Efficient Phrase Querying with an Auxiliary Index

Dirk Bahle Hugh E. Williams Justin Zobel
School of Computer Science and Information Technology
RMIT University, GPO Box 2476V
Melbourne, Australia, 3001.
{dbahle,hugh,jz}@cs.rmit.edu.au

ABSTRACT

Search engines need to evaluate queries extremely fast, a challenging task given the vast quantities of data being indexed. A significant proportion of the queries posed to search engines involve phrases. In this paper we consider how phrase queries can be efficiently supported with low disk overheads. Previous research has shown that phrase queries can be rapidly evaluated using nextword indexes, but these indexes are twice as large as conventional inverted files. We propose a combination of nextword indexes with inverted files as a solution to this problem. Our experiments show that combined use of an auxiliary nextword index and a conventional inverted file allow evaluation of phrase queries in half the time required to evaluate such queries with an inverted file alone, and the space overhead is only 10% of the size of the inverted file. Further time savings are available with only slight increases in disk requirements.

General Terms

Indexing, query evaluation

Keywords

Inverted indexes, nextword indexes, evaluation efficiency, index size, stopping, phrase query

1. INTRODUCTION

Search engines are used to find data in response to ad hoc queries. On the Web, most queries consist of simple lists of words. However, a significant fraction of the queries include phrases, where the user has indicated that some of the query terms must be adjacent, typically by enclosing them in quotation marks. Phrases have the advantage of being unambigous concept markers and are therefore viewed as a valuable addition to ranked queries [6, 7].

In this paper, we explore new techniques for efficient evaluation of phrase queries.

A standard way to evaluate phrase queries is to use an inverted index, in which for each index term there is a list of postings, and each posting includes a document identifier, an in-document frequency, and a list of offsets. These offsets are the ordinal word positions at which the term occurs in the document. Given such a word-level inverted index and a phrase query, it is straightforward to combine the postings lists for the query terms to identify matching documents. This does not mean, however, that the process is fast. Even with an efficient representation of postings [16], the list for a common term can require several megabytes for each gigabyte of indexed text. Worse, heuristics such as frequency-ordering [13] or impact-ordering [1] are not of value, as the frequency of a word in a document does not determine its frequency of participation in a particular phrase.

A crude solution is to use stopping, as is done by some widely-used web search engines (the Google search engine, for example, neglects common words in queries), but this approach means that a small number of queries cannot be evaluated, while many more evaluate incorrectly [12]. Another solution is to index phrases directly, but the set of word pairs in a text collection is large and an index on such phrases difficult to manage.

In recent work, nextword indexes were proposed as a way of supporting phrase queries and phrase browsing [2, 3, 15]. In a nextword index, for each index term or *firstword* there is a list of the words or *nextwords* that follow that term, together with the documents and word positions at which the firstword and nextword occur as a pair. The disadvantage of a nextword index is its size, typically around half that of the indexed collection. Also, as described originally, nextword index processing is not particularly efficient, as the nextwords must be processed linearly and (compared to an standard inverted index) for rare firstwords the overhead of the additional layer of structure may outweigh the benefits.

In this paper we propose that phrase queries be evaluated through a combination of an inverted index on rare words and a form of nextword index on common words. We explore the properties of phrase queries and show experimentally that query evaluation time can be halved if just the three most common firstwords are supported through a nextword index. While phrase browsing is not possible with such an arrangement, the disk overheads of the partial nextword index are small and the benefits are substantial.

We have observed that many ordinary queries — those without quotation marks — nonetheless resolve successfully if processed as a phrase query, a phenomenon that search engine users are familiar with, as the most popular engines

highly rank matches in which the query terms are adjacent. This suggests that phrase querying is a potential method for a fast "first cut" evaluation method, as it allows more rapid identification of documents in which the terms occur as a phrase.

