

# Efficient Phrase Querying with Common Phrase Index<sup>\*</sup>

Matthew Chang and Chung Keung Poon

Dept. of Computer Science, City U. of Hong Kong, China  
{kcmchang, ckpoon}@cs.cityu.edu.hk

**Abstract.** In this paper, we propose a *common phrase index* as an efficient index structure to support phrase queries in a very large text database. Our structure is an extension of previous index structures for phrases and achieves better query efficiency with negligible extra storage cost. In our experimental evaluation, a common phrase index has 5% and 20% improvement in query time for the overall and large queries (queries of long phrases) respectively over an *auxiliary nextword index*. Moreover, it uses only 1% extra storage cost. Compared with an *inverted index*, our improvement is 40% and 72% for the overall and large queries respectively.

## 1 Introduction

In this information age, search engine acts as an efficient tool for seeking information from a vast heap of online texts. By providing an ad hoc query, we can immediately get a set of texts from gigabytes text database. As the Internet is growing at an extremely fast pace, the number of hosts available increased more than 130 folds (from 1.3 to 171 millions<sup>1</sup>) in the last decade. Hence, search engines should be able to evaluate queries efficiently and effectively. In other words, the systems should resolve queries quickly and also provide accurately what the users want [13]. In order to improve the effectiveness of searching, considering phrases in searching and indexing seems to be an interesting idea for the following reasons:

- Phrases come closer than individual words or their stems to express structured concepts.
- Phrases have a smaller degree of ambiguity than their constituent words. That is, while two words are both ambiguous, the combination is not, since each of its two constituent words creates a context for the unambiguous interpretation of the other.
- By using phrases as index terms, a document that contains a phrase would be ranked higher than a document that just contains its constituent words in unrelated contexts.

---

<sup>\*</sup> The work described in this paper was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 1198/03E].

<sup>1</sup> Source: Internet Software Consortium (<http://www.isc.org/>).

Previous research works on phrase recognition and automatic phrase formation [6, 7, 8] resulted in improved retrieval effectiveness. Also, an amalgamated hierarchical browsing and hierarchical thesaurus browsing can be used to display the result set of documents in a more effective way than simply listing the results [11].

To efficiently resolve a query with gigabytes of online texts, an efficient way of indexing is essential. A conventional and practical way of indexing is the *inverted index* [17]. For each term in the index, there are a list of postings with document identification numbers, within-document frequencies and offsets at which the term appears. To resolve a query, we can simply combine the lists attached to each of the query terms to get the results. It is straightforward to intersect the lists with the consideration of the offsets of the terms to produce the result set for a phrase query. However, for evaluating common words, the process of merging is slow due to the long list of postings retrieved. Even if we use pruning techniques like *frequency-sorted indexing* and *impact-sorted indexing* so that we need not retrieve the whole postings list of a term [1, 2, 5, 12], the number of retrieved highest-scored postings of a common word can still reach several megabytes. Also, these early termination heuristics do not retain the complete set of result documents.

*Nextword index* provides a fast alternative for resolving phrase queries, phrase browsing, and phrase completion [15]. Unlike an inverted index, it has a list of nextwords and positions following each distinct word. The set of first words is known as the *vocabulary* or *firstword*. The words following the firstwords are called *nextwords*. The major drawback of using *nextword index* is its large space consumption which is around 60% of the size of the indexed data. With careful optimization techniques [3], the size of a nextword index file can be reduced to 49% of the indexed data. An *auxiliary nextword index* proposed by Bahle et. al [4] further reduces the space overhead to only 10% of the size of the inverted index file.

In this paper, we propose a new indexing structure which we call *common phrase index*. Building on the ideas of the inverted index and auxiliary nextword index, it divides the vocabulary into two sets: *rare words* and *common words*. We attach a list of postings to each of the rare words as in an inverted index. For the common words, it differs from the nextword index structure in that it has a collection of trees such that each root-to-leaf path represents a phrase starting from a common word and ending in a terminal word. Moreover, postings lists are only attached to the leaf nodes of the tree. This combination of structures breaks down the original inverted lists of terms and provides useful additional information for evaluating phrases efficiently from a large text database. Similar to our index, Moffat and Zobel [10] proposed that inverted lists can be broken into groups by introducing *synchronization points*. However, our index applied the concept of phrase and hence has the following advantages over it. First, it supports fast phrase query evaluation. Also, it breaks only some inverted lists but provides significant improvement in efficiency. Last, it is not required to determine the number of document pointers in a group.

