Access-Ordered Indexes

Steven Garcia **Hugh E. Williams** **Adam Cannane**

School of Computer Science and Information Technology,
RMIT University, GPO Box 2476V, Melbourne 3001, Australia.
{garcias,hugh,cannane}@cs.rmit.edu.au

Abstract

Search engines are an essential tool for modern life. We use them to discover new information on diverse topics and to locate a wide range of resources. The search process in all practical search engines is supported by an inverted index structure that stores all search terms and their locations within the searchable document collection. Inverted indexes are highly optimised, and significant work has been undertaken over the past fifteen years to store, retrieve, compress, and understand heuristics for these structures. In this paper, we propose a new self-organising inverted index based on past queries. We show that this access-ordered index improves query evaluation speed by 25%–40% over a conventional, optimised approach with almost indistinguishable accuracy. We conclude that access-ordered indexes are a valuable new tool to support fast and accurate web search.

Keywords Information retrieval, web search engines, inverted indexes, efficiency.

1 Introduction

Almost everyone with a computer searches the Web and the collections of searchable resources are staggeringly large. Users search for information on topics as diverse as travel, health, education, entertainment, and science, and for resources including forms, legislation, books, and research. However, despite the diversity of needs and the difficulty of the task, users expect accurate and almost instantaneous responses from web search engines.

Innovative, scalable, and fast solutions are essential to support web search. A key component of all practical search engines is an *inverted index* structure that supports query evaluation. An inverted index consists of two components: first, a *term dictionary* that contains all terms that occur in the searchable collection; and second, for each term, a *postings list* of locations at which that term occurs. In most search engines, the vast majority of the searchable terms are words, and the postings specify where words occur within documents or resources. Inverted index design is a key focus of search engine efficiency research.

Around 80% of queries that are posed to search engines are ranked or "bag of words" queries (Spink, Wolfram, Jansen & Saracevic 2001). An inverted index supports ranked query evaluation as follows. First, the terms from a query are looked-up in the term dictionary and the location of the postings lists are found. Second, the postings lists are retrieved from disk and processed sequentially. Third, for each document that appears in the postings list, an *accumulator* is updated with the result of a similarity computation; the similarity function approximates the likely relevance of the document to the query. Last, the accumulators are partially sorted, and the highest-scoring documents are identified, retrieved, and shown to the user in summary form.

Inverted index design is crucial to search engine performance. For example, the choice of the search structure for terms has a profound impact on overall speed: an optimised hash table is around five times faster than an optimised splay tree (Zobel, Heinz & Williams 2001). Similarly, the choice of compression scheme for postings lists is crucial: recent work has shown that a simple bytewise compaction scheme halves query evaluation times compared to the fastest bitwise compression approach (Scholer, Williams, Yiannis & Zobel 2002) (which, in turn, is faster than retrieving uncompressed data). At a higher level, postings list design, query evaluation strategy, and evaluation heuristics can all have an impact on overall system speed.

Novel postings list organisations can permit fast and accurate query evaluation. Conventionally, postings within each list are ordered by increasing document number. This approach supports heuristic query evaluation where only selected postings lists are processed (for example, common words may be ignored), but any list that is selected must be decoded in its entirety. In contrast, in frequency-ordered (Persin, Zobel & Sacks-Davis 1996) and impact-ordered (Anh & Moffat 2001, Anh & Moffat 2002) indexes, postings within each list are ordered by decreasing impact on the ranking similarity function. Using this approach, postings lists may only need to be partially decoded, since those postings with high impact are processed first, and those with low impact may not need to be decoded to find an accurate ranking of documents. Frequency-ordered indexes have been shown to improve query evaluation time by reducing memory and processing requirements, while impact-ordered indexes have also been shown to improve evaluation time while increasing the accuracy of search.

In this paper, we propose a new index organisation that we call *access-ordered* indexes. We have observed that over a large number of queries, only a fraction of the documents in a collection are ranked highly in response to those queries. Moreover, similarly to other trends such as the frequency of occurrence of document and query words, the access frequencies follow an inverse power-law (or Zipfian) distribution. Our access-ordered approach takes advantage of this phenomena: postings for documents that are frequently returned in response to queries

are stored at the beginning of the lists, while less-frequently retrieved documents are stored towards the end. Query evaluation stops when a heuristic threshold is reached, with the aim of returning accurate results in less time than a conventional approach.

Access-ordered index organisation requires that queries are available to establish document access frequencies. We show that after processing 200,000 queries, access-ordering leads to query evaluation that is over 20% faster than a document-ordered approach, while accuracy remains almost unchanged. We also show that access-ordered indexes may be a valuable component of a two-tiered search structure, where a much smaller access-ordered index is used to process queries that are likely to have similar results to past queries. Using this approach, we demonstrate that an *access-pruned* index is only 60% of the size of a conventional document-ordered index, while accuracy is unchanged and query evaluation over 40% faster.

2 Background

In this section, we present a background on ranked query evaluation. We focus on inverted indexes, and report on selected optimisations in index design to improve query evaluation efficiency.

