Fast Phrase Querying with Combined Indexes

HUGH E. WILLIAMS, JUSTIN ZOBEL, and DIRK BAHLE
RMIT University

Search engines need to evaluate queries extremely fast, a challenging task given the 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. Our 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. Alternatively, special-purpose phrase indexes can be used, but it is not feasible to index all phrases. We propose combinations of nextword indexes and phrase indexes with inverted files as a solution to this problem. Our experiments show that combined use of a partial nextword, partial phrase, and conventional inverted index allows evaluation of phrase queries in a quarter the time required to evaluate such queries with an inverted file alone; the additional space overhead is only 26% of the size of the inverted file.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search Process*; H.3.4 [**Information Storage and Retrieval**]: Information Systems—*Performance Evaluation (efficiency and effectiveness)*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Nextword indexes, Phrase queries, Query evaluation, Web search

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 ordered and adjacent, typically by enclosing them in quotation marks. Phrases have the advantage of being unambiguous concept markers and are therefore viewed as a valuable retrieval technique [Clarke et al. 1997; Croft et al. 1991; de Lima and Pedersen 1999; Lewis and Croft 1990]. Moreover, phrase querying is described as being intuitive and is used with little error [Spink and Xu 2000].

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 [Witten et al. 1999], the list for a common term can require several megabytes for each gigabyte of indexed text. Worse, heuristics such as frequency-ordering [Persin et al. 1996] or impact-ordering [Anh et al. 2001] 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 was done by some widely-used web search engines until recently — the Google search engine, for example, neglected common words in phrase queries until 2002 — but this approach means that a small number of queries cannot be evaluated, while many more evaluate incorrectly [Paynter et al. 2000].

Another solution is to index phrases directly. A phrase query such as "albert park lake" can be answered extremely fast with this index, because answer documents can be determined directly from the postings of that phrase. However, complete phrase indexes that cover all phrases in a given text are likely to be prohibitive in size and construction time, while storing only selected phrases creates the problem of how to effectively identify useful phrases in the absence of queries.

In recent work, we proposed nextword indexes as a way of supporting phrase queries and phrase browsing [Bahle et al. 2001a; 2001b; Williams et al. 1999]. 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, our original nextword index was inefficient because 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 for rare words, a form of nextword index for the commonest words, and a phrase index for the commonest phrases. 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. We also show that directly indexing the 10,000 most common phrase queries reduces query evaluation time by over 10%. The disk overheads of these approaches are small and the benefits are substantial. The main cost is of maintaining the additional index structures. For example, on document insertion the number of inverted lists that must be modified is increased by perhaps 10%–20%. The benefits are even greater when the two approaches are combined: the complementary use of direct phrase indexing and a partial nextword index reduces query evaluation to only a quarter of the time taken with an inverted index.

We have observed that around 40% of ordinary ranked queries — those without quotation marks — resolve successfully if processed as a phrase query. This is unsurprising as, for example, it is rare than users put peoples' names in quotations when searching for home pages. It is also perhaps encouraged by the behaviour of

popular engines that highly rank matches in which the query terms are adjacent. This suggests that our phrase querying approach 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.

This paper is structured as follows. Section 2 overviews the properties of phrase queries, investigates the impact of stopping, and describes the test data we use in our experiments. Section 3 explains how inverted indexes are used in phrase query evaluation, and presents results using an efficient inverted index. We describe the special-purpose partial phrase and nextword indexes in Section 4, and show results of using a full nextword index for phrase querying. In Section 5, we present our novel ideas for combining phrase and inverted indexes for fast, space-effective query evaluation. Our results are presented in Section 6.

2. PROPERTIES OF QUERIES

With large web search engines being used daily by many 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 from 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 sanitizing 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. This is consistent with other studies of actual queries, which show that about 5%–10% are explicit phrase queries [Spink et al. 2001; Jansen and Pooch 2001]. Interestingly, we also found that almost exactly 41% of the remaining non-phrase queries actually match a phrase in an around 20 gigabyte web dataset.

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". In total, 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 other queries, 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 [Paynter et al. 2000]. Other all-stopword queries in the query logs include `to be or not to be`, `who are we`, and `all in all`.

Taken together, these observations suggest that *stopping* or ignoring 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.

It can be argued that stopwords are often insignificant, and that even a document that is technically a mismatch — due to the wrong stopword being present — may be just as likely to be relevant as a document where the match is correct. However, it is worth emphasising that there are many queries in which stopwords do play an important role. The words "to" and "from" are often stopped, for example, but mismatches to the query `flights to london` are likely to be incorrect. Another instance is that the word "the" often forms part of a description, thus `the moon` should not match websites about a moon of Jupiter, Keith Moon, or a book publisher. Examples in the weblogs of queries that mismatch due to stopping include `so many roads` and `how many roads`, and `man in the moon` and `man on the moon`.

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 containing the words "the", "to", or "of" have them 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.

2.1 Test data

We use the WT10g collection in our experiments. This collection is 10.27 Gb in size and contains around 1.67 million documents. The collection was created from a web crawl in 1997 and was motivated by the need for a realistic web collection that can be used to compare different retrieval methods in a constant environment [Bailey et al. 2003; Soboroff 2002]. The collection was primarily used at the TREC-9 and TREC 2001 conferences [Harman 1995; Voorhees and Harman 2001].

Table I gives an overview of the most frequent words and word-pairs in WT10g. The table shows that common words participate in forming common word-pairs with the exception of the word-pair `home page`, which is composed of two words that are both not amongst the twenty commonest words. Note that most frequent word-pairs occur an order of a magnitude less often than most frequent words.

