Hybrid Index Organizations for Text Databases

Christos Faloutsos†
University of Maryland
College Park, MD

H. V. Jagadish
AT&T Bell Laboratories
Murray Hill, NJ

ABSTRACT

Due to the skewed nature of the frequency distribution of term occurrence (e.g., Zipf's law) it is unlikely that any single technique for indexing text can do well in all situations. In this paper we propose a hybrid approach to indexing text, and show how it can outperform the traditional inverted B-tree index both in storage overhead, in time to perform a retrieval, and, for dynamic databases, in time for an insertion, both for single term and for multiple term queries. We demonstrate the benefits of our technique on a database of stories from the Associated Press news wire, and we provide formulae and guidelines on how to make optimal choices of the design parameters in real applications.

1. INTRODUCTION

Unformatted databases are rapidly gaining popularity, particularly for the dissemination of text and other "free-format" information. This information is divided into pieces, which we shall refer to as **documents** in this paper, though each piece could well be a page from a book, a "hypercard", an image, a bar of music, or whatever.

Each document has several **terms** that may be used as keys to retrieve the document. For a text document, these terms could simply be all the words in the text. Alternatively, they could be more complex entities, such as, word pairs or triples. There is usually a "stop-list" of common words that cannot be used as key terms. Sometimes a list of "keywords" is explicitly specified and only these may be used as terms that can be used as keys for retrieval. For non-text documents, there is an even greater variety of techniques by which a set of key terms can be obtained for any given document: whether through automated analysis of the document or through human input.

The question we address in this paper is how to create an index over a large number of terms for a large set of documents. Clearly, there are two objective functions: one is to minimize the storage overhead required for the index. The other is to minimize the amount of time taken for a retrieval. If the database is dynamic (e.g., archival, such as a library collection of books), the effort to update the index is also in issue. We address all these issues.

There are several indexing techniques that have been proposed [5] for problems of the sort we are interested in. The most popular commercial approach is the "inverted index" [2]. For each index term, a (variable length) "postings list" is created of the documents that it occurs in. The index terms themselves are usually organized in a B-tree [4], so that given a particular term, one tree traversal is required to get to the appropriate leaf where the required list of documents may be found. The major drawback of this technique is that there is a substantial space overhead – from 50% up to 300% in some cases [8]. In addition, long lists of documents may have to be manipulated when queries involve multiple terms, a task that may be onerous. Also, when a new document is added to the database, an update of the inverted index is required for each term in the document.

† This research was sponsored in part by the University of Maryland Institute for Advanced Computer Studies, by the National Science Foundation under grants DCR-86-16833, IRI-8719458 and IRI-8958546 and by the Air Force Office of Scientific Research under Grant AFOSR-89-0303.

The simplest possible indexing technique is to keep a bit matrix, one bit per term per document [10]. If the particular term occurs in the particular document, the bit is on, else it is off (see Figure 1). This technique by itself is rarely used, since the bit matrix is likely to be very sparse, wasting a lot of storage. However, variants of this technique are used, with one or more bits assigned to each term, and superimposing the bit assignments so that each bit may be turned on if any one of several terms occurs in a given document. These techniques as a group are called "signature file" techniques. See, for example, [7]. These techniques can generally be implemented with a smaller storage overhead (10%-20%), and a considerably lower insertion cost, than inverted indices. However, the search time grows linearly as the number of documents in the collection grows, since the signature (a bit or a bit pattern) associated with each individual document has to be examined in response to each query.

	$term_1$	...	$term_t$
doc_1	1	...	0
...			
doc_N	0	...	0

Figure 1. Example of a bit matrix (or bitmap or "signature file")

A central motivation for our work is Zipf's law [16], which, informally, states that a few instances occur most of the time and that most instances occur rarely. Variations of Zipf's law are often quoted as the 80-20 rule, or sometimes the 90-10 rule. On account of Zipf's law, it is not reasonable to assume that each index term occurs roughly equally often in the documents to be indexed. In fact, there is likely to be a tremendous imbalance.

In view of this imbalance, it is unlikely that any single technique for indexing text can do well in all situations. In this paper we propose a hybrid approach that merges the good points of both families of techniques discussed above. Hybrid approaches to indexing, in related but different contexts, have previously been proposed in [13] and in [15].

