Compaction Techniques for Nextword Indexes

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

Abstract

Most queries to text search engines are ranked or Boolean. Phrase querying is a powerful technique for refining searches, but is expensive to implement on conventional indexes. In other work, a nextword index has been proposed as a structure specifically designed for phrase queries. Nextword indexes are, however, relatively large. In this paper we introduce new compaction techniques for nextword indexes. In contrast to most index compression schemes, these techniques are lossy, yet as we show allow full resolution of phrase queries without false match checking. We show experimentally that our novel techniques lead to significant savings in index size.

1 Introduction

Web search engines are used to answer information needs around 250 million times per day.[1] Most queries to textual information retrieval systems such as web search engines are informal, bag-of-words queries that contain two or three terms. Trained users pose other query types, such as Boolean queries, that require more complex formulation. For ranked and Boolean queries, much is known about implementing fast and accurate search systems [8, 11].

The majority of information needs can be met with ranked and Boolean queries: that is, with ranked and Boolean queries, users can quickly find documents (or links to documents) that satisfy their requirements. However, some information needs require alternative querying techniques, and some documents simply cannot be found with bag-of-word queries. A search technique that helps address these issues is phrase querying, where users pose a query in which the ordering and adjacency of the search terms is important. For a Web search engine, a user typically expresses a phrase query as a quoted string such as "Stan and Oliver". An answer to a phrase query is the set of documents—or summaries and links to documents—containing that exact phrase. Phrase querying is not necessarily an exclusive alternative: it is often combined with ranked or Boolean

[*]Contact author. Postal Address: Hugh Williams, School of Computer Science and Information Technology, RMIT University, GPO Box 2476V, Melbourne 3001, Australia. Telephone: +61 3 9925 4149. Facsimile: +61 3 9925 4098. Email: hugh@cs.rmit.edu.au

[1]See `http://www.searchenginewatch.com/reports/perday.html`

querying to provide additional expressiveness. Moreover, phrase queries can be evaluated using an augmented index structure that also supports the basic query modes.

There are other search techniques that have not been applied to large collections. An example is phrase browsing, a technique to allow the user for exploration of the vocabulary used in the collection and the phrases formed by the words in the vocabulary. Such a technique is particularly powerful when a user cannot easily express a concept in a few words, or does not have a concrete list of words to express the concept. Phrase browsing has been shown to be practical for small collections [6, 7, 10], but it is unclear how these techniques may work for large collections.

In a query log from the Excite search engine [9] of around 1.8 million queries, almost 7% were phrase queries or contained a phrase query component. Efficient evaluation of phrase queries is therefore crucial to overall retrieval performance. An index structure specifically designed for fast phrase querying has previously been proposed and compared to conventional structures [10]. This structure—the *nextword* index—has been shown to be around five times faster than a conventional structure for resolving typical two- or three-word phrase queries. An additional benefit of the structure is that it supports a form of mono-directional phrase browsing in large collections. The drawback is that the nextword index is around 51% of the size of the indexed text when conventional compression techniques are used.

In this paper, we propose two novel compaction techniques to reduce the size of nextword indexes. We show that more effective compaction is possible if the relationships between words in a document are considered when storing the locations of pairs of words. These techniques are lossy, but still allow phrase querying without false matches, and thus are strikingly different to conventional index coding techniques. We show that both schemes are a sound and complete equivalent to a conventional approach. With careful optimisation of these novel techniques, we show that for web data the size of a nextword index can be reduced from 56% to 49% of the size of the indexed text.

The drawback of our new compaction schemes is that query optimisation is difficult. It has been shown elsewhere that careful consideration of which pairs of words in a phrase are to be evaluated, and the order of their evaluation, is an important performance consideration [1]. It is unclear how these techniques may be applied to the new compaction approaches, and for this reason long queries cannot be evaluated as fast as with our earlier nextword compression scheme. Overall, however, we believe that the benefits of additional space savings justify the cost of a slight increase in computational cost.

2 The Nextword Index Structure

The nextword index shown in Figure 1 can be divided into two major components, an *on-disk* component and an *in-memory* component. The in-memory component is a fast in-memory lookup structure for single words that occur in the indexed collection. This component is the starting point of any query and it allows a query to be terminated quickly when a query contains a word that does not occur in the indexed collection.