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 are fast for ranked query evaluation. An inverted index is a two-level structure. The upper level is a *vocabulary* or *lexicon* of all the index terms for the collection. For text

Table I. *Frequently occurring words and word-pairs in WT10g.*

Word	Document count	Word-pair	Document count
the	1, 442, 962	of the	875, 373
to	1, 409, 600	to the	699, 322
of	1, 359, 221	with the	369, 047
and	1, 320, 093	home page	316, 342
for	1, 188, 776	in a	311, 550
in	1, 185, 007	will be	279, 379
on	1, 046, 783	with a	260, 492
is	998, 037	to a	225, 361
by	953, 516	that the	221, 836
this	896, 804	the first	204, 629
with	869, 809	you can	197, 432
or	827, 315	you have	161, 783
at	814, 872	to make	145, 685
all	799, 295	want to	144, 270
from	794, 283	well as	138, 491
are	791, 523	you are	137, 504
be	736, 281	which is	107, 897
that	724, 923	we are	106, 644
as	718, 672	we have	101, 024
an	684, 139	you to	85, 818
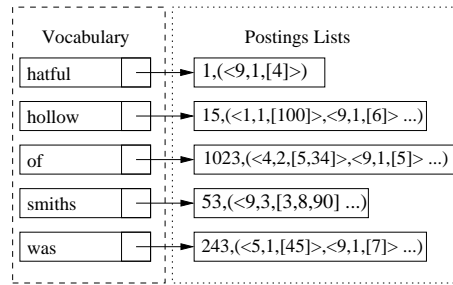
Fig. 1. *An inverted index for a collection with a vocabulary of five words.*

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 [1998], 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 index is shown in Figure 1. In this example the vocabulary has 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 `hatful of hollow` and the index in Figure 1. Of the query terms, "hatful" is the rarest (it occurs once in one document, document 9 at word position 4), and its inverted list is fetched first. The postings are decoded and a temporary

structure is created, recording the document that contains this word and the ordinal word position at which it occurs. The term "hollow" 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 "hollow" is in document 9 but 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. The result of this process is that document 9 at position 4 is the only location of the occurrence of `hatful` – `hollow`. As both the structure and the postings are sorted, this process is a linear merge. Then the postings list for "of" is fetched and decoded, and used to further delete entries from the temporary structure or resolve the query. The remaining entries are documents and word positions at which the phrase occurs, in this case document 9 at position 4.

This *sorted* phrase query algorithm for a query $t_1 \ldots t_n$ of $n > 0$ words can be stated as follows:

(1) Sort the n query terms t by their collection document frequency $d_{f,t}$ such that $t_i < t_{i+1}$.

(2) Fetch and decode postings for t_1 into a set of (d, o) tuples $\mathcal{L}$, where the values o are the offsets at in which t_1 appears in d. To simplify merging, subtract from each value o the position of t_1 in the original phrase.

(3) For each remaining query term $t_i = t_2, \ldots, t_n$, fetch the postings for t_i and remove from $\mathcal{L}$ any postings for which there is no corresponding posting in t_i. That is, each t_i has a set of (d, o) tuples, where the offsets have been modified by subtracting from each value o the position of t_i in the original phrase. The new $\mathcal{L}$ is the intersection of the original $\mathcal{L}$ with the set of tuples derived from the postings of t_i.

The initial set of (d, o) postings in $\mathcal{L}$ is a superset of the set of answer documents. The above algorithm has a lower bound of n fetching operations for posting lists and requires $n - 1$ merging steps. This bound must be implemented to arrive at an accurate result. Resolving, for example, only postings for the words "hatful" and "hollow" — for the album name `hatful of hollow` — could result in answer documents that may contain a phrase such as "hatful from hollow", but not the original phrase. We can thus exit the algorithm when either $\mathcal{L}$ is empty or postings for all words have been processed.

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 but never added to. It is therefore 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). The rarity of a query term is approximated by $d_{f,t}$ and this is typically used to derive a query evaluation plan; document frequency is an approximation because it is the number of documents containing the term, and not the total term occurrences in the collection.

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 [Moffat and Zobel 1996] cannot easily be 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 [Persin et al. 1996] and impact-ordering [Anh et al. 2001]. 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. Perhaps the only obvious optimisation is that, once a superset of possible matches is established, a list can be abandoned if a posting is for a document that has an ordinal number greater than the largest in the temporary structure; however, this relies on the postings being document-ordered and we use this optimisation in our experiments.

Other techniques also have the potential to reduce query evaluation time, in particular skipping [Moffat and Zobel 1996], 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.

As discussed previously, another additional strategy that could be efficient is stopping. Despite the false matches, there is significant potential for efficiency gains. We have implemented a phrase query evaluator based on inverted lists, using compression techniques similar to those employed in the open source MG text retrieval engine and described by Witten et al. [1999]. We used this implementation to test the efficiency of phrase query evaluation using the Excite query log and WT10g. The initial inverted index size for WT10g is around $1,429$ Mb and stopping results in avoiding processing posting lists that are around 427 Mb in total size. A stopped inverted index thus requires $1,002$ Mb.

Average retrieval times sorted by query length in words are shown in Table II. The row labeled "All" shows the average time for all queries, while the remaining rows are concerned with average retrieval times for two- to seven-word queries. The second column shows average retrieval times for the evaluation of all query words and the third column shows timings for phrase query evaluation with a carefully constructed list of of 490 stopwords.

Our results show that an average query of more than two words without stopping requires more than one second to evaluate. In contrast, stopped queries are on average 5 (for two words) to 17 (for seven words) times faster. However, this

Table II. *Average times for phrase querying with an inverted index using the 132,276 phrase entries in the Excite query log resolved on WT10g.*

Phrase	Retrieval time (sec)	
length	Full index	Stopped index
All	1.04	0.20
2	0.35	0.17
3	1.62	0.26
4	3.73	0.26
5	3.80	0.26
6	4.67	0.30
7	4.99	0.29

improvement comes at the expense of accuracy: querying without stopping results in a total of around 107 million answer documents, while stopping results in about 733 million answer documents. It is therefore unclear whether the tradeoff between efficiency and effectiveness favours stopping.

4. PHRASE INDEXES

A *phrase index* is an inverted index where the items stored in the vocabulary are word sequences rather than individual words. A *partial phrase index* stores information about selected phrases, and can thus only be used to answer queries on these phrases efficiently. Complete phrase indexes are likely to be prohibitive in construction time and space requirements, while partial indexes are only effective if frequently-queried phrases have been indexed.