Inverted indexes are an essential component of all web search engines and text retrieval systems (Witten, Moffat & Bell 1999). An inverted index has two components: a largely in-memory vocabulary of the terms that occur in the searchable collection, and largely on-disk postings that list the locations of occurrence of the search terms. Typically, searchable terms are words extracted from the text (Williams & Zobel n.d.).

For each term, there is one posting for each document that contains the term. Zobel and Moffat (1998) proposed a notation to formally describe such postings: each term t has postings $< f_{d,t}, d >$, where $f_{d,t}$ is the frequency f in document d. For example, consider the postings for the term "tigerland":

$$< 1, 107 > < 1, 114 > < 3, 1048 >$$

This postings list indicates that the word occurs once in the 107th document, once in the 114th document, and three times in the 1,048th document. A separate structure — called a *mapping table* — is used to resolve document numbers to disk locations where the documents themselves are stored.

This inverted index structure is sufficient to support ranked and Boolean query evaluation. In practice, other query types are supported by most retrieval systems. For example, web search engines support phrase queries, where the ordering and adjacency of words determines which documents are matches. To support such fine-grained measures, word position information is also stored in the inverted lists, but we do not discuss this further here.

Ranked query evaluation with an inverted index proceeds as follows. First, the query terms are searched for in the in-memory vocabulary. Second, for those terms that are found, the locations of the on-disk postings lists are determined. Third, the postings lists are retrieved. Fourth, the postings in the lists are processed sequentially. For each posting, a document *accumulator* is updated with the result of a function that computes the statistical similarity of the document to the query; we discuss this further next. Last, the accumulators are partially sorted to identify the best matches, the best matching documents are retrieved from disk, and summaries of the documents are shown to the user. There are many optimisations possible in this process, and we discuss some of these later.

A state-of-the-art similarity function is the Okapi BM25 formulation (Robertson & Walker 2000, Robertson, Walker, Hancock-Beaulieu, Gull & Lau 1992). This measure computes the similarity $\mathrm{sim}(q, d)$ of a document d to query q that contains the terms t as follows:

$$\mathrm{sim}(q, d) = \sum_{t \in q} \log \left(\frac{N - f_t + 0.5}{f_t + 0.5} \right) \times \frac{(k_1 + 1) f_{d,t}}{K + f_{d,t}}$$

As previously, $f_{d,t}$ is the frequency of term t in document d. The total number of documents in the collection is N, and f_t is the number of documents that contain t. K is $k_1((1 - b) + b \times L_d/\mathrm{avl})$, where L_d is the document length, avl is the average document length in the collection, and k_1 and b are constants that are set to 1.2 and 0.75 respectively.

The Okapi BM25 formulation uses statistics that are readily available to the retrieval engine. The values of d and $f_{d,t}$ are stored in the postings for each term. The constants N and avl are fixed for a collection and can be gathered as the inverted index is created. The value f_t is typically maintained with the vocabulary entries, and L_d can be computed from the mapping table that resolves document numbers to physical disk locations or it can be stored as a separate pre-computed entry. A detailed explanation of the Okapi BM25 formulation is presented elsewhere (Sparck-Jones, Walker & Robertson 2000).

The efficiency of ranked querying is highly dependent on the effective organisation and storage of postings lists. Conventionally, postings lists are organised so that the postings are sorted by increasing document order. Continuing our previous example, this permits differences to be taken between adjacent document identifiers prior to storage on disk:

$$< 1, 107 > < 1, 7 > < 3, 934 >$$

Since postings are processed sequentially, an original value is restored by summing all previous values; for example, the sum of 107 and 7 gives the original value of 114 for the second posting.

Document-ordered indexes have the advantage that taking the difference between adjacent document numbers improves the compressibility of lists. Improving compression improves query evaluation times, since disk bandwidth is a bottleneck in modern retrieval engines; for example, Scholer et al. (2002) show that compressing postings lists reduces average query evaluation times to around one-third of that of an uncompressed representation. However, a disadvantage of document-ordered lists is that each list must be decoded in its entirety in response to a query; the position of a document in the postings list has no correlation with its similarity to the query.

Optimisations of index processing are possible. A well-known technique is *stopping* (Witten et al. 1999) or ignoring the long lists of around 700 common terms such as "the", "of", and "therefore". This saves time in query evaluation for queries containing common words, but with some penalty to average accuracy. Moreover, some queries that consist of only common terms — such as "the who" or "to be or not to be" — cannot be evaluated at all. It is therefore unclear whether stopping should be used.

Another approach to improving efficiency is to limit the number of accumulators that are created during query evaluation. This has the dual benefits of limiting main-memory use and reducing the computational cost of maintaining values for documents with very low similarity to the query. In previous work (Moffat & Zobel 1996) it has been shown that limiting the number of accumulators to around 1% of the documents of the collection works well in practice.

When the accumulator threshold is reached, there are two possible approaches: first, to QUIT processing the current and remaining postings lists; and, second, to CONTINUE processing lists, but to only update existing accumulators. In experiments, Moffat and Zobel showed that the CONTINUE strategy was as effective as an unlimited number of accumulators, and that the QUIT strategy was less effective; we report similar results in Section 4. However, unsurprisingly, QUIT is the fastest approach. We report results with QUIT and CONTINUE in Section 4.