In our experiments, we implemented a prototype system to compare an inverted index, auxiliary nextword index, and common phrase index against a set of benchmark documents and query logs. Our results show that common phrase index speeds up the overall efficiency and large-sized query evaluation by 5% and 20% respectively. Both the inverted index and nextword index have a significant increase in the query time as the size of query increases. In contrast, common phrase index shows only a slight increase in query time as the query size increases. Also, the storage usage is just increased by about 1% of the auxiliary nextword index.

The rest of this paper is organized as follows. Section 2 and 3 describe the inverted index, nextword index, and auxiliary nextword index. Section 4 introduces our common phrase index and its interesting characteristics. Section 5 explains our experiment setup and presents our results. Section 6 is our conclusion and future work.

## 2 Inverted Indexes

*Inverted index*, or inverted file structure, is the most commonly used index structure for database management and information retrieval systems [9]. An inverted index is a two-level structure. The upper level is all the index terms for the collection. For text database, 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 [16], each posting is a triple of the form:

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

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. To evaluate a query, each query term is used to fetch a postings list via the vocabulary search. For instance, suppose we have the following four documents associated with their contents:

```
Document 1 {Computer Science}
Document 2 {Computer Engineering}
Document 3 {Search Engine}
Document 4 {Computer Science: Search Engine}.
```

Then, the postings lists of “computer” and “science” are:

```
<computer,    3, <1, 1, [1]>, <2, 1, [1]>, <4, 1, [1]>>
<science,     2, <1, 1, [2]>, <4, 1, [2]>>.
```

It indicates that document 1, 2, and 4 have one occurrence of “computer” each at the position 1 of the documents. The term “science” exists at position 2 of both document 1 and 4. To resolve a phrase query “Computer Science”, we first retrieve the corresponding postings list of each query term and then we

sort the lists according to the value  $f_t$  in ascending order. Finally we intersect the lists one by one from the rarest to the most common term with a temporary structure. When intersecting the sorted lists, we have to consider the proximity. The query terms “Computer” and “Science” have a position difference of one. Hence, for each intersect operation, we first see if there is any document identifier present in both structures. For each matched document, we check if the offsets of them are having a position difference of one. So, the results of the phrase query “Computer Science” are document 1 and 4.

### 3 Nextword Indexes and Auxiliary Nextword Indexes

Inverted index is not efficient for evaluating query with common terms since the three most common words account for about 4% of the size of the whole index file [4] and retrieving such long postings list can suffer a long operation time. Hence, nextword index [15] is proposed to construct an index by recording additional index for supporting fast evaluation of phrase queries.

A nextword index is a three-level structure. The highest level is of the distinct index terms in the collection, which we call *firstwords*. At the middle level, for each firstword there is a collection of *nextwords*, which are the words observed to follow that firstword in the indexed text. At the lowest level, for each nextword there is a postings list of the positions at which that firstword-nextword pair occur. Using the same example we employed in Section 2, the postings lists of all firstword-nextword pairs of the nextword index are shown below:

```
<computer_science,      2, <1, 1, [1]>, <4, 1, [1]>>
<computer_engineering,  1, <2, 1, [1]>>
<science_search,       1, <4, 1, [2]>>
<search_engine,        2, <3, 1, [1]>, <4, 1, [3]>>.
```

To pose a phrase query “Computer Science”, nextword index just needs to fetch a single list of postings instead of retrieving two long lists of postings and performing intersection. This speeds up the evaluation of a phrase query. The applications of nextword index can be found in [15]. However, the size of index is large. Bahle et. al [4] observe the weakness of resolving phrase query by using an inverted index and the enormous size overhead of a nextword index and hence proposed auxiliary nextword index. The main idea of the auxiliary nextword index is that only the *top-frequency* words are to be indexed with nextwords. For example, using the same sample documents as Section 2 and assuming that “computer” is the only high-frequency or common word, all firstword-nextword pairs of the auxiliary nextword index are as:

```
<computer_science,      2, <1, 1, [1]>, <4, 1, [1]>>
<computer_engineering,  1, <2, 1, [1]>>.
```