Partial phrase indexes can be categorized further into phrase indexes that keep track of all phrases with a certain length or phrase indexes that keep track of arbitrarily long selective phrases. Again, a partial index with selective phrases of any length is effective if many future queries can be reliably predicted. The same argument applies to partial phrase indexes for phrases of a length l, because a phrase index with $l = 3$ cannot be used efficiently to answer two-word queries, but is effective and efficient if a high percentage of all queries have three words or more.

We experiment with two types of partial phrase index in this paper:

—*Partial phrase indexes*, where selected phrase queries of lengths $l \geq 2$ are stored as terms in a conventional inverted index structure.

—*Partial nextword indexes*, a structure that is organised for efficient evaluation of phrases of length $l = 2$.

We describe these two classes of phrase index in detail in this section.

4.1 Partial phrase indexes

A partial phrase index stores phrases that are exact matches to queries [Gutwin et al. 1998]. An example is shown in Figure 2, where five common phrases are stored with postings lists. Each posting has the form:

$$\langle d, f_{d,p} \rangle$$

where d is a document containing phrase p and the frequency of p in d is $f_{p,t}$. Offsets are not stored, as we only experiment with exact matching and so the location of the phrase within a document is not required.
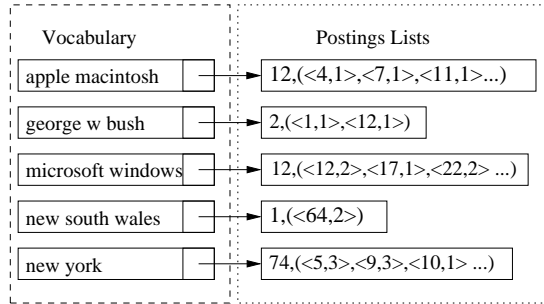
Fig. 2.   *A partial phrase index with a vocabulary of five popular phrases.*

Query evaluation using this structure is straightforward. For each phrase query posed by a user, the phrase vocabulary is searched. When an exact match is found, the postings list is retrieved, and the document identifiers used to retrieve and display the answers. If an exact match is not located, then another approach must be followed to resolve the query; possible choices include the partial nextword index structure we describe in the next section, or using an inverted index and the conventional algorithm we described previously.

The partial phrase index affords the greatest efficiency savings for common queries and for queries that are difficult to resolve using other approaches. For example, with only an inverted index, the phrase query `new york` requires the retrieval and processing of the long postings list for the common word "new". In contrast, the postings list for the phrase "new york" in the partial phrase index is at most as long as the postings list for "york" and probably somewhat shorter. Moreover, precomputation of the answer set saves the cost of merging lists.

The usefulness of a partial phrase index is dependent on successfully predicting future queries and storing them in the structure. A simple approach is to store frequent past queries and we report experiments with this approach later. Consider the queries `lord of the rings` and `britney spears` that occur 12 and 59 times respectively in our log. We can speculate that a frequent phrase in the past can indicate a high likelihood for its recurrence in the future. It is thus straightforward to argue that a frequent query, such as the latter query, should be part of the phrase index.

However, given a stream of queries over a long period and a fixed volume of memory, a strategy may also be required to update the vocabulary so that the least-recently used or least-frequently used queries are replaced; we have not experimented with this in practice. This strategy is a form of query-result caching, which has been shown to significantly reduce the cost of processing a stream of queries [Saraiva et al. 2001]; in contrast to standard caching strategies, however, our intention is to maintain collections of queries for which inverted lists rather than explicit answers sets are maintained. There may also be better strategies for choosing queries — such as selecting those that are difficult to evaluate with an inverted index or weighting the choice to those that have occurred more recently — but we leave this for future work when larger query logs are available.

Interestingly, in contrast to our motivating example `new york`, common phrase queries do not appear to be those that are difficult to evaluate. However, as we show

later, the partial phrase index is still efficient because the total saving over the set of common queries is significant. For example, in the Excite log we use in our experiments, the most common queries are (with frequency in brackets): `thumbnail posts` (198), `thumbnail post` (198), `jennifer lopez` (126), `the cranberries` (79), and `santa claus` (76). With the exception of `the cranberries`, there are no common words in the queries that would necessitate the retrieval of long inverted lists from a word-based inverted index.

A related approach to partial phrase indexing is using an inverted index and approximating phrases through a ranked query technique [Clarke et al. 1997; Lewis and Croft 1990]. We do not experiment with this approach in this paper.

4.2 Nextword indexes

Fast phrase query evaluation is possible with the partial phrase index described in the previous section. However, when an exact match cannot be found, a different strategy is required. One possibility is to resort to evaluation using the conventional inverted index approach described in Section 3. Another possibility is to use another special-purpose structure for phrase querying that complements the partial phrase index and is faster than an inverted index.

An efficient, special-purpose structure with no additional in-memory space overheads is the *nextword index* [Williams et al. 1999], a search structure designed to accelerate processing of word pairs. As noted previously and observed elsewhere, the commonest number of words in a phrase is two [Spink et al. 2001]. In addition, a phrase query can never be less than two words in length, and therefore all phrase queries can be decomposed into (overlapping or non-overlapping) pairs of words; as we show later, this permits using a nextword index to resolve phrase queries of all lengths.

A nextword index is similar to an inverted index in that it uses a set of distinct terms at the root, and posting lists at the bottom of the retrieval structure. However, a nextword index is a three-level structure that is optimized for the retrieval of word-pairs. (For clarity we use *inverted index* solely to refer to a standard word-level inverted file. However, we note that nextword indexes are a form of inverted index.)

In a nextword index, any distinct word-pair such as "web page" is composed of two words $v =$ "web" and $w =$ "page" and we call a distinct word v a *firstword*, while w is a *nextword* of v. Nextword indexes organize firstwords and nextwords such that each firstword is associated with a set of nextwords, and each nextword in turn has a posting list. Finding the word "web", then its nextword "page", and then retrieving the posting list is thus equivalent to answering the phrase query `web page` [Williams et al. 1999].