The previous approaches avoid processing all of the postings lists in response to a query. A different approach is to reorganise the lists themselves so that an individual list may not need to be completely decoded. There are two general approaches in this area: first, inserting additional pointers into the lists so that values can be *skipped* (Moffat & Zobel 1996); and, second, storing postings in a different order so that those that are most likely to be similar to a query are at the beginning of the lists. We do not discuss the former approach here but note that it is compatible with our novel scheme described in Section 3.

Persin et al. (1996) propose organising postings lists by decreasing frequency $f_{d,t}$ to give a *frequency-ordered* index. The motivation for storing postings by frequency is to correlate list ordering to impact on the ranking function. Using this approach, our example postings list is ordered as follows:

$$< 3, 1048 > < 1, 107 > < 1, 114 >$$

Differences can be taken between frequency values to improve compressibility but, because frequencies are typically much smaller than document identifiers, this is less effective than the document-ordered approach.

Recognising the importance of index compression on efficiency, Persin et al. suggest index organisation techniques that use the properties of frequency ordering to reduce redundancy and achieve an overall reduction in index size. In their work, they also suggest an innovative query evaluation technique where posting list $f_{d,t}$ values are tested against two thresholds. First, postings are compared to the threshold C_{ins}. Those postings that satisfy the test are considered significant to the similarity function, and therefore have an accumulator created where one does not already exist. Second, where a posting cannot satisfy the first condition, the posting is compared to a second threshold, C_{add}. The second threshold allows those postings that are not considered significant enough to justify a new accumulator to update pre-existing accumulators.

Similar to the work of Moffat and Zobel (1996), this two-tier approach reduces the memory required for accumulators during query evaluation. By organising the postings lists in decreasing $f_{d,t}$ value, a further optimisation can be applied where each postings list is processed only until the C_{add} test fails. Using this approach Persin et al. showed a significant reduction in the volume of postings decoded during query evaluation.

Another recent approach to list reorganisation is that of Anh and Moffat (2001, 2002) who noted that the similarity measures used by the *vector space model* are not well suited to the short number of terms contained in typical Web queries. They showed that for Web queries there is a tendency for individual terms to dominate accumulators during query evaluation, resulting in many highly ranked documents containing only a small subset of the query terms. To offset this effect they propose to normalise the *impact* of each document in the postings lists. The normalisation step has the effect of increasing the impact of terms that would typically have an almost insignificant contribution to accumulators during query evaluation, therefore reducing the likelihood of individual query terms dominating query evaluation, and allowing the documents that contain several query terms to be more highly ranked.

To efficiently store the impact values they quantise each impact value to a fixed bit integer without a negative effect on system accuracy. They also demonstrate that the quantised impacts can be directly utilised by the similarity measure.

In their *impact-ordered* scheme, Anh and Moffat order postings lists by impact value, that is, postings are organised by decreasing effect on the ranking function. For example, using this approach our postings list may be ordered as follows:

$$< 10, 1, 114 > < 7, 3, 1048 > < 5, 1, 105 >$$

Where each posting consists of the triple $< i_m, f_{d,t}, d >$, and i_m is the quantised impact of term t on document d. In our example, the impact of the word on the 114th document is greater than that of any other document in the list, therefore, it is placed at the head of our list.

Several optimisations can be made to this index structure. In their work they utilised a technique where postings were blocked by impact value to avoid redundancy. Using this structure they reported improvements in retrieval effectiveness with several variations of similarity heuristics during query evaluation. They also proposed several pruning techniques that benefit from the impact-ordered index structure.

3 Access-Ordered Indexes

In this section, we propose our novel *access-ordered* index organisation. Our aim in developing this approach is to investigate whether past queries can be used to identify documents that are likely to be relevant responses to future queries, and whether this can be used to organise an index for fast and accurate query evaluation. We begin by presenting observations on the access frequencies of documents, and then discuss our index design in detail.

Document Access Frequencies

Many behaviours in web retrieval follow an inverse power law distribution. For example, the frequency of word occurrences in large newswire collections (Williams & Zobel n.d.) is such that the commonest word ("the") occurs twice as often as the second most common word ("of"), which in turn occurs 1.1 times as often as "to", and so on; the ratios appear to vary slightly for Web collections, but the overall trend is the same. Similar behaviours are seen in the frequency of new words occurring in collection document sizes (where the number of words grows sublinearly with the collection size (Baeza-Yates & Ribeiro-Neto 1999, Williams & Zobel n.d.)), the use of query terms (where 9% of all query terms are drawn from a set of only 0.05% of unique terms (Spink et al. 2001)), and in the caching behaviour of web documents (Breslau, Cao, Fan, Phillips & Shenker 1999).

To investigate whether this class of behaviour extends to documents that are returned in response to queries, we carried out an experiment with more than 1.9 million ranked queries on a collection of around 1.6 million documents; we discuss the collection and queries in detail later in Section 4. For each query, we evaluated the similarity of the query to the documents in the collection using the Okapi BM25 formulation described in Section 2, and incremented a counter for each of the top 1,000 matching documents. We refer to this counter as the *access count*.