The on-disk component is used together with the in-memory component to verify the existence of query phrases in the collection. The on-disk component can be broken down further into a

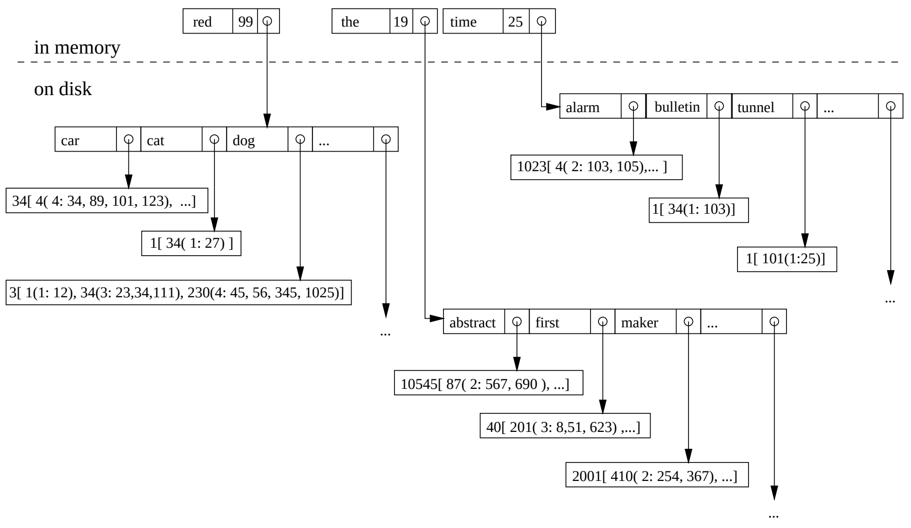

Figure 1: *Organisation of a nextword index.*

Text	the	first	time	the	red	dog	saw	the	red	cat
Offset	1	2	3	4	5	6	7	8	9	10

Figure 2: *A sample document with ordinal offsets.*

vector file and a *nextword file*. The nextword file contains lists of words that occur after a specific word in the collection. We refer to the words in such a list as *nextwords*. A list of case-folded nextwords for the word "and" stored in a nextword file could be:

oliver, silent, stan, was, white, yes

The nextwords are each paired with the word "and", signifying that the word pairs "and Oliver", "and silent", "and Stan", and so on occur in the collection.

In the nextword file, the nextwords are interleaved with offset pointers to *inverted lists* in the vector file. Each nextword has one such inverted list. An inverted list contains all positions where the word pair occurs in the collection, including the document numbers and word offsets from the start of the documents; we refer to this offset representation scheme as an *ordinal* scheme and we describe it formally below. We discuss how a list is stored in a vector later.

Nextword indexes can be described formally as follows.

Definition 2.1 A *document* d is a sequence of word occurrences $w_1 \ldots w_n$. Each word w_i is labelled with an *offset* o_i. Offsets are positive integers. □

Definition 2.2 The offsets $o_1, \ldots, o_n$ in a document d are *ordinal* if $o_i = i$ for $1 \leq i \leq n$. □

An example document is shown in Figure 2, with an ordinal offset scheme.

Definition 2.3 A *nextword index* is a mapping from pairs of terms $f \cdot s$ to lists. A *list* is a set of (d, B) pairs, where each d is a document identifier and B is a set of indicators. An *indicator* b describes an occurrence of the pair $f \cdot s$ in d at *position* k, that is, an occurrence of $f = w_k$ and

$s = w_{k+1}$ for some k where $1 \leq k < n$. An indicator i for pair $p \cdot s$ at position k in document d is *elementary* if $i = o_k$ is an integer. □

We use the term "offset" to refer to both indicators and offsets, where doing so is not ambiguous.

A fragment of a nextword index is shown in Figure 1. In this figure, the offset scheme is ordinal and the indicators are elementary. For example, this index contains the information that "red cat" occurs once in document 34, at position 27, and that "red doc" is in three documents, including three occurrences in document 34.

Basic phrase query evaluation

The task of a phrase query retrieval engine is to find all documents in which the given sequence of words occur in the given order. A nextword index with ordinal offsets and elementary indicators can be used in a straightforward way to evaluate phrase queries. The principle of query evaluation is that the lists for the pairs in the query are intersected.

Definition 2.4 A *query* is a sequence of terms $t_1 \ldots t_m$. An *answer* is a document $w_1 \ldots w_n$ such that, for some $k > 0$, $t_i = w_{k-1+i}$ for $1 \leq i \leq m$. □

Definition 2.5 *Query evaluation algorithm (elementary).* Input is a nextword index and a a query $t_1 \ldots t_m$, which can be regarded as a sequence of pairs $t_i \cdot t_{i+1}$, for $1 \leq i < m$. Output is a set of documents.

1. The list l_1 for the first pair, $t_1 \cdot t_2$, is retrieved from the nextword index. This is the initial list L_1 of candidate answers. (All the answer documents are in L_1, which however may also contain documents that are not answers.)
2. For each remaining pair $t_i \cdot t_{i+1}$, for $2 \leq i < m$, and while L_{i-1} is not empty,
   (a) Create an empty list L_i.
   (b) The list l_i for the pair $t_i \cdot t_{i+1}$ is retrieved from the nextword index.
   (c) For each document identifier d that is in both L_{i-1} and l_i,
      i. B_L is the set of indicators for d in L_{i-1} and B_l is the set of indicators for d in l_i. $B_{l'}$ is a new set of indicators, initially empty.
      ii. For each indicator $b' = o_k \in B_l$ where $b = o_{k-1} \in B_L$, add b' to $B_{l'}$.
      iii. Add the pair $(d, B_{l'})$ to L_i if $B_{l'}$ is not empty.
3. The documents $\mathcal{D} = \{d | (d, B) \in L_{m-1}\}$ are the answers to the query. □