We begin in Section 2 with a review of Zipf's law, and an experimental study of the occurrence frequencies of terms in a database with Associate Press news wire stories. In Section 3, we introduce the main idea behind the proposed hybrid technique. In Sections 4 and 5, we develop the technique for a static (e.g., CD-ROM) database, and for a dynamic (archival) database respectively. In each case, we evaluate our technique analytically, both for single terms and for multi-term queries, and compare the expected performance with respect to the standard inverted index method widely used today. As performance measures, we use storage overhead due to the index, time to perform a query, and, in the dynamic case, time to update the index when a new document is inserted in the database. In Section 6, we recapitulate the results presented earlier in the form of guidelines for a practitioner. We close with some final remarks in Section 7.

2. TERM DISTRIBUTION

We obtained a database of 10,075 documents representing Associated Press newswire stories from 40 random days in 1989. The total database size was a little over 30 Megabytes. Every story from a particular day was selected with some care to eliminate stories that were duplicates or near duplicates. Each story was 300 to 600 words long, and was treated as a separate document. Each alphabetic string in each story was treated as a word, with case being ignored. (All words were converted to lower case for uniformity).

Figure 2 shows a log-log plot of the rank of a word, on the X-axis, versus its frequency, measured as

the number of times it occurred in all the documents combined, on the Y-axis. In addition to the full database just described, we also ran experiments on a 10% fraction of the database, of size about 3Mb. Results for this smaller database are plotted in Figure 2, as crosses ("+"). The curves for the two databases are almost identical in shape, indicating that our results are largely insensitive to database size.

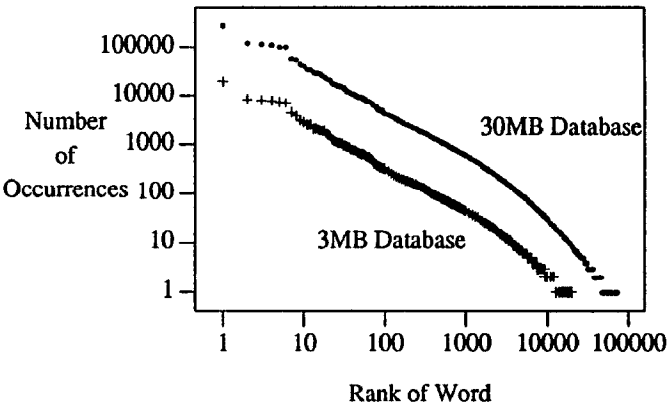

Figure 2: Log-log plot of the frequency vs. rank distribution
for the full (30Mb) database (with "•") and for the sample (3Mb) (with "+").

Zipf [16] observed that not all words occur as often in any given piece of text. In fact, a few words occur very often, and most words occur only infrequently. If words are ranked by their frequency of occurrence, then the following distribution is observed:

$$f_i = F(i) = \frac{K}{i} \tag{2.1}$$

where f_i is the frequency of the word with rank i. K is a constant, usually $K = V$, where V is the vocabulary of the body of the text. Plotted on a log-log graph, the above distribution is a straight line with slope -1.

The curves in Figure 2 are almost straight lines, as predicted by Zipf, but not exactly so. They certainly represent a highly skewed distribution. In other words, even though we have a database that is not a single cohesive document created by a single author, the distribution of the occurrence frequencies of words in the database is highly skewed.

We are interested in not the total frequency of occurrence of a particular word in a document (or set of documents), but rather the **document frequency** of each word, that is, the number of documents that the word appears in (one or more times). That is to say, we discard the second and subsequent occurrences of a word in any document, to obtain a distribution of the number of documents in which each word occurs. Intuitively, we would expect the document frequency distribution to be skewed, too, but not as skewed as the original distribution. The reason is that frequent words will appear many times in the same document, and thus will suffer most from the elimination of duplicate words in a document. We performed measurements. Fig. 3 plots the results in a log-log graph. The X-axis is the rank of the word as before. The Y-axis is the document frequency of the word.

Not every word is worth indexing on. For example, prepositions and common pronouns occur frequently and have little semantic content for purposes of retrieval. In addition, in our particular database, there were a few words that appeared on all stories, such as "est" or "edt" (for the time the story was filed). Words of this nature are called **noise words**, and can be collected into a **stop list**. The rest of the words are called **terms**. In our particular database, we found 128 noise words. These included all of the most common 50 words, and were all within the first 200 ranked words. Without losing much accuracy, we can assume that the first r_n most common words are noise words, with $r_n = 128$ in our case. This

causes the flat part of the curves of Figure 3 to be deleted, resulting in a curve that is closer to a Zipf distribution.