2. PROPERTIES OF QUERIES

With large web search engines being used daily by millions of users, it has become straightforward to gather large numbers of queries and see how users are choosing to express their information needs. Some search engine companies have made extracts of their query logs freely available. In our research, we have made extensive use of query logs provided by Excite dating to 1997 and 1999, as well as more recent logs from other sources. These logs have similar properties (with regard to our purposes), and we report primarily on the Excite logs in this work. In the Excite log, after sanitising to remove obscenity there are 1.583,922 queries (including duplicates). Of these, 132,276 or 8.3% are explicit phrase queries, that is, they include a sequence of two or more words enclosed in quotes. Amongst the rest of the queries—those without a phrase— about 5% contain a word that does not occur at all in the 21.9 gigabytes (Gb) of data we use. However, almost exactly 41% of the remaining non-phrase queries actually match a phrase in the 21.9 Gb dataset we use in our experiments. A surprising proportion of the phrases include a common term. Amongst the explicit phrase queries, 11,103 or 8.4% include one of the three words that are commonest in our dataset, "the", "to", and "of". 14.4% of the phrase queries include one of the 20 commonest terms. In some of these queries the common word has a structural function only, as in `tower of london`, and can arguably be safely neglected during query evaluation. In others, however, common words play an important role, as in the movie title `end of days` or the band name `the who`, and evaluation of these queries is difficult with the common words removed, especially when both "the" and "who" happen to be common terms [12].

Taken together, these observations suggest that stopping of common words will have an unpredictable effect. Stopping may yield efficiency gains, but means that a significant number of queries cannot be correctly evaluated. We experimented with a set of 122,438 phrase queries that between them match 309×10^6 documents. Stopping of common words means that a query such as `tower of london` must be evaluated as `tower -- london`: the query evaluation engine knows that the two remaining query terms must appear with a single term between them. If the commonest three words are stopped, there are 390×10^6 total matches for all queries extracted from the log. However, these are distributed extremely unevenly amongst the queries: for some queries the great majority of matches are incorrect. The figure rises to 490×10^6 for the commonest 20 words, and 1693×10^6 for the commonest 254 words, while a significant number of queries, containing only stopped words, cannot be evaluated at all.

Amongst the phrase queries, the median number of words in a phrase is 2, and the average is almost 2.5. About 34% of the queries have three words or more, and 1.3% have six words or more. A few queries are much longer, such as titles: `the architect of desire beauty and danger in the stanford white family by suzannah lessard`.

Another point of interest is where in a phrase the common words occur. In English, the common words rarely terminate a phrase query. Only 0.4% of phrase queries with "the", "to", or "of" have these words at the end. Almost all of these queries are short: virtually no queries of four words or more terminate with one of the commonest terms. In the short queries ending in a common term, the other query terms are themselves usually common. We take advantage of these trends in the methods for phrase query evaluation proposed in this paper.

3. INVERTED INDEXES

Inverted indexes are the standard method for supporting queries on large text databases; there are no practical alternatives to inverted indexes that provide sufficiently fast ranked query evaluation. An inverted index is a two-level structure. The upper level is a data structure such as a B-tree containing all the index terms for the collection. For text databases, the index terms are usually the words occurring in the text, and all words are included. The lower level is a set of postings lists, one per index term. Following the notation of Zobel and Moffat [17], each posting is a triple of the form:

$$\langle d, f_{d,t}, [o_1, \ldots, o_{f_{d,t}}] \rangle$$

where d is the identifier of a document containing term t, the frequency of t in d is $f_{d,t}$, and the o values are the positions in d at which t is observed. An example inverted file is shown in Figure 1. In this example, there is a vocabulary of five words, each of which has a postings list.

It is straightforward to use an inverted index to evaluate a phrase query. Consider the query `magdalene sue prentiss`. Of these terms, "magdalene" is the rarest, and its inverted list is fetched first. The postings are decoded and a temporary structure is created, recording which documents contain this word and the ordinal word positions in each document at which it occurs. The term "prentiss" is the next rarest, and is processed next. For each document identifier and word offset in the temporary structure created earlier, a posting is sought to see whether "prentiss" is in the document two words later. If the search fails, that word position is discarded from the temporary structure, as is the document identifier if no word positions for that document remain. As both the structure and the postings are sorted, this process is a linear merge. Then the postings list for "sue" is fetched and decoded, and used to further delete entries from the temporary structure. The remaining entries are documents and word positions at which the phrase occurs. Similar approaches have been described elsewhere [5, 10].