The crucial element of this query evaluation algorithm is step 2(c)ii. Prior to this step, we have a set of indicators B_L that are the locations at which the partial phrase $t_1 \ldots t_i$ occurs in document d (where $i > 1$). These indicators consist of the offsets corresponding to each occurrence of term t_{i-1}. When the step is complete, we have a set of indicators $B_{l'}$ that are the locations at which the partial phrase $t_1 \ldots t_{i+1}$ occurs in document d; these indicators have been updated, and now correspond to occurrences of t_i rather than t_{i-1}.

An element that we have not specified in step 2(c)ii is how, given an offset o_k, we recognise o_{k-1}. If the offsets are ordinal this is straightforward: they are integers that differ by 1, that is, $o_k = o_{k-1} + 1$. We explore other offset and indicator schemes below.

Theorem 2.1 Given a collection D of documents d with ordinal offsets, an elementary nextword index, a query $q = t_1 \ldots t_m$, and a set $\mathcal{D}$ of documents returned by algorithm 2.5, then

1. (Soundness.) For each document $d \in \mathcal{D}$, the document d is an answer to q.
2. (Completeness.) For each document $d \in D$ that is an answer to q, document $d \in \mathcal{D}$.

Proof Statement 1 follows from the definitions of ordinal offsets and indicators. Inductively, suppose that if in d there is a phrase $t_1 \ldots t_i$ such that t_{i-1} has ordinal $o_k = k$ (where $k > 1$), and suppose there is an indicator b in the nextword index for $t_i \cdot t_{i+1}$ consisting of ordinal $b = o_{k+1} = k + 1$. Then d must contain the phrase $t_1 \ldots t_{i+1}$.

Statement 2 follows from assumptions concerning the completeness of the nextword index. If we assume that for every occurrence of every pair there is an indicator in the index, it follows by induction on step 2c that all answers will be in $\mathcal{D}$. $\square$

The completeness, or otherwise, of the index governs whether phrases are matched across punctuation. In practice, answers may be disallowed if the sequence $w_k \ldots w_{k-1+m}$ includes a punctuation symbol such as a full-stop or a semi-colon. This is trivially implemented by omitting indicators for pairs $w_k \cdot w_{k+1}$ where such punctuation occurs between the two words.

We illustrate phrase query evaluation on an example query using the example nextword index in Figure 1. Consider a phrase query "the red dog" and the process of verifying if and where it occurs in the collection. The process begins by partitioning the phrase into two word pairs: "the red" and "red dog". The in-memory component of the index is used to find if "the" is a word in the collection. As it does occur, we then find the offset pointer into the list of nextwords on disk for "red". We process the list of nextwords, find "red", and retrieve its inverted list from the vector file. From the inverted list for "the red", an in-memory list of document and offset occurrences is created. The result is a list of locations where the word pair occurred in the collection, but we do not know if it ever overlapped with "red dog".

The search process is repeated for "red dog". We retrieve the vector for "red dog" and eliminate from the in-memory "the red" list each location where "the red" occurred but did not overlap with "red dog". We call this elimination process *matching*. The selection of word pairs to process and the matching algorithm depends on the location information that is actually stored in the vector file inverted lists. Novel storage and matching techniques are presented later in this paper.

With ordinal offsets, this process can be generalised slightly: the word pairs do not need to be evaluated in order [1]. First, a word pair $t_i \cdot t_{i+1}$, where $i < m$, is selected from the set of word pairs in the query. Second, the in-memory component of the nextword index is used to verify if the word t_i occurs in the collection. Third, assuming the word t_i does occur, the nextword list is retrieved for t_i and this nextword list is searched for the word t_{i+1}. Fourth, assuming t_{i+1} is found to be a nextword of t_i, the inverted list for the word pair is retrieved from the vector file. Last, the inverted list is processed and an in-memory list of documents and offsets is created containing the locations of the word pair $t_i \cdot t_{i+1}$ in the collection. If the word t_i does not occur in the second step, or the nextword t_{i+1} does not occur in the third step, then the query process is terminated as the phrase does not occur. This word pair evaluation process is repeated for all word pairs $t_j \cdot t_{j+1}$ (where $j \neq i$). After processing each additional word pair, the in-memory list of documents and offsets (initially the list for $t_i \cdot t_{i+1}$) is intersected with the new documents and offsets. In this intersection process, the offsets must match: for example, if i and j differ by 3, then the offsets must also differ by 3.

It has been shown elsewhere [1] that the order of evaluation of word pairs and selection of which word pairs are to be processed affects the speed and main-memory use of nextword-based querying.

We return to query optimisation and evaluation later.