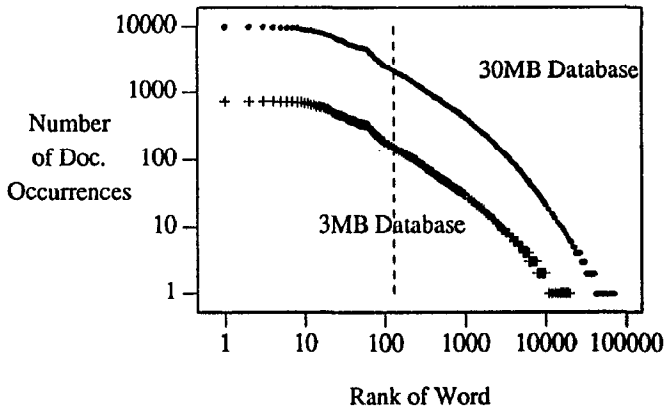

Figure 3: Log-log plot of the document frequency vs. rank
for the full (30Mb) database ("•") and for the 3Mb sample ("+").

When we say **vocabulary** in the following, we mean the total number of different terms in the document, database, or other entity being referred to.

3. PROPOSED APPROACH

Since the inverted index is by far the most common retrieval structure used, let us begin with a description of that as the basic structure. We then address areas for potential improvement.

3.1. The Inverted Index Structure

The traditional inverted index consists of two files: the sorted list of terms, usually organized as a B-tree (although hash implementations have been reported [11] and studied [6]), and the postings file. For the rest of the paper, the first file will be referred to as the "B-tree". Each leaf page contains records of the form,

<term, delimiter, pointer_to_head_of_postings_list>.

Each term is associated with a postings list, that is, a list of pointers to the documents that contain the term. The postings lists are stored in the postings file; depending on the environment, they can be stored contiguously (static environment), or in linked-lists of fixed-length records (dynamic environment), or in some other fashion. Figure 4 illustrates the file structure.

As we shall soon see, most of the storage overhead is due to the postings lists. So minimizing the size of the postings list is very important. Due to the highly skewed distribution, a large fraction of the size of the postings list is contributed by a few commonly occurring terms. Let us see how we can reduce this.

One immediate solution is to maintain a "negative" list, that is, a list of documents in which a term does not occur rather than the ones in which a document does occur. If a term occurs in more than 50% of the documents, this negative list will be shorter than the usual (positive) list. Experimental analysis, for the news story database, showed that while there are a few words that occur in more than 50% of the documents, they are all noise words, such as 'a', 'the', 'to', etc., which are not likely to be useful as index terms, and were all included in the stop list. As such, negative lists are not a useful idea.

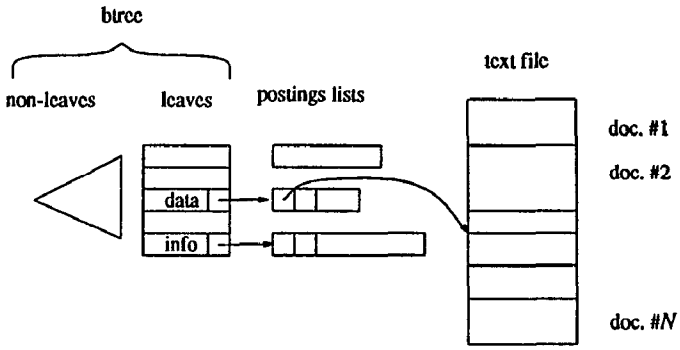

Figure 4: Traditional, inverted-index file structure.

3.2. A Hybrid Strategy

The heart of our proposal is to take advantage of the skew of the frequency distribution, by treating the "frequent" terms in a different way. We still keep the B-tree file, which holds *all* the terms; the "rare" terms are treated the traditional way (variable length postings lists); for each frequent term, we propose storing its postings list as a bit vector. Grouping the bit vectors together, we have a bitmap (see Figure 1 and Figure 5). Notice that we need only one bit, to flag a term as frequent or rare. This bit could be incorporated within another field, e.g., negative number for a postings head can signify a frequent term.