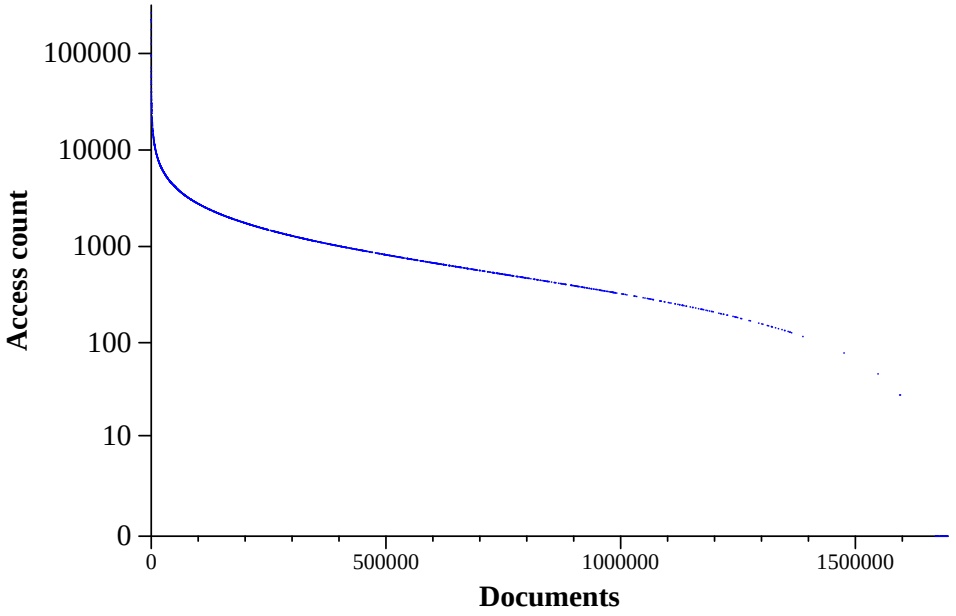

Figure 1: *A plot showing the number of times each document in the WT10g collection is ranked in the top 1,000 responses for around 1.9 million Excite queries. The Y-axis shows the frequency, and the X-axis is the rank of frequency. Points on the line are documents that are relevant to any of the TREC queries used in our experiments. The collections and queries are described in Section 4.*

A plot of the document access counts for the query set is shown in Figure 1. On the Y-axis, the graph shows the access count (the frequency with which the document appeared in the top 1,000 results). On the X-axis, the documents are organised by decreasing rank of frequency so that the most-frequently accessed document is shown at 0 and the least-frequently accessed at 1.6 million. The most frequently accessed document (TREC document WTX092-B35-396) has an access count of 316,282, that is, it appears in the top 1,000 results for around 16% of the queries. Indeed, the top 10% most frequently accessed documents account for almost 50% of all accesses. More than 76% of documents have access counts less than 1,000, that is, they appear in the answers for less than 0.06% of queries. In summary, perhaps as expected, access counts are highly skew.

The line that is plotted in Figure 1 does not show all documents from the collection. Instead, we have only shown the 2,617 documents that have been judged as relevant to any of the 50 TREC queries that are frequently used to compare the behaviour of web retrieval techniques; the TREC framework and queries are discussed in Section 4. This plot illustrates that documents that have been judged as relevant to user information needs are also those with high access counts: around 48% of the relevant documents are in the 10% of those with highest access counts. Indeed, only 8% of the relevant documents are in the half of the collection with the lowest access counts.

We concluded from this initial investigation that document accesses are highly skew when averaged over a large number of queries. In addition, we empirically observed that there is correlation between the access count of a document and its likelihood of being relevant to the user. With this motivation, we propose access-ordered indexes in the next section.

Index Design

Based on our observations in the previous section, we propose organising postings within each postings list by decreasing access count. Our motivation is to store those postings that are most likely to appear in the top answers to queries at the beginning of each list and, similarly to the frequency- and impact-ordered approaches described in Section 2, to partially process postings lists. However, in contrast to previous approaches, our scheme does not rely on only the document collection and ranking function. Instead, it includes the properties of the queries — the temporal information that users provide — to further refine index design. A successful access-ordered index should permit accurate query evaluation compared to processing all postings, while being faster because less data is used in query evaluation.

Consider an example of an access-ordered index. Suppose that a collection contains five documents, and that five queries are available to determine access counts. Suppose also that we choose to increment access counts when a document appears in the top three answers in response to a query. After evaluating the first query, the following total ranking of documents is achieved: $[2, 1, 4, 3, 5]$. For this query, the access counts of documents 2, 1, and 4 are updated to 1. Suppose then that the four remaining queries achieve the following orderings: $[1, 2, 3, 4, 5]$, $[5, 3, 2, 1, 4]$, $[2, 4, 1, 3, 5]$, and $[1, 2, 3, 4, 5]$. At the conclusion, the access counts are $< 4, 1 >$, $< 5, 2 >$, $< 3, 3 >$, $< 2, 4 >$, and $< 1, 5 >$, where $< a_d, d >$ is the access count a for document d.