Nextword phrase browsing

Nextword indexes can be used for monodirectional phrase browsing. In this scheme, a list of words that follow a phrase can be found and presented to the user. For example, if the user posed the phrase "the red dog", phrase browsing may find that "ran", "sat", and "woofed" are all words that, somewhere in the corpus, occur immediately after the phrase. So, for example, the user can then determine that "the red dog woofed" is a phrase that occurs in the collection. Words that follow the longer phrases can also be found, so that the user can determine that, for example, "the red dog woofed loudly" occurs in the collection. The result is that it is possible for the user to browse a monodirectional hierarchy of phrases that occur in the collection that begin with one or more words provided by the user.

The process of phrase browsing with a nextword index can be divided into two stages: first, a phrase query stage using the techniques described in the previous section; and, second, a phrase browsing stage. The result of the phrase query stage is if and where a query phrase occurred in the collection. The phrase browsing stage retrieves all nextwords of the final word in the query and all lists in the vector file for each of the nextwords. All locations in all of the lists are matched with the in-memory locations from the phrase query stage. Words that do follow the phrase are saved, and the words are shown to the user.

The structure of the nextword index and the algorithms used for phrase query and phrase browsing are described in detail elsewhere [1, 10]. In the following section, we briefly discuss the structure and storage of inverted lists. We follow that with a discussion of possible compression improvements through nextword-specific compaction schemes.

Storing inverted lists

The vector file represents the leaves of the nextword index structure. It contains inverted lists that store location information about the occurrence of a wordpair in the text collection. More specifically, the location information for any word pair in the nextword index is a combination of a document number and an offset. For example, a typical vector for the word-pair "red dog" may look as follows:

$$3[1(1:12),\ 34(3:23,34,111),\ 230(4:45,56,345,1025)]$$

The initial integer in an inverted list (in our example, 3) indicates there are three documents containing the word-pair "red dog". Each number shown before an opening bracket signifies an ordinal document number. So, for our example, the three documents containing the word-pair are documents 1, 34, and 230. The numbers inside the brackets for each document give a count of the number of times the word-pair occurs in the document, and the ordinal offsets of the word-pair.

In this example, document 34 contains 3 occurrences of "red dog" at ordinal offsets 23, 34, and 111.

The inverted list structure described above is conventional, and used in most retrieval systems where ordinal offsets are required to support phrase querying [11]. These integers can be compressed with bit-wise coding schemes; in this case we use parameterised Golomb coding [3, 11] for ordinal document numbers and offsets, and unparameterised Elias gamma codes [2, 11] for the small counters. Since the offsets are guaranteed to be monotonically increasing, differences can be taken prior to compression, to reduce the magnitude of the numbers to be compressed. Use of such compression not only reduces space requirements but also reduces typical query response times, as the cost of decoding is offset by the reduction in disk transfer costs.

The schemes we propose in the next section do not affect the applicability of variable-bit compression schemes. As we show later, by applying the same compression approach to alternative offset schemes, further space savings are possible.

3 Alternative Offset Schemes

The general aim of any index structure is to improve retrieval speed by storing index information in a highly specialized additional structure. Space requirements are to be minimized without penalizing retrieval speed. These space requirements are often met through specialized compression algorithms [2, 3, 11], or other forms of optimisation that can be drawn from the indexed information, or from the index structure itself.

Rather than develop new compression schemes, we have deveveloped new offset schemes that may allow better compression. Our investigation is based on the observation that phrase queries require only the knowledge of relative word positions, rather than absolute word position; a document is a match if it contains the query phrase, no matter where in the document the sequence actually occurred. Alternative offset schemes can be derived by abstracting aspects of the basic nextword index: offsets, indicators, and matching function. The two alternative schemes we develop have in common that they use smaller integers, and thus lead to better compression of indexes.

Compound indicators

With ordinal offsets and elementary indicators, nextword indexes, even compressed, are large—typically around 50% of the size of the indexed text [10]. (We give experimental results on new data later.) The main aim of this research was to explore whether index size could be reduced with alternative offset and indicator schemes. The unifying theme of these explorations was that schemes must allow correct identification of answers via the index (with no false matches or missed hits), but that it is not necessary to identify where in the document the phrase is found: thus it may be possible to evaluate queries with less information.

One scheme we explored is based on counting distinct occurrences of terms rather than the total numbers of terms.

Text	the	first	time	the	red	dog	saw	the	red	cat
Offset	1	1	1	2	1	1	1	2	2	1

Figure 3: *A sample document with w-distinct offsets.*

Definition 3.1 The offsets $o_1, \ldots, o_n$ in a document $d = w_1 \ldots w_n$ are *w-distinct* if, for $1 \leq i \leq n$, the offset $o_i = c$ for the cth occurrence of word w_i in d. $\square$

The example document is shown in Figure 3, with w-distinct offsets. As can be seen, w-distinct offsets do not directly allow identification of phrases. The knowledge that the pair $t_i \cdot t_{i+1}$ has elementary indicator $o_i = c$ gives no information about the indicator to expect for the pair $t_{i+1} \cdot t_{i+2}$. In the example, most of the words have offset 1 and the remainder all have offset 2.