Summarising, phrase queries are evaluated as follows.

1. Sort the query terms from rarest to commonest, keeping note of their original position in the phrase.
2. Fetch the postings list for the first (rarest) query term. Decode this list into a temporary structure of document identifiers and word offset positions.
3. For each remaining query term, decode its postings list, merging it with the temporary data; this merge process discards from the temporary structure all document identifiers and word offsets that do not match any entry in the postings list.

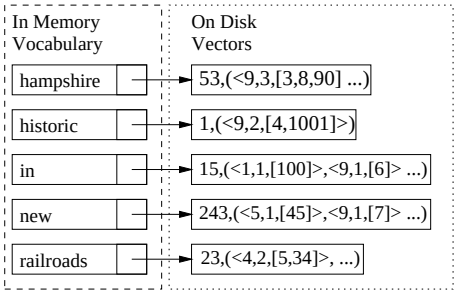

Figure 1: An inverted file for a collection with a vocabulary of five words.

In this query evaluation model, processing of the first query term establishes a superset of the possible locations of the complete phrase, which are maintained in a temporary structure; as the subsequent query terms are evaluated, this structure is pruned, never added to. It is thus essential to begin processing with the rarest query term, to avoid creation of an excessively large temporary structure (or of having to process the inverted lists in stages to stay within a memory limit).

A simple heuristic to address this problem is to directly merge the inverted lists rather than decode them in turn. On the one hand, merging has the disadvantage that techniques such as skipping [11] cannot be as easily used to reduce processing costs (although as we discuss later skipping does not necessarily yield significant benefits). On the other hand, merging of at least some of the inverted lists is probably the only viable option when all the query terms are moderately common.

Whether the lists are merged or processed in turn, the whole of each list needs to be fetched (unless query processing terminates early due to lack of matches). For ranked query processing it is possible to predict which postings in each inverted list are most likely to be of value, and move these to the front of the inverted list; techniques for such list modification include frequency-ordering [13] and impact-ordering [1]. With these techniques, only the first of the inverted lists need be fetched during evaluation of most queries, greatly reducing costs.

In contrast, for phrase querying it is not simple to predict which occurrences of the term will be in a query phrase, and thus such reordering is unlikely to be effective. Offsets only have to be decoded when there is a document match, but they still have to be retrieved.

Other techniques do have the potential to reduce query evaluation time, in particular *skipping* [11], in which additional information is placed in inverted lists to reduce the decoding required in regions in the list that cannot contain postings that will match documents that have been identified as potential matches. On older machines, on which CPU cycles were relatively scarce, skipping could yield substantial gains. On current machines, however, disk access costs are the more important factor, and in other experiments we have observed that the increase in length of lists required by skipping outweighs the reduction in decoding time. We therefore do not use skipping in our experiments.

We have implemented a phrase query evaluator based on inverted lists, using compression techniques similar to those employed in MG [16] to reduce costs, and have used it to test the efficiency of phrase query evaluation. Our test data is 21.9 Gb of HTML containing about 8.3 Gb of text (drawn from the TREC large web track [9]).

Table 1: Size of inverted index (Mb) after stopping of common words.

Number of words stopped	Index size (Mb)
0	2350
3	2259
6	2195
10	2135
20	2089
254	1708

Table 2: Times for phrase query evaluation (seconds) on an inverted index after stopping of common words. Results are shown for all queries; 2-word queries only; and 5-word queries only.

Number of words stopped	Overall time (sec)	2-word queries	5-word queries
0	1.56	0.49	6.41
3	0.66	0.30	1.94
6	0.45	0.29	1.07
10	0.40	0.28	0.81
20	0.37	0.28	0.70
254	0.18	0.16	0.26

Table 1 shows the size of the index with a range of levels of stopping. As can be seen, the three commonest words account for around 4% of the index size, and only small space savings are yielded by stopping. However, as Table 2 shows, the impact of stopping on query evaluation time is dramatic. Just removing the three commonest words reduces average time by about 60%, and by a factor of 3 for longer queries. For these longer queries, the savings continue to increase as more common words are stopped. It is the scale of these savings that make stopping attractive, despite the fact that they are at the cost of inaccurate query results.