Part of an example nextword index is shown in Figure 3. In this example, there are two firstwords shown, "hatful" and "of". Two of the nextwords for "of" are "hollow" and "house". The firstword "hatful" has only one nextword, "of". For each firstword-nextword pair, there is a postings list. Posting information for word-pairs is organized as for inverted indexes.

Using the same syntax as previously, but replacing the notation for a word w with the notation of a word-pair wp, a nextword postings list is as follows:

$$f_{wp}, \langle d, f_{d,wp}, [o_1, \ldots, o_n] \rangle, \ldots, \langle \ldots \rangle$$
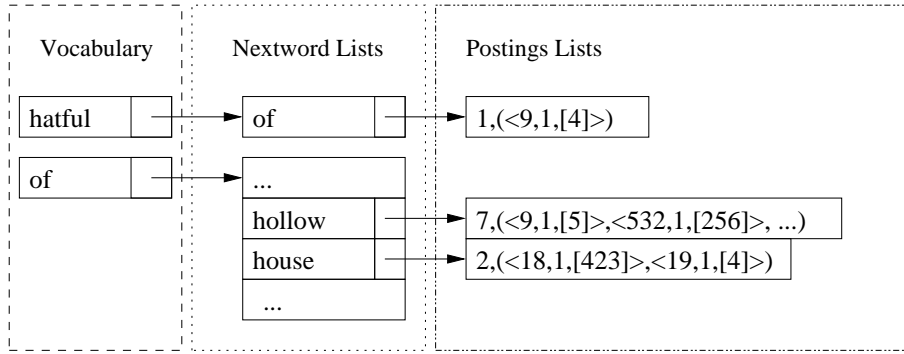
Fig. 3.   *A nextword index with two firstwords.*

Posting lists begin with f_{wp}, the number of documents in which a word-pair wp occurs. The remaining information are triples, that repeat f_{wp} times. Every such triple stores information about the occurrence of wp in a document d. The document word-pair frequency $f_{d,wp}$ shows how often a document d contains wp, and the o values indicate the ordinal position of wp in d.

The postings lists in nextword indexes are typically short, because most pairs 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 for such pairs can be extremely fast.

The original description of nextword indexes [Williams et al. 1999] proposed storing nextwords in an alphabetically sorted front-coded list on disk. Consider a firstword "of" that has nextwords that include "an", "a", "and", and "at". These are sorted alphabetically and compressed by storing the length in characters of the common prefix between the current word and the previous word, the number of new characters that suffix that prefix, and then the suffix itself. For our example, the nextwords are represented as:

$$\langle 0, 1, \mathrm{a} \rangle \ \langle 1, 1, \mathrm{n} \rangle \ \langle 2, 1, \mathrm{d} \rangle \ \langle 1, 1, \mathrm{t} \rangle$$

This representation requires that the nextword list is processed sequentially, that is, each word can only be decoded relative to the previous word. Williams et al. [1999] store the integer values using a fast, variable-byte representation [Williams and Zobel 1999] and store the suffix characters in ASCII.

We use a simple variation of the original approach that affords faster processing of nextwords. We store every $\sqrt{f_{v,w}}$th nextword in a main-memory array, where $f_{v,w}$ is the total number of nextwords w for that firstword v. Each entry holds a nextword, a pointer to its posting list, and a pointer to the next nextword in the nextword list. The in-memory array is searched sequentially for a close match, and the on disk list is searched for an exact match in a second step; since nextwords are stored uncompressed in main-memory, random access to the front-coded nextword list is possible. (We experimented with many different variants of this main-memory approach, and found this to be the most efficient and effective.)

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 care. For example, the query

```
boulder municipal employees credit union
```

can be resolved by fetching the lists for the pairs "boulder"·"municipal", "employees"·"credit", and "credit"·"union". Alternatively, it is possible to fetch the lists for "boulder"·"municipal", "municipal"·"employees", and "credit"·"union". The most efficient plan depends on whether the list for "employees"·"credit" is shorter than the list for "municipal"·"employees".

Unfortunately, establishing which list 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 length of its nextword list. Thus, for example, 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".

Generalising, 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. We have described and evaluated algorithms for choosing order of evaluation elsewhere [Bahle 2003; Bahle et al. 2001b].

Summarising our discussion, an efficient algorithm for evaluating phrase queries $q_1 \ldots q_n$ with a nextword index is as follows.

(1) Identify all $n-1$ firstword-nextword pairs $q_i \cdot q_{i+1}$; then sort them by increasing firstword frequency; then discard from the list the pairs that are completely covered by preceding selections.
(2) This yields a sequence of pairs $P_1, \ldots, P_m$. Each pair has an inverted list. These lists are processed in turn as for phrase processing with an inverted file.

The nextword index for the WT10g collection is 2.75 Gb in size, almost exactly twice that of an inverted file. As shown in Table III, the savings in query evaluation time are dramatic. Average query evaluation time is reduced to 0.02 seconds, faster than inverted files by a factor of around fifty. For two-word queries, the time falls to 0.01 seconds from 0.35. Similar savings are observed for all query lengths, with (for the higher lengths) variations due to behaviour on individual problematic queries.

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 in the introduction that around 41% of all queries in the Excite log successfully evaluate as phrase queries, 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. [2001] 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.

Table III. *Retrieval times (seconds) for inverted and nextword indexes with phrase queries from the Excite query log resolved on WT10g.*

Phrase length	Inverted index	Nextword index
All	1.04	0.02
2	0.35	0.01
3	1.62	0.04
4	3.73	0.03
5	3.80	0.27
6	4.67	0.10
7	4.99	0.16

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 to have complementary advantages, it is attractive to try to combine their strengths, and we explore this in the next section.

5. COMBINED QUERY EVALUATION

We observed in Section 3 that inverted indexes are slowest for phrases involving common words, the case where nextword indexes yield the greatest speed advantage; however, we have also shown that a nextword index is a large structure. Phrase indexes provide a third dimension to this trade-off: they can be used to manage index information for common queries (as opposed to queries involving common words). It is therefore attractive to combine the approaches so that the benefits of speed for nextword index and partial phrase index evaluation, and a compact inverted index are combined.