Consider now the term "richmond" that occurs in the collection of five documents. Suppose that it occurs in documents 1, 2, and 5, and so the conventional document-ordered postings (with arbitrary frequencies, and taking document differences) are:

$$< 1, 1 > < 2, 1 > < 1, 3 >$$

To reorganise this postings list by access frequency, we inspect the frequencies gathered in the training phase, and sort the list as follows:

$$< 5, 2, 2 > < 4, 1, 1 > < 1, 1, 5 >$$

Each posting tuple is expressed as a triple $< a_d, f_{d,t}, d >$. Note that the document identifiers are no longer stored as differences, since an absolute ordering of postings by document is no longer a guarantee; as we show later, this has a significant negative

impact on the compressibility of the lists. However, in practice, a_d is not stored in the posting since it need only be stored once per document and, therefore, can be compactly represented as an additional component of the in-memory mapping table.

To make use of the access-ordered index, a strategy is required to heuristically abandon processing of a postings list. Perhaps the simplest approach is to only process a fixed number of postings P from each list, with the motivation that short lists that may have a larger impact on the ranking function (because of smaller f_t values) will be processed in their entirety, and longer lists will be partially processed. We refer to this scheme as MAXPOST. For example, consider a threshold value of $P = 2$. For the single word query "richmond", only the postings for documents 2 and 1 would be processed, leading to the ranking 2 then 1 by the Okapi BM25 formulation. The computational saving is that the posting for document 5 is not decoded and processed, but the disadvantage of a fixed threshold is that document 5 can never be returned in response to the query; we return to this discussion in Section 4.

There are several other possible approaches to partial list processing. These include:

- MINACCESS: Processing only those postings with an access count greater than a threshold. In preliminary experiments, we found that this simple MINACCESS scheme is inefficient because of the cost of access count look-ups for each posting, and we did not pursue it further.
- AVGACCUM: Calculating an average accumulator contribution for the postings that have been processed in a list, and stopping when this falls below a threshold. We refer to this threshold as A. We report results with this AVGACCUM scheme in Section 4.
- AVGROLLACCUM: Extending AVGACCUM so that the average is computed over a moving window of D documents, with the aim of detecting more local change in accumulator contributions. We call this AVGROLLACCUM and report results in Section 4.

Persin et al. (1996) define a pruning scheme based on two possibilities. First, the processed posting is significant enough to justify a new accumulator if one does not exist; and second, the processed posting is not significant enough alone to justify a new accumulator, but where an accumulator exists it is updated. In their work, this comparison was performed by comparing document term frequency $f_{d,t}$ against two calculated threshold values. Due to the dependence on a correlation between index ordering and the similarity measure, this pruning approach is not directly applicable to our access-ordered index. However, the approach is similar to our MINACCESS scheme.

In their impact-ordered work, Anh and Moffat (2001, 2002) proposed several index pruning schemes that build on the QUIT and CONTINUE strategies of Moffat and Zobel (1996). They defined a BLOCK-FINE approach where the product of transformed document and query impacts are sequentially penalised until the accumulator contribution of the processed postings block is zero, at which point processing of the list is stopped. As the ordering of our access-ordered indexes is not directly related to the effect of a posting on the similarity function this approach cannot be applied to our index. However, the AVGACCUM and AVGROLLACCUM schemes allow for a similar granularity of pruning by terminating the processing of a list when the average contribution of preceding postings falls below a given threshold. Anh and Moffat also defined a variation of the BLOCK-FINE scheme, TERM-FINE, where a penalty is applied to each sequential term—as opposed to each block in a single list—therefore increasing the probability of pruning postings lists for terms that are processed later during the query evaluation process. We have yet to apply a penalty at a term level as in the TERM-FINE approach, and leave this open as an area of further work.

The MAXPOST scheme has a simple but clever extension. A document-ordered index can be *pruned* by removing any posting that is not in the first P postings ordered by access count. This can be achieved by converting a document-ordered list to access-order, removing postings after P, and restoring the list to document-order. Or, more sensibly, by heapifying the postings by access count using a heap of size P, sorting the resultant heap by document number, and writing out the list. The approach has the benefits of both access- and document-ordered indexes: the postings for documents that appear in highly-ranked documents are stored in the index, the compression effectiveness of a document-ordered index is maintained, and no additional processing is required at query time. We refer to this as an ACCESS-PRUNED index and we report results in the next section.

4 Results

We report our results in this section. We begin with a discussion of the queries, collection, and TREC experimental environment. We then present selected overall results. We conclude the section with a broader discussion of parameter choices.

Collection and Queries

The TREC consortium supports research in the area of information retrieval by providing an infrastructure for the evaluation of large-scale text retrieval methodologies[1]. TREC provides document collections with predefined query topics, and listings of the relevant documents for each topic. Our work makes use of the WT10g Web Track document collection (Hawking 2001). The WT10g collection consists of 1.69 million documents in 10 gigabytes of data taken from a web snapshot of 1997.

The query topics each consist of a unique numeric identifier, a query title, a description, and a narrative. Query titles are actual queries extracted from web query logs, while descriptions and narratives contain engineered values added to expand on the needs of the user. We make use of query topics 451-500 from TREC-9, utilising the title fields of each topic as they are the most similar to Web queries.

Along with query topics, the TREC consortium provides relevance judgements that list those documents that are deemed to be relevant for each query topic. The relevant documents are chosen from a pool of results that are returned by TREC participants. For topics 451-500 the mean pool size per topic was 1,401, with a mean of 52.34 documents judged to be relevant per topic. Although not all documents in the collection are tested for relevance, Zobel (1998) found that the pooling technique used by TREC is adequate for the domain in which it is used.