4. NEXTWORD INDEXES

Inverted indexes allow evaluation of phrase queries, but faster evaluation is possible with phrase-oriented indexes. One possibility is to use a conventional inverted index in

which the terms are word pairs. Another way to support phrase based query modes is to index and store phrases directly [8] or simply by using an inverted index and approximating phrases through a ranked query technique [5, 10]. Greater efficiency, with no additional in-memory space overheads, is possible with a special-purpose structure, the nextword index [15], where search structures are used to accelerate processing of word pairs. The nextword index takes the middle ground by indexing pairs of words and, therefore, is particularly good at resolving phrase queries containing two or more words. As noted above and observed elsewhere, the commonest number of words in a phrase is two [14].

A nextword index is a three-level structure. The highest level is of the distinct index terms in the collection, which we call *firstwords*. At the middle level, for each firstword there is a data structure (such as a front-coded list, or for fast access a structure such as a tree) of *nextwords*, which are the words observed to follow that firstword in the indexed text. For example, for the firstword "artificial", nextwords include "intelligence", "insemination", and "hip". At the lowest level, for each nextword there is a postings list of the positions at which that firstword-nextword pair occur.

An example nextword index is shown in Figure 2. In this example, there are two firstwords, "in" and "new". Some of the nextwords for "in" are "all", "new", and "the". For each firstword-nextword pair, there is a postings list. (A nextword index is of course a form of inverted index, but for consistency with other work we use "inverted index" solely to refer to a standard word-level inverted file.)

In nextword indexes, the postings lists are typically short, because most pairs only occur infrequently. For example, the postings list for the firstword-nextword pair "the"·"who" is orders of magnitude smaller than the postings lists for these words in an inverted file. It follows that phrase query evaluation can be extremely fast.

Nextword indexes also have the benefit of allowing phrase browsing or phrase querying [4, 15]; given a sequence of words, the index can be used to identify which words follow the sequence, thus providing an alternative mechanism for searching text collections. We do not consider phrase browsing further in this paper, however.

For phrase queries of more than two words, multiple postings lists must be fetched from the nextword index to resolve the query. Selection of which listings to fetch requires a little care. For example, with the query

boulder municipal employees credit union

the query can be resolved by fetching the postings lists for the firstword-nextword pairs "boulder"·"municipal", "employees"·"credit", and "credit"·"union". Alternatively, it would be possible to get the lists for "boulder"·"municipal", "municipal"·"employees", and "credit"·"union". Which is most efficient depends on which is shorter: the list for "employees"·"credit" or the list for for "municipal"·"employees".

Unfortunately, establishing which is shorter requires two disk accesses, to retrieve the nextwords for "employees" and "municipal". However, we have observed that the frequency of a firstword closely correlates to the lengths of its nextword lists. Thus in the query

historic railroads in new hampshire

we can with confidence choose "railroads"·"in" in preference to "in"·"new", because "railroads" is much less common than "in". Algorithms for choosing order of evaluation are considered by Bahle, Williams, and Zobel [3]. An efficient algorithm for evaluating phrase queries with a nextword index is as follows.

1. If the number of query terms n is even, the query can consist of $n/2$ disjoint firstword-nextword pairs. If the number of query terms n is odd, $\lceil n/2 \rceil$ firstword-nextword pairs must be chosen. However, in both cases it is more efficient to choose more than the minimum number of pairs, if doing so avoids choice of a common word as a firstword.
2. The method we use is to choose all $n-1$ firstword-nextword pairs; then sort them by increasing firstword frequency; then discard from the list the pairs that are completely covered by preceding selections. This approach can lead to processing of more than $\lceil n/2 \rceil$ pairs, but experimentally was shown to reduces costs overall.
3. The selected word pairs are sorted by increasing frequency of the firstwords, then their postings lists are processed as for inverted file phrase query processing.

The size of a nextword index is 4487 Mb, almost exactly twice that of an inverted file. For phrase queries, the savings in query evaluation time are dramatic. Average query evaluation time is reduced to 0.06 seconds, faster than inverted files by a factor of 25. For two-word queries, the time falls to 0.01 seconds, which is faster by a factor of 50. The time for 5-word queries is 0.32.