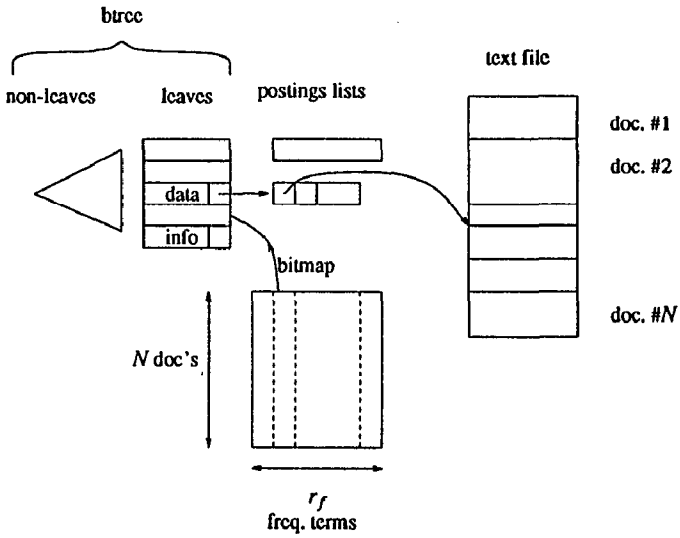

Figure 5: File structure for the proposed hybrid method - static case.

According to the specific environment, we propose different organizations of the bitmap. Moreover, the cut-off for the "frequent terms" is computed based on the parameters of the specific database. Terms 1 through r_f, for some analytically computed cut-off rank r_f, are considered frequent. Terms r_f+1 through V are rare. (Noise words are not considered terms, and are assumed to have been eliminated before ranks are determined for the terms).

Symbol	sample value	Definition
h_{btree}	3	height of the B-tree
h	h_{btree}-1	levels of B-tree NOT in core
m	78	order (fanout) of B-tree
U	U_{static} = 1.0	space utilization of B-tree (static case)
	$U_{dynamic}$ = 0.8	space utilization of B-tree (dynamic case)
O_{tree}		space occupied by B-tree
O_{post}		space occupied by postings
LP_i		length (in pages) of postings list of i-th term
$\overline{LP}$		avg. length of postings list in pages
f_i		no. of docs in which i-th term occurs
C_i		number of postings records of i-th term
$\overline{C}$		avg number of postings records
s	6	pointers per postings record

Table T1: Symbols and Definitions (Data Structure Parameters)

Symbol	sample value	Definition
N	10075	number of documents
S	30Mb	total size of the collection
V_i		vocabulary of the i-th document
$\overline{V_d}$	178	avg. vocabulary of a document
V	71,814	vocabulary of collection
l_v	8	length of a vocabulary word
P	1024	page size in bytes (Mag. Disk)
	16K	page size (Optical Disk – CD-ROM or WORM)
p	4	pointer size, in bytes
b	8	bits per byte
T_{rand}	30 msec	avg. time for random disk access (Mag. Disk)
	200 msec	avg. time for random disk access (Optical Disk)
T_{seq}	3 msec	time for sequential disk access

Table T2. Symbols and Definitions (Input Parameters)

Symbol	sample value	Definition
r_f		rank of last "frequent" term
q_i		prob. that the i-th term appears in a query
q	0.33	prob. that a query term is frequent
c		number of terms in a multiple term query
t_1		search time for single term query
t_c		search time for c-term query
t_{IN}		insertion time

Table T3: Symbols and Definitions (Query & Insertion Parameters)

None of the analysis in this and in the following sections assumes any specific distribution of terms, database size, vocabulary, and so forth. Sample values of parameters of interest to us are listed in Tables T1, T2, and T3. These parameter values are obtained from the experiments described in the previous section, are "standard" values that are generally considered reasonable for a practical implementation, or are values computed analytically from other values determined by one of the methods above.

While we have properly determined parameters to evaluate the performance of an indexing technique for any one particular query, we have not said anything about what the queries are like: what sort of distribution do they have over the terms. Unfortunately, this distribution can only be determined by logging the queries posed to a system by users; even worse, it may change over time, as the users' interests shift from one topic to another. However, as we show next, we can choose the cut-off rank r_f in such a way that, for a query on any given term, our hybrid methods will never do worse than the traditional method. Under such a design, our methods will always have non-zero speed-up, regardless of the distribution of the queries. The distribution of queries will just determine the magnitude of the savings.

The easiest assumption to make is that each term is equally likely to be specified in a query. Notice that Eq. (3.1) and (3.2) reduce to the uniform case, for $q = r_f/V$. In fact, our analysis would be simplified if we could make such an assumption. Unfortunately, such is not likely to be the case. As it has been pointed out [14], the rarest terms are unusual words that are unlikely ever to be the basis of a query; many of them are often typographic errors or misspellings [1]. In particular, they are unlikely to be the basis of a multi-term query because the typical reason to have a multi-term query is that no single term in it is specific enough.