Access count data was gathered with queries taken from two Excite search engine query logs. The first log collected in 1997 contains just over 1 million queries, while the second log, collected in 1999 contains a further 1.7 million queries (Spink et al. 2001). Both logs have been filtered to remove queries with

[1] See http://trec.nist.gov/

Scheme	Threshold	Postings (%)	Lists (%)	Av. Prec.	R-Prec.	Query time (sec)
MAXPOST	$P = 12000$	0.1584	0.6955	0.1807	0.2172	0.089
MAXPOST	$P = 18000$	0.2138	0.7564	0.1906	0.2179	0.096
AVGACCUM	$A = 1.1$	0.6610	0.9655	0.1934	0.2241	0.168
AVGACCUM	$A = 1.2$	0.6550	0.9483	0.1865	0.2161	0.167
AVGROLLACCUM	$A = 2.0$, $D = 1000$	0.3679	0.8976	0.1848	0.2205	0.133
AVGROLLACCUM	$A = 2.0$, $D = 5000$	0.4345	0.9171	0.1867	0.2218	0.142
ACCESS-PRUNED	$P = 18000$	0.9998	0.9999	0.1903	0.2179	0.069
CONTINUE	—	0.9041	0.9607	0.1874	0.2245	0.124
QUIT	—	0.4402	0.8550	0.1823	0.2088	0.057

Table 1: *Overall accuracy and timing results for access-ordered and access-pruned indexes with different thresholding schemes. The Postings column shows the percentage of postings that are processed over the TREC queries, while the Lists column shows the average percentage of each list that was processed. The* CONTINUE *and* QUIT *schemes with a document-ordered index are shown for comparison.*

Threshold	Postings (%)	Lists (%)	Av. Prec.	R-Prec.	Query time (sec)
$P = 10000$	0.1380	0.6684	0.1794	0.2164	0.085
$P = 12000$	0.1584	0.6955	0.1807	0.2172	0.089
$P = 14000$	0.1777	0.7185	0.1863	0.2191	0.092
$P = 16000$	0.1961	0.7384	0.1889	0.2211	0.095
$P = 18000$	0.2138	0.7564	0.1906	0.2179	0.096
$P = 20000$	0.2307	0.7723	0.1914	0.2190	0.099
$P = 22000$	0.2470	0.7870	0.1920	0.2200	0.101

Table 2: *Overall accuracy and timing results for the* MAXPOST *scheme with 1.9 million training queries.*

terms that are ignored in the WT10g collection; ignored terms are mostly those deemed to be offensive.

Two common measurements of text retrieval *effectiveness* are *precision* and *recall* (Witten et al. 1999). Precision is a measure of the fraction of relevant documents retrieved compared to the total number of documents retrieved. Recall is the fraction of relevant documents that are retrieved to the total number of documents that are relevant for a query. *Average precision* is calculated as the average of the precision values at the eleven 10% recall points ranging from 0% to 100%. *R-Precision* is calculated as the precision after examining R results, where R is the number of relevant documents for the query. In this paper we use average precision and r-precision.

All experiments were conducted on Linux servers running Redhat 7.1 and kernel version 2.4.2-2smp. The machines consisted of dual Pentium III, 866 MHz processors with 256Mb of main-memory. Memory was flushed between each run to reduce caching effects, and the machines were under light-load during our timing experiments.

Overall Speed and Accuracy

Table 1 shows selected results of our experiments with access-ordered indexes. All schemes have a limit of 20,000 accumulators and a continue strategy is used in all schemes except QUIT. The access-ordered schemes are trained using all 1.9 million queries from the Excite log. The baseline document-ordered CONTINUE scheme has an average precision of 22.5% with an average of 0.124 seconds per query, and the QUIT strategy 18.2% with an average of 0.057 seconds (Moffat and Zobel recommend not using the QUIT scheme in practice (1996)).

The MAXPOST and ACCESS-PRUNED approaches are fast and accurate, providing the accuracy of CONTINUE but with speed closer to QUIT. For the same settings, MAXPOST and ACCESS-PRUNED process almost identical postings, yet ACCESS-PRUNED is faster because unused postings are omitted, reducing both disk seek and read costs; the schemes are not identical because the threshold results in arbitrary selection of postings with the same access count. By processing at most $P = 18000$ entries from each access-ordered postings list, average precision is slightly higher than CONTINUE, while R-precision is slightly lower. Importantly, compared to CONTINUE, MAXPOST querying is around 25% faster and, even more impressively, ACCESS-PRUNED is over 40% faster.

Both MAXPOST and ACCESS-PRUNED process a fixed number of postings per list. Arguably, this approach may be unacceptable for all applications, as it may prevent a document being returned as an answer to a difficult information need; this can occur when a posting is the Pth or later posting in the lists for all terms in the query. This can be addressed in two ways: first, the threshold P can be dynamically adjusted in MAXPOST depending on the query (perhaps by pre-computing a query quality measure (Cronen-Townsend, Zhou & Croft 2002)); and second, by using the access-pruned index when a query is highly statistically similar to the past queries used in the training process, and using a document-ordered or MAXPOST index otherwise. We have not investigated either approach in detail.