However, compound indicators address this problem.

Definition 3.2 An indicator i for pair $p \cdot s$ at position k in document d is *compound* if $i = o_k \cdot o_{k+1}$ is a pair of integers. $\square$

If the pair $t_i \cdot t_{i+1}$ has compound indicator $o_i \cdot o_{i+1}$, then the indicator for the pair $t_{i+1} \cdot t_{i+2}$ must be $o_{i+1} \cdot x$ for some integer x; in the document there is only one pair beginning with word t_{i+1} at offset o_{i+1}. In the example in Figure 3, the first occurrence of the phrase "the red" has the compound indicator $2 \cdot 1$ while the second has $2 \cdot 2$.

All phrase queries can be resolved unambiguously using w-distinct offsets and compound indicators, but, in contrast to ordinal offsets, the query must be processed left to right and all word pairs must be accessed. Thus queries on longer phrases may not be as efficient.

Definition 3.3 *Query evaluation algorithm (compound).* Algorithm 3.3 is identical to algorithm 2.5, except that step 2(c)ii is replaced with

2(c)ii For each indicator $b' = o_k \cdot o_{k+1} \in B_l$ where $b = o_{k-1} \cdot o_k \in B_L$, add b' to $B_{l'}$. $\square$

Theorem 3.1 Given a collection D of documents d with w-distinct offsets, an compound nextword index, a query $q = t_1 \ldots t_m$, and a set $\mathcal{D}$ of documents returned by algorithm 2.5, then

1. (Soundness.) For each document $d \in \mathcal{D}$, the document d is an answer to q.

2. (Completeness.) For each document $d \in D$ that is an answer to q, document $d \in \mathcal{D}$.

Proof Straightforward. $\square$

We can then use the nextword list and the compound offset of the next word pair to determine whether this word pair followed the first word pair or not. In this scheme, a vector contains compound indicators. As discussed above, for a vector of elementary indicators, differences can be taken prior to compression to reduce the size of the numbers to be coded, and thus reducing the number of bits required per number. For compound indicators, both of the offsets in the indicator are monotonically increasing between occurrences of the same pair in the same document, and thus both of the offsets in an indicator can be differenced prior to compression. Once differences have been taken, the offsets can be compressed with Elias or Golomb codes as before. Experimental results with this scheme are reported in this next section.

Text	the	first	time	the	red	dog	saw	the	red	cat
Offset	1	1	2	3	3	3	4	5	5	6

Figure 4: *A sample document with p-distinct offsets.*

Offsets based on repeated words

Phrase query evaluation does not implicitly require that elementary offsets be unique, only that they allow unique identification of phrases. Indeed, to follow this thought-experiment further, in some documents offsets are not required at all. Suppose that, in a particular document d, no words are repeated. Then the knowledge that both the pair $w_1 \cdot w_2$ and the pair $w_2 \cdot w_3$ occur in d is sufficient to identify that d contains the phrase $w_1w_2w_3$. Generalising this idea, offsets that allow us to distinguish between repeat occurrences of the same word are sufficient for phrase query evaluation. Phrase query matching requires us usually to match the position of different words in a particular sequence. This means that we need a scheme and matching function that will signify the adjacency of word pairs containing different words, and not just word pairs containing the same words.

Compound indicators based on w-distinct offsets achieve this aim; however, with other offset schemes, elementary indicators are sufficient. An offset scheme that has this property is as follows: use an offset counter that increases only if a word repeats, and only if the counter has not been increased in the meanwhile. To eliminate the possibility of false matches, the counter must be increased by 2 at each repetition, in two steps. This will give an offset distance of two (or greater), while the matching distance remains at equal or plus one, as before. We must increase the offset before the reoccurrence of a word by one, and increase the offset of the reoccurring word itself by one. The offset increase is only required if the distance between the two reoccurring words is not already greater than 1.

Definition 3.4 The offsets $o_1, \ldots, o_n$ in a document $d = w_1 \ldots w_n$ are *p-distinct* if, for $1 \leq i \leq n$,

- The first offset $o_1 = 1$ (notionally, $o_0 = 0$);
- If w_i is identical to an earlier word w_j (and hence $i > 1$), and $o_j = o_{i-2}$, then $o_{i-1} = o_{i-2} + 1$ and $o_i = o_{i-1} + 1$;
- Otherwise $o_{i-2} = o_{i-1}$;
- The final offset $o_n = o_{n-1} + 1$. □

An example of p-distinct offsets is shown in Figure 4. The repetition of the word "the" triggers an increment in the offsets, so that the occurrence of "the" following "time" can be distinguished from the first "the". Another increment is triggered by the later "the". The repetition of "red" does not trigger an increment, because it is already distinguished from the first occurrence.

