Term-ordered Query Evaluation versus Document-ordered Query Evaluation for Large Document Databases

Marcin Kaszkiel Justin Zobel
Department of Computer Science, RMIT, GPO Box 2476V, Melbourne 3001, Australia
{marcin,jz}@cs.rmit.edu.au

Abstract *There are two main families of technique for efficient processing of ranked queries on large text collections: document-ordered processing and term-ordered processing. In this note we compare these techniques experimentally. We show that they have similar costs for short queries, but that for long queries document-ordered processing is much more costly. Overall, we conclude that term-ordered processing, with the refinements of limited accumulators and hierarchical index structuring, is the more efficient mechanism.*

Techniques for evaluation of ranked queries on large text collections are well developed. In a typical ranked system each document in the collection is heuristically assigned a score representing its similarity to the query, and the documents with the highest scores are returned to the user. The most efficient of the current systems are based on inverted files; query evaluation involves fetching of inverted files, processing them to determine similarity values, then fetching of the top-scoring documents. Typically the number of documents fetched is small, whereas a high proportion of documents in the collection will have a non-zero similarity.

These evaluation techniques are used in many applications, ranging from the short queries posed to Internet search engines, typically of two to five words, to extended queries posed by searching experts and long queries generated by techniques such as query expansion and relevance feedback. These techniques can provide better effectiveness than straightforward ranking, but involve many more query terms and are thus lead to increases in query evaluation costs.

There are two principal techniques for evaluation of ranked queries: *term-ordered* (TO) processing and *document-ordered* (DO) processing. Both are based on inverted files [2, 7], a data structure containing, for each term, a sorted inverted list of the identifiers of the documents in which the term appears and the frequency of the term in each document.

We compare TO and DO processing experimentally. In TO processing, the inverted list of each term is processed in full before the next is considered. For each document d in which each term appears, a *partial* similarity value is computed from the inverted list. Each partial similarity value is added to an accumulator corresponding to d. When processing of the inverted lists is complete, the accumulators are sequentially processed to normalise them with regard to document length and to identify the highest normalised scores. This style of processing is used, for example, in SMART [4] and MG [6].

A variant on TO processing is to limit the number of accumulators, to say 2% of the total number of documents, and to structure the lists hierarchically [2]. In this TO′ or "skipping" style of processing, rare terms are considered first, and are free to add accumulators, up to the limit, as new document identifiers are observed. When the accumulator limit is reached no further accumulators can be added, and only a fraction of the information in the subsequent inverted lists is used; the hierarchically structuring allows this information to be skipped, significantly reducing CPU time (for list and accumulator processing) and memory requirements (for accumulators). Reducing the limit on the number of accumulators simultaneously reduces both memory requirements and processing time, but also reduces the ability of the mechanism to identify relevant documents, that is, reduces its effectiveness.

Another variant of TO processing is to reorder lists by in-document frequency, so that larger partial similarities are to the front of each inverted list. We do not experiment with frequency-sorting here (as it is incompatible with the broader aims of our research, into passage ranking), but it allows significant gains over TO′ processing [3].

In DO processing, the inverted lists for all the query terms are processed simultaneously, in document order. At each stage the least document identifier d in any list is found, all information about d is consumed from the front of all lists in which d is referenced, a similarity value is computed for d, and processing proceeds to the next least document. Only a small number of intermediate results—final similarity values—are required.

Thus DO processing has the advantage of not requiring memory space for accumulators, but has several potential disadvantages. First, either enough buffer space must be allocated to hold all inverted lists simultaneously or query evaluation times will rise because several disk accesses are required to fetch each inverted list; in contrast, with TO processing it is feasible to fetch the whole of all but the longest lists, because lists are fetched in turn. Second, as query length increases the cost of identifying the list with the least document identifier will gradually dominate, as this cost is $O(n \log n)$ in the number of query terms, while all other costs are asymptotically constant or linear. Third, with DO processing it is not possible to use optimisations such as skipping.