To resolve a query, the steps are similar to an inverted index. However, the auxiliary nextword index first contends with the common words of query terms, and then the rest. Experimental result in [4] shows that having the three most common terms as firstwords consumes just 10% of space of the inverted index. Therefore, a huge amount of space is saved compared with nextword index. The ideal size of phrase query for an auxiliary nextword index is two because only one fetching is required. However, in the query log of Excite dating 1997 and 1999, queries of size two occupied only 35.28%. For queries of size one, which accounts for 25.36%, no index can be more efficient than an inverted index. For the remaining 39.36% of queries, a nextword index has to perform at least two fetchings and one intersection. We will show that a common phrase index can further improve the efficiency for query size larger than two.

## 4 Common Phrase Indexes

The main difference between a common phrase index and an auxiliary nextword index is that the additional index terms are not fix-sized *firstword-nextword* pairs but variable-sized *common phrases*. For each common phrase, there is a postings list of the positions at which that common phrase occur.

### 4.1 Common Phrases

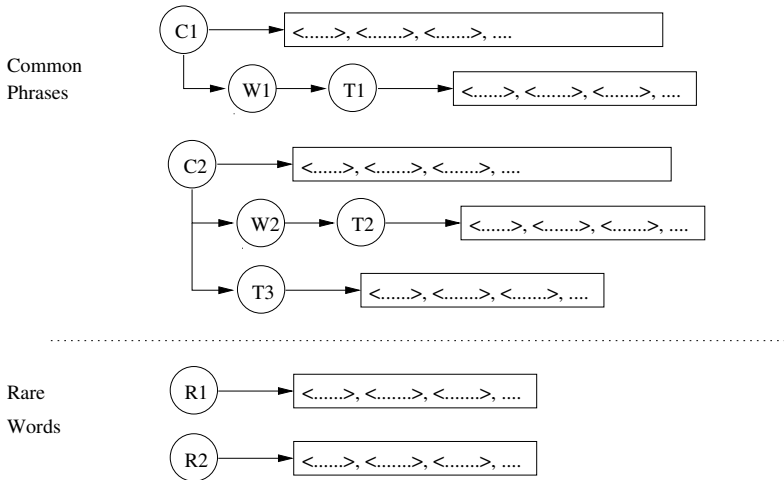
We define *common phrase* as a sequence of two or more contiguous words that starts with a *common word* and ends in a *terminal word*.

We define common words as those words having the highest frequencies in a set of queries. Thus our notion of common word is sensitive to the query workload. For concreteness, we take the Excite query log dating 1997 and 1999 as reference. We first count the frequency of each distinct word of the query logs. The highest-frequency words are known as *common words* and the others are called *rare words*.

Terminal words are words which are likely to be the end of a phrase. We observe that in an auxiliary nextword index, some of the indexed pairs (e.g. “in\_all”, “in\_new”, “in\_the”) can actually be expanded in order to achieve further efficiency improvement. These *firstword-nextword* pairs do not end in a significant or *terminal* word (to be defined). We have investigated the 2.1 million queries in the query logs of the Excite dating 1997 and 1999. In the logs, we count the frequency of each function (or part-of-speech) of the last word of all queries by submitting the words to *Merriam-Webster OnLine* and retrieving the corresponding functions, see Table 1. If a word has more than one function, we take the most popular function of the word. Note that, the function *Others* includes misspellings and proper nouns. In the table, we observe that prepositions, adverbs, conjunctions, definite articles, indefinite articles, and pronouns occupy only 0.408% of all last words of the queries. That is, a query often contains a phrase with last word having other kinds of function. Thus, we define the *terminal word* of our common phrase as *any word whose function*

**Table 1.** Function Frequency of Query Last Word

Function	Frequency	Percentage
adjective	62482	2.938%
preposition	1556	0.073%
adverb	5116	0.241%
conjunction	209	0.010%
definite article	79	0.004%
indefinite article	15	0.001%
verb	126252	5.936%
noun	1050804	49.410%
pronoun	1686	0.079%
others	878512	41.308%



**Fig. 1.** Common Phrase Index Structure

is not a preposition, adverb, conjunction, definite article, indefinite article, or pronoun.