Different methods of combining are considered below and all these have one aim, to construct a compact partial phrase or partial nextword index that is used together with a complete inverted index. Nextword and phrase indexes should arguably be partial, because certain queries can be resolved as efficiently with the inverted index. In contrast, the inverted index must be complete, because other query modes — such as ranked or Boolean querying — still need to be fully supported.

5.1 Combined inverted and nextword indexes

We propose that common words only be used as firstwords in a partial nextword index, and that this new index be used where possible in evaluation of phrase queries. We have explored other schemes based on the frequency of words in the indexed collection, or based on the frequency of words in the query log. None of these offered a better space and time trade-off.

An example of a combined index is shown in Figure 4. 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 "of" and "was" 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 `hatful of hollow`. The
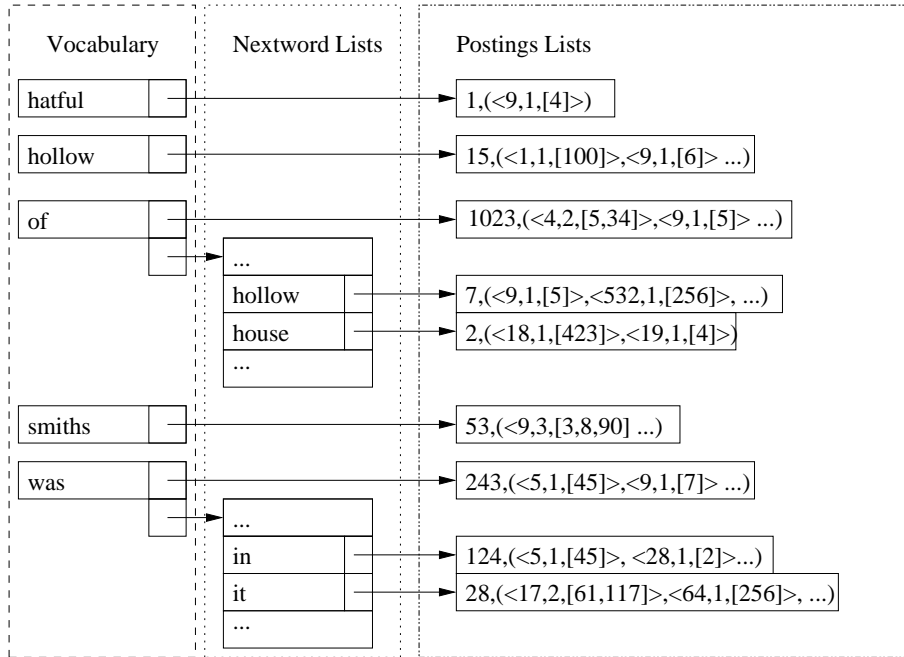
Fig. 4. *A combined inverted file and partial nextword index.*

term "hatful" is not a common word, so establishing that it occurs in the phrase involves fetching its postings lists from the inverted index and processing in the usual way. However, since "of" is common, there is a choice: first, we can retrieve the posting list for the firstword-nextword pair "of"·"hollow" from the partial nextword index; or, second, we can to retrieve the posting lists for "of" and "hollow" from the inverted index, and process in the usual way. The benefit of the first approach is that the nextword index postings list for "of"·"hollow" cannot be longer than the inverted index postings list for "hollow" 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 identify all pairs in the list in which the first term is an indexed firstword, and all terms not in a firstword-nextword pair. The postings lists are then processed in increasing order of collection document frequency $d_{f,t}$, 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 is rare in the Excite query log.

We state this algorithm as follows. For a phrase query, $q_1 \ldots q_n$ with $n > 1$ words, we make use of both inverted and nextword indexes:

(1) Create a set $\mathcal{P} = \{P_1, \ldots, P_m\}$ of pairs $q_i \cdot q_{i+1}$ such that q_i is an indexed firstword.
(2) Form set $\mathcal{Q}$ of the remaining query terms.

(3) Associate the document frequency with each member of $\mathcal{P}$ and $\mathcal{Q}$.

(4) Form the union $R = \mathcal{P} \cup \mathcal{Q}$, and sort the members of R by increasing document frequency.

(5) This yields a sequence of members $R_1, \ldots, R_m$, where $m \leq n$. Each member has an inverted list. These lists are processed in turn as for phrase processing with an inverted file.

We have tested other query resolution methods that involved term sorting based on nextword frequency $f_{v,w}$, firstword frequency (the number of pairs in the collection in which the firstword appears as the left-most word), and combinations of these and document frequency. 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 document frequency only. We therefore sort inverted index terms and nextword terms based on document frequency since we do not need to keep another statistical value per index term and sorting is straightforward.

5.2　Combined inverted and phrase indexes

The second combination that we propose is that common phrases be evaluated with a partial phrase index where possible, and that otherwise an inverted index be used. In this approach, the vocabulary of searchable terms consists of both words and phrases, and each search term has an inverted list. In our implementation, we store the phrase vocabulary in a separate, compact search structure [Zobel et al. 2001].

With this combined index, processing involves postings lists from both the inverted index and the partial phrase index. Consider again the query `new york`, and the phrase index shown in Figure 2. This phrase is treated as a single term, and looked-up in the phrase vocabulary. As it is present in the structure, the postings list is retrieved and the query evaluation is complete. Consider our other query `hatful of hollow`. Again, the phrase is treated as a single term and searched for in the partial phrase index. As it is not present as a phrase, the inverted index is used as described in Section 3 to evaluate the query. The benefit of finding a query `new york` in the phrase index are that a short postings list is retrieved — it cannot be longer than the lists for either "new" or "york" — and no management of temporary structures or merging is required. In contrast, the drawback of not finding the query `hatful of hollow` in the phrase index is that an extra lookup is required prior to the standard query evaluation being followed.

