Top Subset Retrieval on Large Collections Using Sorted Indices

Paul Ferguson, Alan F. Smeaton, Cathal Gurrin and Peter Wilkins
Centre for Digital Video Processing, Dublin City University, Ireland.
{pferguson, asmeaton, cgurrin and pwilkins}@computing.dcu.ie

ABSTRACT

In this poster we describe alternative inverted index structures that reduce the time required to process queries, produce a higher query throughput and still return high quality results to the end user. We give results based upon the TREC Terabyte dataset showing improvements that these indices give in terms of effectiveness and efficiency.

Categories and Subject Descriptors: H.3.3 Information Storage, Information Search and Retrieval
General Terms: Performance, Experimentation
Keywords: Sorted Inverted Index, Large-Scale Retrieval

1. INTRODUCTION

The task of developing a fast and effective search engine to deal with many billions of web pages is very difficult. While indexing billions of web pages is beyond the needs of most organisations, indexing large test collections (such as the TREC GOV2 collection of over 25,000,000 pages) is often necessary. In this poster we examine alternative indexing techniques to make this possible with even very limited resources.

In our previous work [2], we presented a search engine architecture for an efficient Terabyte search engine. We distributed GOV2 across four leaf search engines and used an aggregate engine to combine search results. In an effort to produce an even more cost-effective means to search the collection we indexed the entire collection on a single Pentium 4, 2.6GHz machine, with 1.5GB of RAM. However when reducing the cost of indexing a collection by reducing the number of machines used, problems with index structures become more acute. It was with this in mind that we examined alternative index structures.

2. TOP-SUBSET RETRIEVAL

The main challenge as we see it, is not in the indexing of the collection on a single machine but in performing retrieval. This can be seen when using the TREC Terabyte topics titles from 2004 as queries, after removing stopwords there remains an average of over a million documents associated with each term, and with an average of 3.1 terms per topic that amounts to over 3 million similarity calculations per query. In queries with more popular terms there are over 12 million similarity calculations. Working on a standard desktop machine, this is difficult to achieve efficiently.

A possible solution to this is to choose to process only a certain number of documents associated with each term. As described in [2] one of the index structures supported by our search engine is similar to a conventional inverted index. Fundamentally, for each term in a collection-wide lexicon, there is an object that contains the list of documents where that term occurs, and it's corresponding term frequency (TF). This structure allows easy sorting of documents associated with each term, which provides a mechanism to allow the retrieval of only a top subset of documents associated with each term and still attain relatively high accuracy, with only a fraction of overall memory requirements.

If the documents are sorted naively we cannot attempt to take only a top subset of documents for each term and hope to find a large portion of relevant documents. By re-ranking documents in the postings lists in descending order based on their term frequency, similar to Persin et al in [3] and first introduced in [4], the most influential documents for each term are processed first and so processing only a limited number of documents for each term can produce high quality results with only a fraction of the computational costs. Persin also describes methods to compress this type of inverted index and presents means by which their frequency sorted indices can be stored in less space than the conventional document-order inverted index.

In addition to this term frequency sorted index we also created an alternative index structure: we sorted the index by the normalised TF (NTF = TF/document length(dl_t)) to rank based on the term's overall influence on that document rather than purely on the number of occurrences of the term. It would seem intuitive to think that this index, sorted based on the NTF would perform better than one sorted based on TFs. However we found the opposite to be the case, as was the case for Anh et al in [1]. This poor performance in comparison with TF sorting may be explained by the high ranking of small documents in the inverted index which would not be normalised greatly due to their small length. In order to promote important documents with not just overly long TFs (as in the case of TF sorting) and not overly short documents (as can be the case in NTF sorting), we devised the following alternative

SIGIR'05, August 15–19, 2005, Salvador, Brazil.
ACM 1-59593-034-5/05/0008.

method of sorting, (calculated based on divergence from the average document length) which would penalise both long and short documents, and then combine this with their TF to give an overall ranking as to the importance for each document associated with each term.

$$Weight_{tf} = \log(BiDist_{avg} + e) \times log(TF + e) \quad (1)$$

$$BiDist_{avg} = \begin{cases} \frac{dl_t}{avg_{dl}} & \text{if } dl_t \leq avg_{dl} \\ 1 - \frac{dl - avg_{dl}}{max_{dl} - avg_{dl}} & \text{otherwise} \end{cases} \quad (2)$$

where e is the base of natural logarithms, avg_{dl} is the average and max_{dl} is the maximum document length.