The structure of common phrase index is illustrated in Figure 1, where the *C*'s are common words, *R*'s are rare words, *T*'s are terminal words, and *W*'s are the others. The concept of common phrase index can be easily illustrated by using a simple example to compare the structure of auxiliary nextword index and common phrase index. For instance, suppose we have the following documents:

- Document 1 {Students of the same year}
- Document 2 {Computer and applications}
- Document 3 {Usage of the Search Engine}.

For the sake of brevity, we omit all the postings lists of the rare words. Then, the auxiliary nextword index for the documents contains:

```

<and_applications,      1, <2, 1, [2]>>
<computer_and,         1, <2, 1, [1]>>
<of_the,               2, <1, 1, [2]>, <3, 1, [2]>>
<the_same,             1, <1, 1, [3]>>
<the_search,          1, <3, 1, [3]>>

```

while the common phrase index for the documents contains:

```

<and_applications,      1, <2, 1, [2]>>
<computer_and_applications, 1, <2, 1, [1]>>
<of_the_same_year,     1, <1, 1, [2]>>
<of_the_search,        1, <3, 1, [2]>>
<the_same_year,        1, <1, 1, [3]>>
<the_search,           1, <3, 1, [3]>>.

```

The number of contiguous words in an auxiliary nextword index is capped at two while that in a common phrase index is not limited. The size of a common phrase stops growing when it encounters a terminal word. In the above example, the postings list of the *firstword-nextword* pair “of\_the” in the nextword index is broken down into “of\_the\_same\_year” and “of\_the\_search” in the common phrase index. By issuing a query “computer and applications”, auxiliary nextword index has to perform two fetchings and one intersection while the common phrase index has to perform just a single fetching.

Note that by using common phrase index, high efficiency in phrase evaluation can be sustained even if the query size is large. Our experiments in Section 5 show that improvement of common phrase index enlarges when the query size increases. The further breakdown of the postings lists in common phrase index supports high efficiency with low additional storage overhead comparing with auxiliary nextword index.

## 4.2 Query Evaluation

To evaluate a query with common phrase index, we perform the following steps:

1. Identify the first common word of the query.
2. Expand the common word to a common phrase by adding the succeeding words of it until a terminal word or end of query is reached. If the common phrase is found, fetch the postings list to a temporary structure. Then mark all the words of the common phrase as *covered*.
3. Find the next common word from the query terms that are not yet *covered*.
4. Repeat *Step 2* and intersect the fetched postings list with the temporary structure until all common words are *covered*.
5. For all the words not yet *covered*, which must be rare words, fetch the associated postings list and intersect with the postings list in the temporary structure.

## 4.3 Dynamic Nature of Common and Rare Words

Our experiments use a set of common words that is “ideal” for a certain set of queries. In reality, the set of highest-frequency words in query log keeps changing.

We have studied a report that analysed an Excite query log of 16 September 1997 [14] which listed the top 75 terms that are occurring more than 1110 times in the 531,416 unique queries. The top 8 terms with more than 5000 times each in unique queries are the words likely to have high-frequency at anytime, they are “and”, “of”, “sex”, “free”, “the”, “nude”, “pictures”, and “in”. These terms have 80778 or 15.2% of the total occurrences of the top 75 terms. However, some terms ranked in the top 75 are not that stable especially for those having less than 2000 times each in unique queries. They are popular for querying at a certain period of time because of some incidences. For example, the terms “princess” and “diana” which are having 1461 and 1885 times respectively in 1997 because many people are in memory of Diana, Princess of Wales. The high-frequency word “diana” is ranked at 31<sup>st</sup> in 1997 but it falls to 315<sup>th</sup> when we take both 1997 and 1999 Excite query log into consideration. The term “princess” is ranked from 45<sup>th</sup> in 1997 and dropped to 362<sup>nd</sup> in both 1997 and 1999.

Our observation suggests that common phrase index should be dynamically changing according to the high-frequency terms at that time. Basically, common phrase index is a combination of an inverted index and additional indexes of common phrases. By using this combination, it is easy for updating the index file. The system has to keep the frequency of each query term. For each word newly falling into the set of common words, we break down the corresponding postings list into several lists indexed by common phrases. In contrast, for each word that is newly excluded from the set of common words, we can union the lists of common phrases into one list.