P-distinct offsets are sufficient to allow unambiguous left-to-right evaluation of queries, but are smaller than ordinal offsets, thus allowing savings in storage requirements, as explored in our experiments. At first sight it might be concluded that the magnitude of offsets can be further reduced. With p-distinct offsets, at each word repetition, the magnitude of the offset is increased by 2, in two steps. However, if the offset is only increased by 1, query evaluation gives rise to false matches. Consider for example the document

the dog and all the

With only a single increment, this document would have the offset sequence "1 1 1 2 2", in which "and all" has the indicator 1 and "all the" has the indicator 2. For the document

the dog and all by all the

the offset sequence is "1 1 1 1 2 2 2", in which, again, "and all" has the indicator 1 and "all the" has the indicator 2. But the second document does not include "and all the", and so would be a false match.

With p-distinct offsets, then, we have that $o_{k-1} \leq o_k \leq o_{k-1} + 1$. This condition—in conjunction with the information that the same term is involved in both pairs—is sufficient to correctly identify documents that are answers.

Theorem 3.2 Given a collection D of documents d that have p-distinct offsets, an elementary nextword index, a query $q = t_1 \ldots t_m$, and a set $\mathcal{D}$ of documents returned by algorithm 2.5, then

1. (Soundness.) For each document $d \in \mathcal{D}$, the document d is an answer to q.

2. (Completeness.) For each document $d \in D$ that is an answer to q, document $d \in \mathcal{D}$.

Proof Matches that are found with ordinal offsets will clearly be found with p-distinct offsets, and thus statement 2 follows from Theorem 2.1.

Proof of statement 1 is more complex. We proceed by induction, and assume that in d there is a phrase $t_1 \ldots t_i$ such that t_{i-1} has ordinal o_k. The current pair is $t_i \cdot t_{i+1}$, with indicator b_i. We need to establish that, if $o_{k-1} \leq b_i \leq o_{k-1} + 1$, then d contains the phrase $t_1 \ldots t_{i+1}$.

There are five distinct cases to consider. Let f, g, and h be any distinct words. The sequence $t_{i-1}t_it_{i+1}$ from the query is either fff, ffg, fgf, gff, or fgh. By analysis of these five cases, it is straightforward to show that the condition $o_{k-1} \leq b_i \leq o_{k-1} + 1$ only holds if the same sequence occurs in both query and document.

Consider for example the sequence fgf. There can be a false match only if the document contains both fg and gf, but not as part of fgf. Now fg has offset o_{k-1} and gf has indicator b_i. If gf precedes fg, then $b_i < o_{k-1}$ and the condition fails. If gf follows fg, with intervening words, then $o_{k-1} + 2 \leq b_i$ and again the condition fails. The other four cases are similar. Thus the theorem holds. $\square$

A further small saving is available at coding time. In an inverted list for a pair $t_1 \cdot t_2$ in which t_1 and t_2 are different (as they almost always are), with p-distinct offsets it is guaranteed that, when the pair occurs more than once in the same document, the offset will always differ by at least 2. Thus the difference in the offsets can be decremented by 1 prior to coding, further reducing the magnitude of the numbers to be stored. This optimisation is not used in our experiments because the saving is small and additional work is required during query evaluation.

Table 1: *Index size for the ordinal, compound, and p-distinct offset schemes.*

	Index size (Gb)		% of indexed data		% of ordinal vectors	
	Web	WSJ	Web	WSJ	Web	WSJ
Ordinal	4.62	0.25	56.0%	51.4%	100.0%	100.0%
Compound	4.04	0.21	49.0%	43.6%	83.9%	80.6%
P-distinct	4.25	0.22	51.4%	46.7%	89.6%	88.1%

4 Experimental Results

All experiments were run on an Intel 750 MHz Pentium III-based server with 512 Mb of memory, running the Linux operating system. For all timings, the disk and CPU cache were flushed prior to running the retrieval system for the first query with each scheme, and the results are an average over all queries for each scheme. The machine was under light load, that is, no other significant tasks processes were accessing memory or disk.

Queries were drawn from a query set of over 1.7 million queries as logged on one day in 1999 by the Excite search engine [9]. We preprocessed these queries with the filtering scripts used to remove offensive and undesirable documents from the TREC Very Large Collection Web documents [5]. TREC is an ongoing international text retrieval experiment sponsored by ARPA and NIST [4]. From the filtered queries, we selected all phrase queries, that is, those queries that were surrounded by quotation marks, giving a set of 33,286. The average query length is 2.55 with 21,346 queries of two words in length and only 1,307 that are longer than four words; the longest query had sixteen words.

We used two collections in our experiments. The first was drawn from the TREC Very Large Collection of Web documents. In all, we extracted 21.9 Gb of HTML documents from the collection and, when indexing, ignored any HTML markup, documents that appeared to be non-textual, and other markup. The overall result was that from the 21.9 Gb we indexed around 8.3 Gb of textual data. We refer to this collection as WEB.