An interesting possibility suggested by these results is that—given space for a nextword index—all queries be evaluated as if they were phrases. We observed above that a significant fraction of all queries successfully evaluate, and indeed on browsing the query logs it is obvious that many of the queries without quotation marks are nonetheless intended to be phrases. Spink et al. [14] suggest that most two word queries should be treated as a phrase query even if they were entered as a ranked query. Given that search engines return as highest matches the pages in which the query words appear in sequence, use of a nextword index provides a rapid mechanism for finding these pages.

Much of the speed improvement for phrase queries yielded by nextword indexes is for queries involving a non-rare word. Indeed, for queries of rare words there may be little gain, as query processing with nextword indexes involves more complex structures than does processing with inverted indexes. As the two approaches to phrase query processing appear, then, to have complementary advantages, it is attractive to try to combine their strengths.

5. COMBINED QUERY EVALUATION

We have observed above that inverted indexes are the least efficient for phrases involving common words, the case where nextword indexes yield the greatest speed advantage. We therefore propose that common words only be used as firstwords in a stripped-down nextword index, and that this new index be used where possible in evaluation of phrase queries. We call this a *top frequency* based scheme, since only the most frequent words are indexed in the nextword index. We have explored other schemes based on the frequency of words in the indexed collection, or based on the

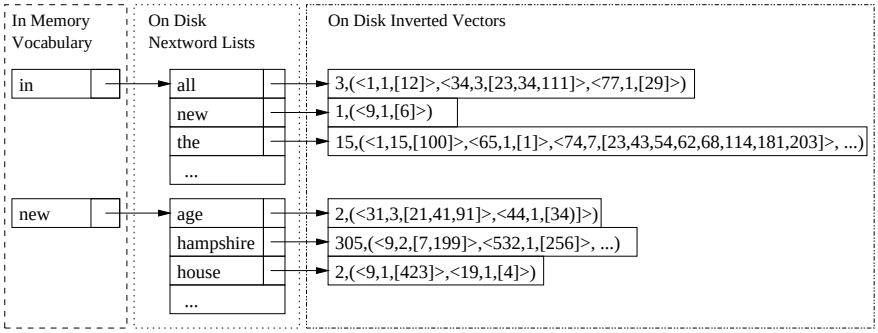

Figure 2: A nextword index with two firstwords.

frequency of words in the query log. None of the investigated schemes offered a better space and time trade-off, so we report only results from the top frequency scheme.

An example of a top frequency combined index is shown in Figure 3. At the left there is a vocabulary of five words. Each word has an inverted list, together constituting a complete inverted file for these words. In addition, the common words "in" and "new" have a nextword index.

With a combined index, processing involves postings lists from both the inverted index and the nextword index. Consider again the query:

historic railroads in new hampshire

Neither "historic" nor "railroads" is a common word, so establishing that these terms occur in the phrase involves fetching their postings lists from the inverted index and processing in the usual way. However, "in" and "new" are both common. The posting list for the firstword-nextword pair "in"·"new" from the nextword index must be fetched and processed. Then there is a choice. On the one hand, the nextword index postings list for "new"·"hampshire" cannot be longer than the inverted index postings list for "hampshire" and in all likelihood is a great deal shorter. On the other hand, compared to the inverted index, an extra disk access is required to fetch a postings list from the nextword index. In our implementation, we process using the nextword index if possible, and resort to the inverted index only for terms that are not in an indexed firstword-nextword pair.

In summary, we use the following process:

1. Identify all pairs in the list in which the first term is an indexed firstword. Sort these terms, and prune the list as for standard evalution of phrase queries via a nextword index.
2. For all terms not in a firstword-nextword pair, sort.
3. Process the postings lists in increasing order of firstword frequency, so that processing of nextword index lists and of inverted file lists is interleaved.

In this model, a common word need only be evaluated via its postings list in the inverted file if it occurs as the last word in a query, which in the Excite query log is a rare event.