We state this algorithm as follows. For a phrase query, $q_1 \ldots q_n$ with $n > 1$ words, we make use of both inverted and partial phrase indexes:

(1) Using the partial phrase index, look for an exact match to the query $q_1 \ldots q_n$.
   (a) If found, fetch and decode the postings.
   (b) If not found, then proceed with the inverted index query plan described in Section 3 beginning with sorting the n query terms t by their collection document frequency.

As discussed previously, we have not experimented with using partial phrase indexes for partial matching. For example, the query `new york city` can be resolved using the partial phrase index to find the locations of "new york" and merging with

the inverted index postings list for "city". However, this requires that all contiguous overlapping subphrases of the query are searched for in the partial phrase index (in this example, "new york city", "new york", and "york city"), or that a heuristic is available to determine which phrases are likely to be in that index. Moreover, a requirement for partial matching prohibits use of hashing, the fastest look-up technique for this application, and it is only in phrases of more than two words that it would be of value. It seems unlikely that approaches of this kind would be faster in practice than resolving the query using conventional techniques, and we chose not to continue investigation of partial matching.

5.3   Three-way index combination

The third and final combined index that we consider is that of a partial nextword, partial phrase, and complete inverted index. As discussed previously, we believe this combination is likely to yield significant advantages: the partial nextword index can accelerate the processing of word pairs that begin with a common word, the partial phrase index can resolve frequent phrases that do not typically contain common words, and the inverted index can be used for the remaining, less difficult cases.

As we consider only exact matching with the partial phrase index, there is only one possible strategy for query evaluation with this combination. First, a phrase query is looked-up in the partial phrase index; if it is found, then the query is resolved and, if not, a second step is required. Second, the combined plan for a partial nextword and complete inverted index is pursued: as detailed previously, word pairs in which the firstword is present in the nextword index are evaluated with the nextword structure, while the remaining terms are evaluated through the inverted index; processing is ordered by increasing document frequency.

We state this algorithm as follows. For a phrase query, $q_1 \ldots q_n$ with $n > 1$ words, we make use of the full inverted, partial phrase, and partial nextword indexes:

(1) Using the partial phrase index, look for an exact match to the query $q_1 \ldots q_n$.
   (a) If found, fetch and decode the postings.
   (b) If not found, then proceed with the combined inverted and nextword index plan described in Section 5.1.

6.   EXPERIMENTAL RESULTS

In this section, we report our experiments with combined indexes for phrase querying. We use the Excite phrase query log and WT10g collection described in Section 2.1. All experiments were run on an Intel 700 MHz Pentium III-based server with 2 Gb of memory and data stored on local disks, running the Linux operating system under light load.

6.1   Inverted + nextword

In Table IV we show sizes of nextword indexes in which only the commonest terms are allowed as firstwords. The table shows that a nextword index that contains only the three commonest terms consumes 192 Mb, that is, just over 13% of the space of the inverted index or around 2% of the size of the original HTML collection. The main-memory overheads of storing frequent words, and the structure to support skipping into nextword lists, are insignificant.

Table IV. *Overheads for a combined inverted and partial nextword index, with different numbers of common words used in the nextword index. The percentage is the index size compared to original data size.*

	Number of firstwords						
	3	6	12	24	48	96	192
Disk (Mb)	192	309	404	514	619	730	865
Disk (%)	1.83	2.94	3.84	4.89	5.88	6.94	8.23
Memory (kb)	30	55	92	149	230	366	594

Query evaluation time with a combined index is shown in Figure 5. The figure shows the tradeoff between the size of the additional index structures — as shown in Table IV — and query evaluation times. As can be seen by the solid line, phrase query evaluation times fall dramatically as additional structures of less than 200 Mb are added to the inverted index structures. Indeed, use of a partial nextword index of 2% of the HTML collection size more than halves query evaluation time; a partial nextword index of 4% of the size of the collection cuts time to a third. These are substantial savings at low cost. Perhaps the most significant impact is on index update costs, as the number of inverted lists to be modified on a document insertion is increased by the number of distinct firstword-nextword pairs in that document. With a nextword index on the three commonest words, the increase in number of lists would be around 10%. The increase in maintenance time would be smaller than 10%, as short lists are relatively cheap to modify.

6.2 Inverted + phrase

As discussed previously, entries in our partial phrase index are selected from frequent past queries. To do this, we split the Excite query log in half, and we refer to the first and last 66,000 queries as the *head* and *tail* of the log respectively. We have used the most frequent 100, 1000, and 10,000 distinct phrase queries from head of the query log to determine phrase index entries, and used the tail for querying. In all, we have executed four experiments, one with the full inverted index structure to determine a baseline measure and three more with the above phrase index configurations. The space required to store the phrase indexes is less than 0.1% of the collection size: 2.1 Mb, 4.8 Mb, and 12.8 Mb respectively for the three experiments.

The average times for resolving the 66,000 tail queries with the inverted index and the three partial phrase indexes are shown in Table V. Average timings are shown for all queries and for queries with lengths from two to seven words. Overall, storing 10,000 queries in the partial phrase index reduces the average query evaluation cost by around 15%.