The second collection was 491 Mb of data from The Wall Street Journal newswires extracted from TREC disks 1 and 2. As for the WEB collection, we did not index the markup used to identify document boundaries and so on. We refer to this collection as WSJ.

Index size

Table 1 shows the index size achieved with each offset scheme. In the first case, "ordinal", we store ordinal offsets with elementary indicators; in the second case, "compound", we store w-distinct offsets with compound indicators; in the third case, "p-distinct", we store p-distinct offsets with elementary indicators. The "percentage of indexed data" size is against only the part of the data that is actually indexed; that is, tags and so on are excluded. As these results show, the index size is not a constant proportion of data size. The WSJ collection is relatively controlled, and, compared to the web data, specific in topic. The WEB data contains more typographic errors and is more

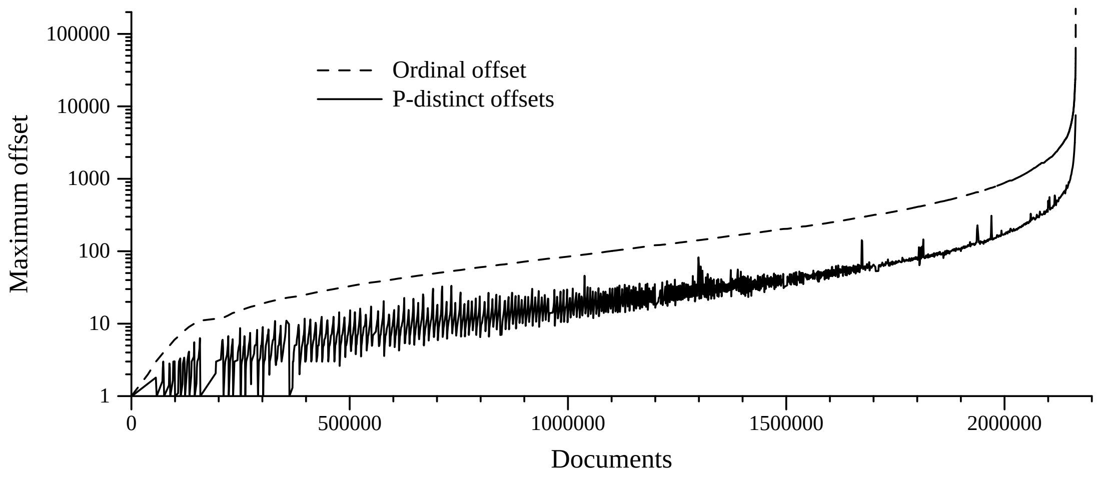

Figure 5: *Maximum offset stored for each* WEB *document using the ordinal and p-distinct offset schemes. Documents are shown in sorted order from shortest to longest document. Maximum values shown for the p-distinct scheme are averages over blocks of documents.*

diverse, leading to larger numbers of distinct word pairs. Thus indexes for WEB are, relatively, larger than for WSJ—a result that is somewhat counterintuitive, since the larger collection might be expected to lead to economies of scale, due to decline in growth in the number of distinct word pairs.

In both collections the new offset schemes lead to significant space savings. The compound method is particularly effective. Total index size falls by 7% of data set size in both data sets. The last two columns show the savings for the vectors alone, the only part of the index affected by the new offset scheme; for WSJ, the compound method leads to savings of almost 20% of vector file size. The use of p-distinct offsets does not yield such large reductions, but the savings are still significant, of just over 4.5% of data set size for both WSJ and WEB.

Figure 5 is an illustration of how the space savings have been achieved. In this graph, the documents are ordered across the x-axis by the number of words they contain, that is, the value of the maximum ordinal offset. This value is plotted on the upper, smooth line. The lower, jagged line shows the corresponding value of the maximum p-distinct offset for each document. The p-distinct offsets are, as can be seen, typically one-quarter to one-eighth the size of the ordinal offsets; with an Elias gamma code, this reduction corresponds to a typical saving of 4 to 6 bits per encoded offset, or slightly less reduction with a Golomb code.

Query processing time

Table 2 shows query processing time for ordinal and p-distinct offsets. Two factors lead to different times: first, with p-distinct offsets vectors are smaller, leading to reductions in disk access time; second, with ordinal offsets, the pairs can be processed in a more efficient order, allowing some pairs from a query to be omitted altogether and earlier termination if there are no matches. Compound indicators share with p-distinct offsets the limitation that queries must be processed linearly and both have similar processing time; the additional time is due to greater decoding complexity.

Table 2: *Average query times for the ordinal and p-distinct offset schemes for all queries, resolved queries (those with one or more answers), and unresolved queries (those where no answer was found in the collection). All results are on the Web collection.*

	Ordinal (seconds)	Compound (seconds)	P-distinct (seconds)
Overall	0.082	0.128	0.104
Resolved	0.106	0.145	0.116
Unresolved	0.044	0.103	0.086

As can be seen, ordinal offsets allow faster processing, on average in around 80% of the time of the p-distinct scheme and 65% of the compound scheme. For the p-distinct scheme, most of this difference is on queries that cannot be resolved, that is, do not have matching documents in the database; on queries that do have matches, the difference is smaller. In the context of the large gains in efficiency available through nextwords—in our earlier work [10], we showed that short phrase queries on WSJ could be resolved about five times faster with a nextword index than with a conventional inverted index of document identifiers and word positions—the loss due to use of p-distinct or compound indicators must be regarded as small.