The columns labelled "Postings" and "Lists" in Table 1 report the amount of data processed under each scheme as a percentage. "Postings" is the percentage of all postings that could have been processed that are actually processed under the scheme. In contrast, "Lists" is the average percentage of each list that is processed. For the access-ordered schemes, "Postings" is much lower than "Lists", since a few long lists are abandoned early; this saves the processing of a significant number of postings, but has a lesser effect on the percentage of lists that are processed because many short lists are processed almost completely. For CONTINUE and ACCESS-PRUNED, the values are not 100% because we use an optimisation that abandons processing of a document-ordered list when 20,000 accumulators have been created and the document number in the list exceeds the highest document number in the accumulators.

The AVGACCUM and AVGROLLACCUM schemes are either accurate and slow, or fast and inaccurate compared to the MAXPOST scheme; however, in general, AVGROLLACCUM is the preferred accumulator-based scheme because the local window-based scheme is more sensitive to a drop in accumulator contributions, and therefore more effective in abandoning a

Queries	Postings (%)	Lists (%)	Av. Prec.	R-Prec.	Query time (sec)
20,000	0.1777	0.7099	0.1846	0.2191	0.090
200,000	0.1777	0.7099	0.1843	0.2183	0.092
500,000	0.1777	0.7185	0.1848	0.2166	0.096
1,979,077	0.1777	0.7185	0.1863	0.2191	0.092

Table 3: *Overall accuracy and timing results using different numbers of training queries and the* MAXPOST *scheme with* $P = 14000$.

Threshold	Av. Prec	R-Prec.	Query time (sec)	Index size (Gb)
$P = 8000$	0.1758	0.2071	0.051	1.19
$P = 10000$	0.1794	0.2164	0.056	1.26
$P = 12000$	0.1808	0.2172	0.061	1.31
$P = 14000$	0.1863	0.2191	0.064	1.36
$P = 16000$	0.1889	0.2211	0.067	1.40
$P = 18000$	0.1903	0.2179	0.069	1.44

Table 4: *Accuracy, timing, and index size results for* ACCESS-PRUNED *indexes.*

list. Both schemes are slower than MAXPOST for two reasons: first, but less importantly, an average has to be maintained; and, second, the schemes are less sensitive to list length as evidenced by the closer values in the "Postings" and "Lists" columns of Table 1. The result of the latter effect is that more postings than MAXPOST are processed in longer lists than are needed to achieve an accurate ranking, while short lists are also processed almost completely.

Index Sizes

The document-ordered index for the WT10g collection is 2.37 Gb or around 23% of the size of the collection. The access-ordered index for MAXPOST is 25% larger, requiring 2.97 Gb or around 29% of the collection size. This is the result of the loss of compressibility of removing document ordering, as discussed in Section 3. The access-pruned index for $P = 18000$ is compact: it is 1.44 Gb or almost 40% smaller than a complete document-ordered index and only 14% of the collection size. All indexes are compressed using variable-byte coding (Williams & Zobel 1999), a scheme that has been shown to be less effective in storage than well-known bitwise schemes (Witten et al. 1999) but much faster for query evaluation (Scholer et al. 2002).

Parameters

We experimented with a wide range of parameter settings in our exploration of access-ordered indexing. We present some of these parameters here, including different thresholds in MAXPOST, varying the number of training queries used to determine access counts, and different pruning thresholds for ACCESS-PRUNED. For compactness, we omit detailed listing of parameters for other schemes.

Table 2 shows the effect of varying the threshold for the MAXPOST scheme. For values of P less than 14,000, the scheme is fast but significantly less accurate than the CONTINUE approach. For values of P larger than 16,000, average precision is higher than CONTINUE, while R-precision is within 0.7%. The tradeoff is that query evaluation is only 7%–20% faster than the CONTINUE approach. Overall, we recommend processing between $P = 12000$ to $P = 16000$ postings per list.

Table 3 shows how query training set size affects the speed and accuracy of the MAXPOST scheme using the value $P = 14000$. We found similar trends with other values of P but do not present them here. The results show that access-ordering is extremely simple to establish: with only 20,000 queries—a tiny fraction of those posed to a popular search engine each hour—an ordering can be established that achieves almost indistinguishable accuracy from that of using 1.9 million queries. This is supported by other evidence that shows that the set of documents with high access counts does not change substantially as more queries are processed; we do not present detailed results here.

Our final results are presented in Table 4, which shows the size and accuracy of various settings for the ACCESS-PRUNED approach. The accuracy results reflect the same settings for the MAXPOST scheme, but the index sizes are markedly smaller than the full document-ordered index of 2.37 Gb. In addition, the query times are around half those of CONTINUE, with the average precision marginally higher for values of P equal to or greater than 16,000, and marginally lower for R-precision.

5 Conclusion

Web search is perhaps the most popular tool for information discovery. With staggering increases in query loads, numbers of users, and collection sizes, it is essential that innovative solutions continue to be developed to support query evaluation. In this paper, we have proposed a new, fast and accurate query evaluation approach based on a new organisation of inverted indexes.