We have tested other query resolution methods that involved term sorting based on nextword frequency (or NWF, the number of nextwords for a firstword), inverted document frequency (or IDF, the number of documents in which a word occurs), or both. We also experimented with resolving nextword entries of a given query always first, or always last. We found overall that these different resolution methods did not significantly vary in query speed and behaved almost identically to sorting by IDF only. This is why we propose sorting inverted index terms and nextword terms based on IDF only: we do not need to keep another statistical value per index term and sorting is straightforward.

Table 3: Size of nextword index (Mb) containing only common firstwords.

Number of common words	Index size (Mb)
3	254
6	427
10	520
20	657
254	1366

In Table 3 we show sizes of nextword indexes in which only the commonest terms are allowed as firstwords. For a nextword index on the three commonest terms only, for example, it can be seen that the space consumed is just over 10% of the size of the inverted index or around 1% of the size of the original HTML collection.

Query evaluation time with a combined index is shown in Table 4. (The "0" line is repeated from Table 2.) In contrast to earlier tables of evaluation time in this paper, in every line of this table all queries are correctly resolved. As can be seen, use of a nextword index allows evaluation of all phrase queries, and much more rapidly than was previously possible. Use of a partial nextword index of 1% of the HTML collection halves query evaluation time; a partial nextword index of less than 3% of the size of the collection cuts time to a third. The time savings with more aggressive stopping are less significant, but the additional disk costs with more stopping are small.

These are substantial savings at low cost. Phrase query processing time with a nextword index is only slightly greater than with a stopped inverted file, and no answers are lost.

Such combined processing can be integrated with other heuristics for phrase query evaluation. For example, a strategy that is likely to be successful in the context of a web search engine is to maintain indexes (perhaps for a limited time only) on phrases, or word pairs from phrases,

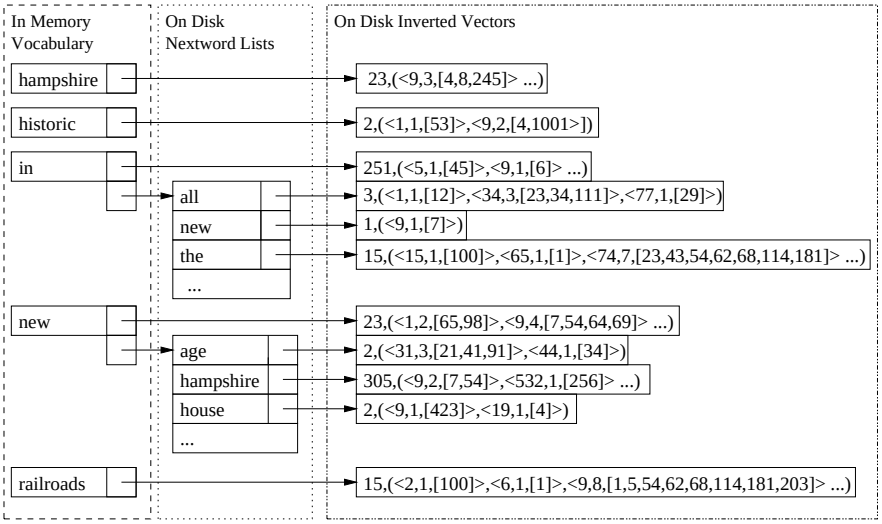

Figure 3: A combined inverted file and nextword index.

Table 4: Times for phrase query evaluation (seconds) on a combined index, with different numbers of common words used in the nextword index. Results are shown for all queries; 2-word queries only; and 5-word queries only.

Number of common words	Overall time (sec)	2-word queries	5-word queries
0	1.56	0.49	6.41
3	0.76	0.31	2.99
6	0.57	0.31	2.28
10	0.53	0.30	2.10
20	0.50	0.30	1.98
254	0.46	0.27	1.83

that are commonly posed as queries. Amongst our 132,276 queries, 72,184 are distinct. The commonest phrase query (`thumbnail post`) occurs 205 times and involves no common words. The queries themselves contain 92,846 distinct word pairs; the commonest pair occurs 683 times. Indexing of common query pairs has the potential to yield significant further savings.

