times and within close proximity its probably on the subject and deemed to be important. This is why the system must be able to detect if the number of times this pattern appears is more frequent than other actual content.

This approach definitely has some areas of concerns such as the added overhead of consulting a thesaurus and the possibility of discarding useful (if not possibly some of the top) documents. However I feel that with some investigation, useful threshold values may be found and then this would be a good technique for limiting the number of pages solely used to boost search-engine ranking.

4. CONCLUSIONS

In this paper I have shown three ways in which it is possible to reduce the number of documents that needs to be examined when answering a particular query. By minimizing the number of documents to process at query-time we save on memory, cpu and even disk usage. These combined can drastically improve performance and in some cases the quality of the documents returned.

Ferguson et al. took a query-independent approach to web documents by using a range of heuristics and Global BM25 sorting. Smeaton et al considered the problem of finding the nearest neighbour and limiting the number of documents it has to look at in the postings list. Finally Persin et al. took two approaches, the first being to minimize the number of accumulators by using some threshold values, and the second is to sort the postings list by term frequency and only taking the top part of the list thus reducing the number of postings traversed.

I then proposed two new ways of limiting the number of documents needing examined, the first approach I propose is to look at link-to-content ratio and lower the ranking, or remove altogether, the documents in which (by some appropriate) there are more links than content. The second method I proposed is another heuristic for web documents in which it may be possible to remove documents that are solely used for search-engine-optimization. Removing documents would not only help remove the number of documents found but also punish those that are using SEO ranking techniques that are usually discouraged.

So as you can see by removing the number of documents that we need to examine at query-time increases performance and in some cases the quality of documents returned. I believe this to be an important field as the number of documents continues to grow and with it the number of poorer quality documents and time it takes to process them.

5. REFERENCES

[1] T. Fagni, R. Perego, F. Silvestri, and S. Orlando. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Trans. Inf. Syst.*, 24(1):51–78, Jan. 2006.

[2] P. Ferguson and A. F. Smeaton. Index ordering by query-independent measures. *Inf. Process. Manage.*, 48(3):569–586, May 2012.

[3] R. Lempel and S. Moran. Predictive caching and prefetching of query results in search engines. In *Proceedings of the 12th international conference on World Wide Web*, WWW ’03, pages 19–28, New York, NY, USA, 2003. ACM.

[4] A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Trans. Inf. Syst.*, 14(4):349–379, Oct. 1996.

[5] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *J. Am. Soc. Inf. Sci.*, 47(10):749–764, Sept. 1996.

[6] A. F. Smeaton and C. J. van Rijsbergen. The nearest neighbour problem in information retrieval: an algorithm using upperbounds. *SIGIR Forum*, 16(1):83–87, May 1981.