Our novel access-ordered index is motivated by the observation that only a fraction of the documents in a collection are frequently returned in response to queries. We have proposed organising the index so that those documents are stored at the beginning of inverted lists, so that they can be processed early in the query evaluation process and the process abandoned when less popular documents are encountered. Our results show that this approach results in a time saving of between 25%–40% over a conventional, optimised approach while maintaining almost the same accuracy.

We will continue our investigation of access-ordered indexes with further investigation of the impact and performance gains possible considering both the average query and abnormal queries with the goal of identifying and resolving those conditions that do not directly benefit from access-ordering. We plan to investigate new compaction schemes with the aim of developing more space effective indexes and, as a result, even faster query evaluation. We also plan to compare our schemes to other state-of-the-art index organisation schemes such as the frequency- and impact-ordered schemes described in Section 2. Last,

a larger study is planned using a larger collection and query log to investigate the robustness of access-ordering over longer time periods, as temporal drift occurs in queries and documents, and as documents change.

Acknowledgements

This work is supported by the Australian Research Council and the State Government of Victoria.

References

Anh, V. & Moffat, A. (2001), Improved retrieval effectiveness through impact transformation, *in* X. Zhou, ed., 'Proc. Australasian Database Conference', Vol. 24(2), Australian Computer Society, Melbourne, Australia, pp. 41–48.

Anh, V. & Moffat, A. (2002), Impact transformation: Effective and efficient web retrieval, *in* K. Järvelin, M. Beaulieu, R. Baeza-Yates & S. H. Myaeng, eds, 'Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval', Tampere, Finland, pp. 3–10.

Baeza-Yates, R. & Ribeiro-Neto, B. (1999), *Modern Information Retrieval*, Addison-Wesley-Longman.

Breslau, L., Cao, P., Fan, L., Phillips, G. & Shenker, S. (1999), Web caching and zipf-like distributions: Evidence and implications, *in* 'INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies', Vol. 1, IEEE, New York, United States, pp. 126–134.

Cronen-Townsend, S., Zhou, Y. & Croft, W. (2002), Prediciting query performance, *in* K. Järvelin, M. Beaulieu, R. Baeza-Yates & S. H. Myaeng, eds, 'Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval', Tampere, Finland, pp. 299–306.

Hawking, D. (2001), Overview of the TREC-9 web track, *in* E. Voorhees & D. Harman, eds, 'The Ninth Text REtrieval Conference (TREC-9)', NIST Special Publication 500-249, Maryland, United States, pp. 87–103.

Moffat, A. & Zobel, J. (1996), 'Self-indexing inverted files for fast text retrieval', *ACM Transactions on Information Systems* **14**(4), 349–379.

Persin, M., Zobel, J. & Sacks-Davis, R. (1996), 'Filtered document retrieval with frequency-sorted indexes', *Journal of the American Society for Information Science* **47**(10), 749–764.

Robertson, S. & Walker, S. (2000), Okapi/Keenbow at TREC-8, *in* E. Voorhees & D. Harman, eds, 'The Eighth Text REtrieval Conference (TREC-8)', NIST Special Publication 500-246, Gaithersburg, MD, pp. 151–161.

Robertson, S., Walker, S., Hancock-Beaulieu, M., Gull, A. & Lau, M. (1992), Okapi at TREC, *in* D. Harman, ed., 'The First Text REtrieval Conference (TREC-1)', NIST Special Publication 500-207, Gaithersburg, MD, pp. 21–30.

Scholer, F., Williams, H., Yiannis, J. & Zobel, J. (2002), Compression of inverted indexes for fast query evaluation, *in* K. Järvelin, M. Beaulieu, R. Baeza-Yates & S. H. Myaeng, eds, 'Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval', Tampere, Finland, pp. 222–229.

Sparck-Jones, K., Walker, S. & Robertson, S. (2000), 'A probabilistic model of information retrieval: development and comparative experiments. Parts 1&2', *Information Processing and Management* **36**(6), 779–840.

Spink, A., Wolfram, D., Jansen, M. & Saracevic, T. (2001), 'Searching the web: The public and their queries', *Journal of the American Society for Information Science and Technology* **52**(3), 226–234.

Williams, H. & Zobel, J. (1999), 'Compressing integers for fast file access', *Computer Journal* **42**(3), 193–201.

Williams, H. & Zobel, J. (n.d.), 'Searchable words on the web', *International Journal of Digital Libraries* . To appear.

Witten, I., Moffat, A. & Bell, T. (1999), *Managing Gigabytes: Compressing and Indexing Documents and Images*, second edn, Morgan Kaufmann Publishers, Los Altos, CA 94022, USA.

Zobel, J. (1998), How reliable are the results of large-scale information retrieval experiments?, *in* R. Wilkinson, B. Croft, K. van Rijsbergen, A. Moffat & J. Zobel, eds, 'Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval', Melbourne, Australia, pp. 307–314.

Zobel, J., Heinz, S. & Williams, H. (2001), 'In-memory hash tables for accumulating text vocabularies', *Information Processing Letters* **80**(6), 271–277.

Zobel, J. & Moffat, A. (1998), 'Exploring the similarity space', *ACM SIGIR Forum* **32**(1), 18–34.