Around 70% of all queries in the larger of our three partial phrase indexes are two-word queries, 21% are three-word queries, and about 6% are four-word queries. There are also longer queries such as `an american dictionary of the english language` and `los angeles department of water and power`, but queries of this length are rare and, in any case, the 10,000 most frequent queries includes some that occur only once. As discussed in Section 5, larger-scale studies of how to choose and maintain a phrase index are desirable but would require larger, publicly-available query logs.
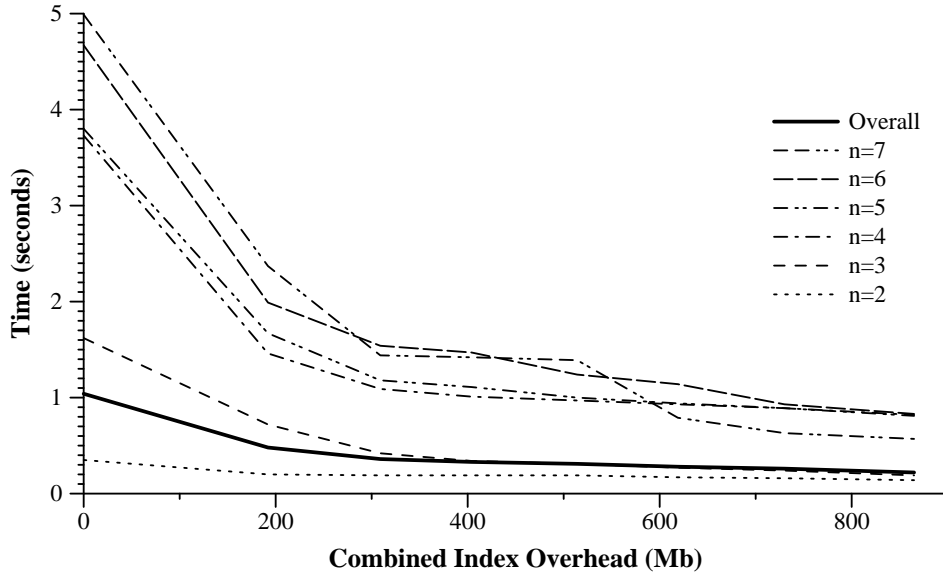
Fig. 5. *A plot of times for phrase query evaluation (seconds) on a combined index versus space required (Mb) for the index structures. The space requirements are those shown for the numbers of firstwords listed in Table IV. The solid line shows overall performance for all query lengths. The other lines show the performance for queries of length $n = 2$ to $n = 7$.*

We expect, based on the above observations, that short queries improve on query time efficiency, rather than long queries. This expectation is met by the results, where queries with five to seven words improve at best by about 6% in comparison to shorter queries, which for two words improve by around 29%. These results are particularly good considering that we use a simple scheme of selecting phrases based on frequency of occurrence, and additional memory and disk storage costs are tiny.

6.3 Inverted + phrase + nextword

Table VI shows the average timings for phrase queries of lengths of two to seven words for the inverted index, and for partial nextword and phrase indexes used in combination with the inverted index. Comparing these results to those in the previous two sections shows that the improvements afforded by both phrase querying structures are complementary. That is, both phrase and nextword indexes improve subsets of queries that do not overlap significantly. For example, queries with a length of two words have improved in by 60% compared to the inverted index and partial phrase index combination, and by 15% compared to the combined inverted index and partial nextword index. Overall, depending on the choice of number of firstwords in the partial nextword index, phrase querying is between 60%–80% faster than using an inverted index alone.

Table V. *Average query times (seconds) for inverted index queries from tail of query log with phrase indexes constructed from the most frequent head queries of the Excite query log, taking the top 0, 100, 1,000, and 10,000 queries.*

Query	Number of frequent queries			
length	0	100	1,000	10,000
All	1.04	1.03	0.98	0.89
2	0.35	0.34	0.29	0.25
3	1.62	1.61	1.57	1.44
4	3.73	3.74	3.51	3.15
5	3.80	3.67	3.64	3.58
6	4.67	4.68	4.69	4.46
7	4.99	5.00	5.01	4.94

Table VI. *Average query times (seconds) of around 66,000 tail queries resolved with a combined full inverted index, an index of 10,000 common phrases, and a nextword index based on 0 to 192 most common firstwords.*

Query	IF	Number of firstwords						
length	only	3	6	12	24	48	96	192
All	1.04	0.41	0.30	0.27	0.26	0.24	0.22	0.20
2	0.35	0.14	0.14	0.14	0.13	0.13	0.12	0.11
3	1.62	0.64	0.39	0.31	0.28	0.25	0.22	0.17
4	3.73	1.32	1.00	0.92	0.88	0.84	0.81	0.74
5	3.80	1.62	1.13	1.06	0.95	0.89	0.85	0.77
6	4.67	1.78	1.39	1.33	1.21	1.12	0.91	0.81
7	4.99	2.31	1.41	1.41	1.37	0.78	0.63	0.56

7. CONCLUSIONS

Phrase queries are important tool for meeting information needs: around 10% of queries to web search engines are explicit phrase queries, and more than 40% of all queries have answers if they are evaluated using phrase techniques. We have proposed that phrase queries on large text collections be supported by use of small auxiliary indexes. 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 and the commonest phrases are indexed as terms in the inverted index. The nextword index stores postings lists for firstword-nextword pairs. We have experimented with different combinations of these structures.

Table VII summarises the results from our experiments. Phrase querying with an inverted index alone is slow at an average of 1.04 seconds per query. In the past, search engines have chosen to solve this problem by introducing stopping, which we have found reduces times to around 0.20 seconds per query but introduces mismatches and prevents queries containing only common words from being evaluated at all. Search engines no longer use stopping in phrase querying, and we agree that stopping is unacceptable for this task.

Depending on the availability of disk, we have shown that our solutions can reduce the average time to evaluate a phrase query to around or less than that of a stopping-based approach. With a large, complete nextword index, phrase querying is 50 times faster than using an inverted index or 10 times faster than the stopped

Table VII.   *Summarising space and time efficiencies observed in all experiments with WT10g.*

Scheme	Total index size (Mb)	Av. query time (sec)
Inverted	1,429	1.04
Inverted (stopping)	1,002	0.20
Nextword index	2,816	0.02
Inverted + nextword, 24 firstwords	1,943	0.31
Inverted + phrase, top 10,000	1,442	0.89
Combined	1,956	0.26

alternative. More practically, we have proposed that combinations of partial phrase, partial nextword, and full inverted indexes be used and we have shown that these are fast with low disk overheads. Most significantly, phrase querying with a combination of all three approaches is more than 60% faster on average than using an inverted index alone and requires structures that total only 20% of the size of the collection. We conclude that our approaches make stopping unnecessary and allow fast query evaluation for all phrase queries.

Our schemes have scope for improvement. In particular, we are further investigating structures for representing nextword lists. As larger query logs become available, we are also keen to investigate other strategies for selecting common phrases for the partial phrase index, and strategies for removing queries when they become uncommon.