We suspected that the major cause of difference in evaluation times was due to the inability to re-order query processing with the new schemes, so that it is likely that the difference in times grows with query length. For this reason we timed the queries of different lengths separately, as reported in Table 3. These results show that for queries of 2 or 3 words—around 95% of our query set—the new techniques are virtually indistinguishable from use of ordinal offsets. However, for long queries the differences become more pronounced. For phrases of 5 words, the new methods are twice as slow as ordinals; over the 11 phrases of 11 words or more, the new methods again are almost twice as slow. This is because, in the longer queries, use of ordinals means that only every second pair is needed and unsuccessful queries can be terminated earlier. Addressing this issue is a subject of our current research.

5 Conclusions

A nextword index has previously been proposed as an efficient data structure for resolution of phrase queries on large text databases. In this paper we have outlined two novel techniques for reducing the space required to store the vectors in a nextword index. Both techniques are a form of compaction: they rely on the fact that absolute word positions are not required for unambiguous resolution of phrase queries, but that it is sufficient to store integers that distinguish between different occurrences of the same word.

Our experimental results show that significant space savings are available with both of the techniques. In particular, the simple strategy of labelling words by their count of occurrence within a document, then labelling a pair by the matching pair of occurrence counts (the compound scheme) can allow reductions in the size of the compressed vectors of nearly 20%. However, with

Table 3: *Average query times of the ordinal and p-distinct offset schemes for each query length in the Excite query log. All results are on the Web collection.*

Query Length	Number of queries	Ordinal (seconds)	Compound (seconds)	P-distinct (seconds)
16	2	0.042	0.594	0.533
15	1	0.001	0.019	0.023
14	4	0.072	0.579	0.503
13	1	2.535	4.154	3.410
12	3	1.219	2.489	1.800
11	4	0.074	1.989	1.349
10	14	0.070	2.055	1.425
9	19	1.143	2.171	1.468
8	33	0.422	1.595	1.127
7	141	0.497	1.266	0.913
6	289	0.306	0.981	0.732
5	796	0.234	0.696	0.532
4	2687	0.347	0.559	0.411
3	7946	0.120	0.139	0.126
2	21346	0.021	0.024	0.023

these strategies it is not possible to evaluate the pairs in queries in the most efficient order, and there is consequently an increase in query processing time. The increase is acute for long queries, but insignificant for the short queries that consistitute the large majority of phrases posed to search engines.

We are investigating further improvements to nextword indexing. We have identified possible new ways of optimising query processing that can be applied with the compound and p-distinct representations. Further space savings are also possible; to take a trivial example, if a word is unique in a document, an offset is not required at all. Moreover, providing an incomplete nextword index that stores only the class of phrases that are commonly queried and using this index in conjunction with conventional structures may further reduce query times and storage costs. Even without these improvements, however, nextword indexes are an efficient and practical way of evaluating phrase queries, and are significantly improved by the new representations investigated in this paper.

References

[1] D. Bahle, H.E. Williams, and J. Zobel. Optimising phrase querying and browsing of large text databases. In M. Oudshoorn, editor, *Proc. Australasian Computer Science Conference*, volume 23, pages 11–19, Gold Coast, Australia, 2001.

[2] P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, IT-21(2):194–203, March 1975.

[3] S.W. Golomb. Run-length encodings. *IEEE Transactions on Information Theory*, IT–12(3):399–401, July 1966.

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

[5] D. Hawking, N. Creswell, and P. Thistlewaite. Overview of TREC-7 very large collection track. In E. Voorhees and D.K. Harman, editors, *Proc. Text Retrieval Conference (TREC)*, pages 91–104, Washington, 1999. National Institute of Standards and Technology Special Publication 500-242.

[6] C.G. Nevill-Manning and I.H. Witten. Compression and explanation using hierarchical grammars. *Computer Journal*, 40(2/3):103–116, 1997.

[7] C.G. Nevill-Manning, I.H. Witten, and G.W. Paynter. Browsing in digital libraries: A phrase-based approach. In *Proceedings of the 2nd ACM International Conference on Digital Libraries*, pages 230–236, Philadelphia, PA, 1997. ACM Press, New York.

[8] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983.

[9] 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.

[10] H.E. Williams, J. Zobel, and P. Anderson. What's next? Index structures for efficient phrase querying. In John Roddick, editor, *Proc. Australasian Database Conference*, pages 141–152, Auckland, New Zealand, January 1999.

[11] I.H. Witten, A. Moffat, and T.C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, second edition, 1999.