## 5 Experiments

In our experiment, we implemented a prototype system to evaluate phrase queries for comparing the efficiency and total size of inverted index, auxiliary nextword index, and common phrase index. We used a database system, *MySQL*, to store all the postings lists, vocabularies, and dictionary. All experiments were run on an Intel 3 GHz Pentium 4-based server with 2 Gb of memory, running *Linux* operating system under light load.

We used a set of benchmark documents and query logs in our experiments. For test documents, we used the *.Gov* web research collection from TREC. This collection is especially for Information Retrieval systems in a Web context and large-scale information retrieval systems design and evaluation. It was collected in early 2002 and contains about 1.25 million *.gov* web pages with a total size of about 18.1 Gbytes. For test queries, we used the query logs provided by Excite dating to 1997 and 1999. These are the same query logs used by Bahle et. al [4]. There are 2.1 million queries in the query logs (including duplicates).

We did the following preprocessing to each document. First of all, any formatting information such as HTML or XML tags are removed from the document. Special characters are replaced by blanks and all upper-case letters are changed to lower-case. After these, a document becomes a sequence of words separated by blanks. We then constructed the postings lists for each kind of indexing and stored them in database.

In our experiments, we first focus on comparison in efficiency between auxiliary nextword index and common phrase index. We tried designating different number of words as common words in the vocabulary for both indexes, see Table 2. For each size of the set of common words, we show the overall efficiency and also break down the results according to the query size in order to pinpoint where improvements are made. For all the different sizes of the set of common words we have tested, we have about 4% improvement in the overall efficiency. For queries of size one and two, the common phrase index does not have any

**Table 2.** Efficiency in average milliseconds of Auxiliary Nextword Index (ANI) and Common Phrase Index (CPI) with 10, 20, and 255 common words. ( $\Delta = ANI - CPI$ )

Query Size	ANI(10)	CPI(10)	$\frac{\Delta}{ANI(10)}$	ANI(20)	CPI(20)	$\frac{\Delta}{ANI(20)}$	ANI(255)	CPI(255)	$\frac{\Delta}{ANI(255)}$
Overall	367.47	352.30	4.13%	359.34	341.72	4.90%	247.77	237.55	4.12%
1	79.99	79.98	0.01%	80.05	79.66	0.49%	77.36	76.88	0.62%
2	302.49	302.42	0.02%	300.69	299.60	0.36%	243.04	241.44	0.66%
3	452.96	448.21	1.04%	445.90	438.91	1.57%	322.16	311.65	3.26%
4	643.85	619.25	3.82%	625.79	596.87	4.62%	383.80	362.47	5.56%
5	708.75	642.19	9.39%	680.87	608.02	10.70%	375.01	335.58	10.52%
$\geq 6$	980.00	804.69	17.89%	921.68	729.56	20.85%	397.77	331.88	16.56%

**Table 3.** Efficiency in average milliseconds of Inverted Index (II), Auxiliary Nextword Index (ANI) and Common Phrase Index (CPI) with 255 common words

Query Size	II	ANI(255)	CPI(255)	$\frac{II-ANI(255)}{II}$	$\frac{II-CPI(255)}{II}$
Overall	401.80	247.77	237.55	38.33%	40.88%
1	79.97	77.36	76.88	3.26%	3.86%
2	307.57	243.04	241.44	20.98%	21.50%
3	488.34	322.16	311.65	34.03%	36.18%
4	701.48	383.80	362.47	45.28%	48.33%
5	857.19	375.01	335.58	56.25%	60.85%
$\geq 6$	1223.12	397.77	331.88	67.48%	72.87%

**Table 4.** Size of Inverted Index (II), Auxiliary Nextword Index (ANI) and Common Phrase Index (CPI) with 10, 20, and 255 common words

Index(Common words)	Index Size (Gb)	$\frac{CPI(x)-ANI(x)}{ANI(x)}$	$\frac{index(x)-II}{II}$
II	23.08	-	-
ANI(10)	25.67	-	11.24%
ANI(20)	26.56	-	15.08%
ANI(255)	26.86	-	16.38%
CPI(10)	25.96	1.13%	12.50%
CPI(20)	26.94	1.44%	16.73%
CPI(255)	27.28	1.59%	18.22%

improvement as we have expected because it can at most extract phrases of the same length as in auxiliary nextword index. Note that, the improvement rate increases when the size of query increases. Common phrase index can even achieve more than 16% improvement when the query size is larger than or equal to six. We also compare the efficiency between the inverted index, auxiliary nextword index, and common phrase index, see Table 3. Again, we observe that common phrase index has a better improvement rate (the rightmost two columns) than auxiliary nextword index.