6. CONCLUSIONS

We have proposed that phrase queries on large text collections be supported by use of a small auxiliary index. In this approach, all words in the text are indexed via an inverted file; in addition, the commonest words are indexed via an auxiliary nextword index, which stores postings lists for firstword-nextword pairs. We have shown that the cost of evaluating phrase indexes can be cut by a factor of three, with an auxiliary index that is only 3% of the size of the indexed data.

These results show that there is no need to use stopping in phrases. Indeed, the statistics on the number of matches indicate that such stopping leads to significant error rates. While it can be argued that mistakes in matching due to stopping of common words will often be unimportant, we have demonstrated that there is no reason to make such mistakes at all.

Our schemes have scope for improvement. In particular, choosing of pairs during query evaluation requires further exploration, and we are further investigating structures for representing nextword lists. However, our results show that evaluation of phrase queries can be dramatically accelerated with only a small additional index, and that stopping of phrases leads to errors and is not necessary for efficiency.

7. REFERENCES

[1] V. N. Anh, O. Kretser, and A. Moffat. Vector-Space ranking with effective early termination. In W. B. Croft, D. J. Harper, D. H. Kraft, and J. Zobel, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 35–42, New York, Sept. 9–13 2001. ACM Press.

[2] D. Bahle, H. Williams, and J. Zobel. Compaction techniques for nextword indexes. In *Proc. 8th International Symposium on String Processing and Information Retrieval (SPIRE2001)*, pages 33–45, San Rafael, Chile, 2001.

[3] D. Bahle, H. E. Williams, and J. Zobel. Optimised phrase querying and browsing in text databases. In M. Oudshoorn, editor, *Proc. Australasian Computer Science Conference*, pages 11–19, Gold Coast, Australia, Jan. 2001.

[4] P. Bruza, R. McArthur, and S. Dennis. Interactive internet search: keyword, directory and query reformulation mechanisms compared. In N. J. Belkin, P. Ingwersen, and M.-K. Leong, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 280–287, Athens, 2000.

[5] C. L. Clarke, G. V. Cormack, and E. A. Tudhope. Relevance ranking for one- to three-term queries. In *Proc. of RIAO-97, 5th International Conference "Recherche d'Information Assistee par Ordinateur"*, pages 388–400, Montreal, CA, 1997.

[6] W. B. Croft, H. R. Turtle, and D. D. Lewis. The use

of phrases and structured queries in information retrieval. In A. Bookstein, Y. Chiaramella, G. Salton, and V. V. Raghavan, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 32–45, Chicago, 1991. ACM.

[7] E. F. de Lima and J. O. Pedersen. Phrase recognition and expansion for short, precision-biased queries based on a query log. In *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 145–152, Berkeley, 1999. ACM Press.

[8] C. Gutwin, G. Paynter, I. Witten, C. NevillManning, and E. Frank. Improving browsing in digital libraries with keyphrase indexes. *Decision Support Systems*, 27(1/2):81–104, 1998.

[9] D. Hawking, N. Craswell, P. Thistlewaite, and D. Harman. Results and challenges in web search evaluation. In *Proc. of the Eighth International World-Wide Web Conference*, volume 31, pages 1321–1330, May 1999.

[10] D. D. Lewis and W. B. Croft. Term clustering of syntactic phrases. In J.-L. Vidick, editor, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 385–404. ACM, 1990.

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

[12] G. W. Paynter, I. H. Witten, S. J. Cunningham, and G. Buchanan. Scalable browsing for large collections: A case study. In *Proc. of the 5th ACM International Conference on Digital Libraries*, pages 215–223, San Antonio, 2000.

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

[14] A. Spink, D. Wolfram, B. J. Jansen, and T. Saracevic. Searching the web: The public and their queries. *Journal of the American Society for Information Science*, 52(3):226–234, 2001.

[15] H. Williams, J. Zobel, and P. Anderson. What's next? index structures for efficient phrase querying. In M. Orlowska, editor, *Proc. Australasian Database Conference*, pages 141–152, Auckland, New Zealand, Jan. 1999.

[16] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, California, second edition, 1999.

[17] J. Zobel and A. Moffat. Exploring the similarity space. *SIGIR Forum*, 32(1):18–34, Spring 1998.