This sorting scheme works as follows: $BiDist_{avg}$ is a measure of the distance of the document length from the average document length, and becomes smaller the further the document length deviates from the average. When this is combined with the TF to calculate $Weight_{tf}$, it gives a good measure of the term's overall influence on the document. A variant on this is to combine the NTF: with the $BiDis_{avg}$ measure to produce a new weighting ($Weight_{ntf}$):

$$Weight_{ntf} = \log(BiDist_{avg} + e) \times \log(NTF + e) \quad (3)$$

An alternative method of sorting is to combine the NTF with a measure ($Dist_{avg}$), which only penalises short documents, as the longer documents would have been penalised enough already by being divided by their large dl_t:

$$SWeight_{ntf} = \log(Dist_{avg} + e) \times \log(NTF + e) \quad (4)$$

$$Dist_{avg} = \begin{cases} \frac{dl_t}{avg_{dl}} & \text{if } dl_t < avg_{dl} \\ 1 & \text{otherwise} \end{cases} \quad (5)$$

3. EXPERIMENTS AND RESULTS

We conducted our experiments using the TREC Terabyte 2004 data, and running automatic queries from the title field of the topics. We present performance details using different forms of sorted indices and show how utilising only the top subset of each of these indices can effect system performance as measured in terms of MAP and Precision at 10. Overall

Table 1: Index Names and Descriptions

Index Name	Description
DocSorted	Sorted by document number
NTFSorted	Sorted by the NTF
TFSorted	Sorted by TF
WeightTFSorted	Sorted using $Weight_{tf}$ (1)
WeightNTFSorted	Sorted using $Weight_{ntf}$ (3)
SWeightNTFSorted	Sorted using $SWeight_{ntf}$ (4)

the WeightTFSorted index performed best in terms of Precision at 10(P@10) and Mean Average Precision(MAP). The WeightNTFSorted and SWeightNTFSorted indices achieved a clear and consistent improvement over all other index formats in terms of P@10, however with WeightNTFSorted this gain is at the expense of MAP in which it performs poorly.

It is interesting to note that for TFSorted, WeightTFSorted and SWeightNTFSorted there is a degradation in P@10 the more documents that are evaluated. While WeightedNTF managed to maintain a regular level of consistency.

Perhaps the best improvement to be seen from these new indices is clear increase in precision at 10 (P@10) that the

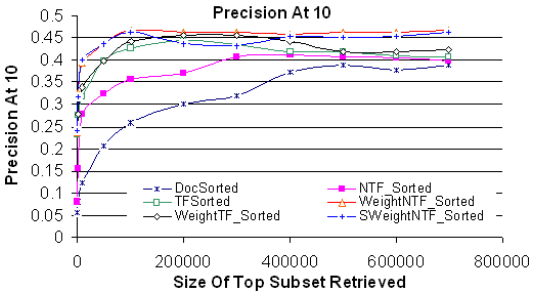

Figure 1: Precision at 10 Comparisons.

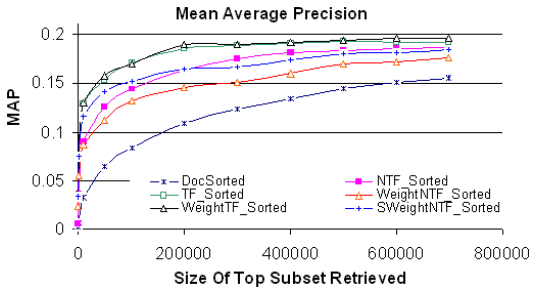

Figure 2: Mean Average Precision Comparisons.

WeightedNTF and SWeightNTF indices gives. At only 100,000 documents processed they give P@10 values of 0.4673 and 0.4612 respectively, which none of the other indices can achieve, even after processing as much as 700,000 documents for each term. These type of indices could clearly be useful in a web search engine(or similar) where the typical user wants a fast response time with high precision results at the top of the ranked list and generally will not browse past the top 10-20 results.

4. CONCLUSIONS

We have presented additional techniques for sorting inverted indices which give substantial improvements over conventional document sorted indices, and moderate improvements over TF sorted indices. These experiments have been carried out on a single machine, but we have achieved similar improvements using a distributed search engine architecture, allowing these approaches to scale up to work with larger collections.

Acknowledgement: This work was supported by Science Foundation Ireland, under grant number 03/IN.3/I361.

5. REFERENCES

[1] V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. SIGIR, 2001.

[2] P. Ferguson, C. Gurrin, P. Wilkins, and A. F. Smeaton. Físréal: A low cost terabyte search engine. In *Proceeding of European Conference in IR*, March 2005.

[3] M. Persin, J. Zobel, and Ron-Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *"J. American Society of Information Science"*, 47, 1996.

[4] A. F. Smeaton and C. J. van Rijsbergen. The nearest neighbour problem in information retrieval: an algorithm using upperbounds. In *Proc. of ACM SIGIR conference on Information storage and retrieval*, 1981.