The total index sizes of different indexings are shown in Table 4. As can be seen, total index size of the common phrase index is just larger than that of the auxiliary nextword index by about 1.5%. It is a negligible trade-off for higher efficiency in query evaluation.

## 6 Conclusion

We have proposed a novel extension of auxiliary nextword index. Phrase queries on large text databases can be supported by using *common phrase index*. In this approach, all words in the text document are indexed the same as inverted index; in addition, the most common words are indexed via common phrase. Unlike the inverted index and auxiliary nextword index where the size of an index term is fixed (one word for an inverted index and two words for an auxiliary nextword index), common phrase index has variable-sized index term. These variable-sized index terms further break down the postings lists and support the fastest phrase query evaluation among inverted index and auxiliary nextword index. Having efficiency improvement especially for query size larger than or equal to three, the total size of it is just larger than that of an auxiliary nextword index by about 1.5%.

Our experimental results show that we can implement common phrase index for evaluating phrase queries with no significant storage overhead. The only additional requirement is having a dictionary for checking of terminal words. However, the dictionary is rather static. Therefore, it can be used for a long period of time after we built it once.

In our future work, we will also implement and experiment a version of our index that can adjust its structure according to the dynamic nature of common words. Since the performance of a common phrase index is highly related to the chosen common words, it requires the index to be updated after a certain period of time. We expect that our system will further improve the efficiency with the dynamic update nature implemented.

## References

1. V. Anh and A. Moffat. Compressed inverted files with reduced decoding overheads. In *Proc. of the 21th Annual SIGIR Conf. on Research and Development in information retrieval*, pages 290–297, 1998.
2. V. Anh and A. Moffat. Vector-space ranking with effective early termination. In *Proc. of the 24th Annual SIGIR Conf. on Research and Development in information retrieval*, pages 35–42, 2001.

3. D. Bahle, H. E. Williams, and J. Zobel. Compaction techniques for nextword indexes. In *Proc. 8th International Symposium on String Processing and Information Retrieval (SPIRE2001)*, pages 33–45, 2001.
4. D. Bahle, H. E. Williams, and J. Zobel. Efficient phrase querying with an auxiliary index. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, 2002.
5. S. Chaudhuri and L. Gravano. Optimizing queries over multimedia repositories. pages 91–102, 1996.
6. W. B. Croft, H. R. Turtle, and D. D. Lewis. The use of phrases and structured queries in information retrieval. In *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Chicago, Illinois, USA, October 13-16, 1991 (Special Issue of the SIGIR Forum)*, pages 32–45, 1991.
7. E. F. de Lima and J. O. Pedersen. Phrase recognition and expansion for short, precision-biased queries based on a query log. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 145–152, 1999.
8. J. L. Fagan. *Experiments in automatic phrase indexing for document retrieval: a comparison of syntactic and non-syntactic methods*. PhD thesis, Cornell University, 1987.
9. G. Kowalski. *Information Retrieval Systems: Theory and Implementation*. Kluwer Academic Publishers, 1997.
10. A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379, 1996.
11. G. W. Paynter, I. H. Witten, S. J. Cunningham, and G. Buchanan. Scalable browsing for large collections: A case study. In *Proceedings of the Fifth ACM International Conference on Digital Libraries*, 2000.
12. M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society of Information Science*, 47(10):749–764, 1996.
13. G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1998.
14. A. Spink, D. Wolfram, B. Jansen, and T. Saracevic. Searching the web: The public and their queries. *Journal of the American Society for Information Science*, 52(3):226–234, 2001.
15. H. E. Williams, J. Zobel, and P. Anderson. What’s next? - index structures for efficient phrase querying. In *Proc. Australasian Database Conference*, pages 141–152, 1999.
16. J. Zobel and A. Moffat. Exploring the similarity space. In *ACM SIGIR Forum*, pages 18–34, 1998.
17. J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23(4):453–490, 1998.