ACKNOWLEDGMENTS

REFERENCES

Anh, V. N., de Kretser, O., and Moffat, A. 2001. Vector-space ranking with effective early termination. In *Proc. ACM-SIGIR Int. Conf. on Research and Development in Information Retrieval*, W. B. Croft, D. J. Harper, D. H. Kraft, and J. Zobel, Eds. ACM Press, New Orleans, Louisiana, 35–42.

Bahle, D. 2003. Efficient phrase querying. Ph.D. thesis, School of Computer Science and Information Technology, RMIT.

Bahle, D., Williams, H. E., and Zobel, J. 2001a. Compaction techniques for nextword indexes. In *Proc. String Processing and Information Retrieval Symp.* IEEE Computer Society Press, Los Alamitos, California, San Rafael, Chile, 33–45.

Bahle, D., Williams, H. E., and Zobel, J. 2001b. Optimised phrase querying and browsing in text databases. In *Proc. Australasian Computer Science Conf.*, M. Oudshoorn, Ed. Conferences in Research and Practice in Information Technology. Australian Computer Society, Gold Coast, Australia, 11–19.

Bailey, P., Craswell, N., and Hawking, D. 2003. Engineering a multi-purpose test collection for web retrieval experiments. *Information Processing & Management 39,* 6, 853–871.

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

Croft, W. B., Turtle, H. R., and Lewis, D. D. 1991. The use of phrases and structured queries in information retrieval. In *Proc. ACM-SIGIR Int. Conf. on Research and Development in Information Retrieval*, A. Bookstein, Y. Chiaramella, G. Salton, and V. V. Raghavan, Eds. ACM Press, New York, Chicago, Illinois, 32–45.

DE LIMA, E. F. AND PEDERSEN, J. O. 1999. Phrase recognition and expansion for short, precision-biased queries based on a query log. In *Proc. ACM-SIGIR Int. Conf. on Research and Development in Information Retrieval*, M. Hearst, F. Gey, and R. Tong, Eds. ACM Press, Berkeley, California, 145–152.

GUTWIN, C., PAYNTER, G., WITTEN, I., NEVILL-MANNING, C., AND FRANK, E. 1998. Improving browsing in digital libraries with keyphrase indexes. *Decision Support Systems 27,* 1/2, 81–104.

HARMAN, D. 1995. Overview of the second text retrieval conference (TREC-2). *Information Processing & Management 31,* 3, 271–289.

JANSEN, B. AND POOCH, U. 2001. A review of web searching studies and a framework for future research. *Jour. of the American Society for Information Science and Technology 52,* 3, 235–246.

LEWIS, D. D. AND CROFT, W. B. 1990. Term clustering of syntactic phrases. In *Proc. ACM-SIGIR Int. Conf. on Research and Development in Information Retrieval*, J.-L. Vidick, Ed. ACM, Brussels, Belgium, 385–404.

MOFFAT, A. AND ZOBEL, J. 1996. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems 14,* 4 (Oct.), 349–379.

PAYNTER, G. W., WITTEN, I. H., CUNNINGHAM, S. J., AND BUCHANAN, G. 2000. Scalable browsing for large collections: A case study. In *Proc. ACM Digital Libraries*. ACM Press, New York, San Antonio, California, 215–223.

PERSIN, M., ZOBEL, J., AND SACKS-DAVIS, R. 1996. Filtered document retrieval with frequency-sorted indexes. *Jour. of the American Society for Information Science 47,* 10, 749–764.

SARAIVA, P. C., MOURA, E. S., ZIVIANI, N., FONSECA, R., MEIRA, W., MURTA, C., AND RIBEIRO-NETO, B. 2001. Rank-preserving two-level caching for scalable search engines. In *Proc. ACM-SIGIR Int. Conf. on Research and Development in Information Retrieval*, W. B. Croft, D. J. Harper, D. H. Kraft, and J. Zobel, Eds. ACM Press, New Orleans, Louisiana, 51–58.

SOBOROFF, I. 2002. Does WT10g look like the web? In *Proc. ACM-SIGIR Int. Conf. on Research and Development in Information Retrieval*, M. Beaulieu, R. Baeza-Yates, S. H. Myaeng, and K. Jävelin, Eds. ACM Press, New York, Tampere, Finland, 423–424.

SPINK, A., WOLFRAM, D., JANSEN, B. J., AND SARACEVIC, T. 2001. Searching the web: The public and their queries. *Jour. of the American Society for Information Science 52,* 3, 226–234.

SPINK, A. AND XU, J. 2000. Selected results from a large study of web searching: the Excite study. *Information Research 6,* 1. Available at: http://InformationR.net/ir/6-1/paper90.html.

VOORHEES, E. M. AND HARMAN, D. K. 2001. Overview of TREC 2001. In *The Tenth Text REtrieval Conference (TREC 2001)*, E. M. Voorhees and D. K. Harman, Eds. National Institute of Standards and Technology Special Publication 500-250, Gaithersburg, MD, 1–15.

WILLIAMS, H. E. AND ZOBEL, J. 1999. Compressing integers for fast file access. *Computer Jour. 42,* 3, 193–201.

WILLIAMS, H. E., ZOBEL, J., AND ANDERSON, P. 1999. What's next? index structures for efficient phrase querying. In *Proc. Australasian Database Conf.*, M. Orlowska, Ed. Springer-Verlag, Auckland, New Zealand, 141–152.

WITTEN, I. H., MOFFAT, A., AND BELL, T. C. 1999. *Managing Gigabytes: Compressing and Indexing Documents and Images*, Second ed. Morgan Kaufmann, San Francisco, California.

ZOBEL, J. AND MOFFAT, A. 1998. Exploring the similarity space. *SIGIR Forum 32,* 1 (Spring), 18–34.

ZOBEL, J., WILLIAMS, H. E., AND HEINZ, S. 2001. In-memory hash tables for accumulating text vocabularies. *Information Processing Letters 80,* 6 (Dec.), 271–277.