Turtle and Flood's analysis of the performance of ranking algorithms in limited memory suggests that DO is more efficient than TO [5]. However, the model of processing used in this analysis is based on simplifying assumptions that are not valid; in particular, these assumptions imply that the processing costs are linear in the volume of inverted index information required (which is false for DO)

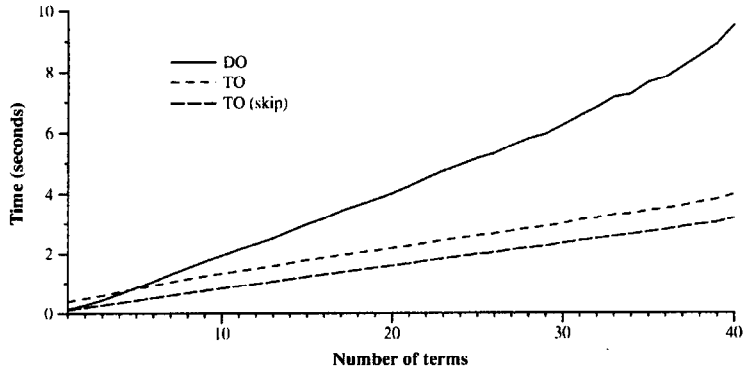

Figure 1: *Elapsed time for each processing method on TREC disks 2 and 4.*

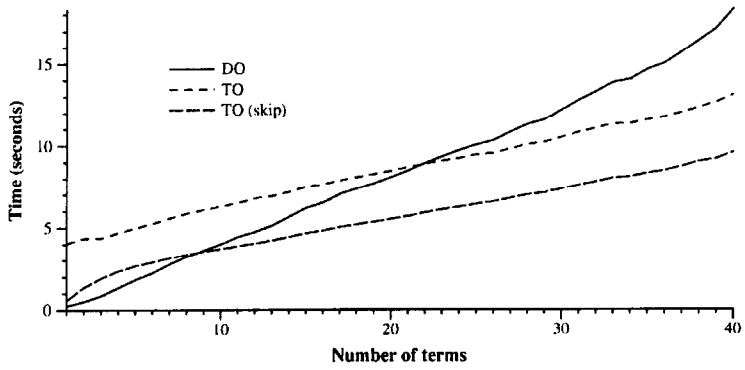

Figure 2: *Elapsed time for each processing method on TREC disks 2 and 4, after division into short documents.*

and that seek and latency times can be neglected.

Using the MG prototype text database system we have compared TO, TO′, and DO processing experimentally. In our first group of experiments we used TREC disks 2 and 4 [1], of about 530,000 documents, and queries 251–300. To simulate queries of varying length we generated 1-word queries by taking the first word of each query, 2-word queries by taking the first two words, and so on. We then measured memory requirements, speed, and effectiveness.

We found that all three methods have similar effectiveness. TO processing required about 2.1 Mb of memory; DO required up to 1.2 Mb; and TO′ required up to 0.3 Mb. In the case of DO inverted lists were prefetched and buffered, a strategy that increases memory requirements but significantly reduces evaluation time. Elapsed evaluation time is shown in Figure 1. As can be seen, TO′ processing is always the most efficient method, and the cost of DO rises rapidly as query length increases.

We also compared TO, TO′, and DO processing over a larger collection, formed by dividing the documents on TREC disks 2 and 4 into fragments of 50–500 words each, giving almost 8,000,000 documents. Where possible these divisions were made at a sentence or paragraph boundary. Evaluation time is shown in Figure 2. For short queries DO processing is more efficient; for other queries TO′ processing is clearly preferable. Both strategies used up to 3 Mb of memory, compared to 30 Mb for TO processing.

Both methods can have their memory requirements smoothly reduced in response to system load, in the case of DO by reducing the size of buffers for storing inverted lists and in the case of TO′ by reducing the number of accumulators. For DO, tight memory constraints will cause the system to thrash. For TO′, tight memory constraints will reduce effectiveness but increase query throughput.

Overall, we conclude that TO′ processing using limited accumulators and skipping is effective and efficient, and is the preferred query evaluation mechanism for large document databases.

References

[1] D. Harman. Overview of the second text retrieval conference (TREC-2). *Information Processing and Management*, 31(3):271–289, 1995.

[2] A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379, October 1996.

[3] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *J. American Society of Information Science*, 47(10):749–764, 1996.

[4] G. Salton. *Automatic Text Processing: The transformation, analysis, and retrieval of information by computer.* Addison-Wesley, 1989.

[5] H. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *Information Processing and Management*, 31(6):831–850, 1995.

[6] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and indexing documents and images.* Van Nostrand Reinhold, 1994.

[7] J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions On Database Systems.* To appear.