In order to obtain some arithmetic examples for the speed-up, we have explicitly introduced the parameter "q", which denotes the probability that a term specified in a query is a frequent term. Thus, if q_i is the probability that the i-th term will appear in a query, then we assume that

$$q_1 = q_2 = \cdots = q_{r_f} = q/r_f \tag{3.1}$$

and

$$q_{r_f+1} = \cdots = q_V = (1-q)/(V-r_f) \tag{3.2}$$

We have used a value of 0.33 for q, the probability that a term specified in a query is a frequent term.

4. STATIC CASE

Proposed Hybrid Structure

Whereas each document posting recorded in an inverted index for a term requires p bytes of storage, a simple matrix organization would require only 1 bit. We use a hybrid storage structure, recording the usual inverted index, but for "frequent" terms, rather than storing a postings list, we store a reference to an appropriate position in a bit matrix, such as the one shown in Fig. 1. From now on, such a matrix will be referred to as the **bitmap** or **bit matrix** or **signature file**; one row of it will be called **document signature**; one column of it will be called **bit slice** or **bit vector**. For the static case, it is most efficient to store the bit matrix in a bit-sliced form (column-wise), since only the columns corresponding to the specified terms need be retrieved in response to a query.

When a retrieval request is supplied, the B-tree is traversed as usual. If the term specified in the query is "rare", then a postings list is obtained at the leaf as usual. However, if the term specified is "frequent", then the appropriate bit vector in the bit matrix is examined.

Space for the traditional method

The traditional inverted index consists of a B-tree with all the terms in the vocabulary, plus a file with postings (pointers to the qualifying documents). Thus the space overhead O_{inv} for the inverted index is

$$O_{inv} = O_{tree} + O_{post} \tag{4.1}$$

For the B-tree, the non-leaf nodes will account for $\approx 1/m$ of the space of the B-tree, where m is the fanout (or "order") of the tree. Typically, $m \gg 1$. Thus the bulk of the B-tree overhead is due to the leaves:

$$O_{tree} \approx (l_v + 1 + p) * V/U \tag{4.2}$$

where V is the total vocabulary of the database and U is the utilization of the leaves. If we pack the tree carefully, we can assume

$$U = U_{static} \approx 1.00 \tag{4.3}$$

Let V_i be the vocabulary of the i^{th} document, and $\overline{V_d}$ be the average vocabulary over all the documents in the database.

$$\overline{V_d} = 1/N \sum_{i=1}^{N} V_i = 1/N \sum_{i=1}^{V} f_i \tag{4.5}$$

We shall find it convenient to divide $\overline{V_d}$ into two parts: $\overline{V_{freq}}$ and $\overline{V_{rare}}$. $\overline{V_{freq}}$ is the average number of frequent terms per document, $\overline{V_{rare}}$ is the average number of rare terms per document, and $\overline{V_{rare}} + \overline{V_{freq}} = \overline{V_d}$.

$$\overline{V_{freq}} = 1/N \sum_{i=1}^{r_f} f_i \tag{4.6}$$

$$\overline{V_{rare}} = 1/N \sum_{i=r_f+1}^{V} f_i \tag{4.7}$$

Then the space required for the postings list can also be divided into two parts corresponding to the rare and frequent terms.

$$O_{post,freq} = N * \overline{V_{freq}} * p \tag{4.8}$$

$$O_{post,rare} = N * \overline{V_{rare}} * p \tag{4.9}$$

$$O_{post} = O_{post,freq} + O_{post,rare} = N * \overline{V_d} * p \tag{4.10}$$

Space for hybrid method

The space overhead for storing the index in the proposed hybrid method is given as:

$$\begin{aligned} O_{hyb} &= O_{tree} + O_{post,rare} + O_{bitmat,freq} = O_{tree} + O_{post} - O_{post,freq} + O_{bitmat,freq} \\ &= O_{inv} - O_{post,freq} + O_{bitmat,freq} \end{aligned} \tag{4.11}$$

Thus the only space difference between the hybrid method and the inverted index is for the frequent terms, which are stored as bit-vectors rather than in postings lists. The space required for the bit-vectors can be computed as

$$O_{bitmat,freq} = N * r_f / b \quad \text{bytes} \tag{4.12}$$

where b is the number of bits in a byte. The savings in storage with respect to the inverted index method is:

$$O_{post,freq} - O_{bitmat,freq} = \sum_{i=1}^{r_f} (p * f_i - N/b) \tag{4.13}$$

We wish to choose r_f such as to maximize this summation. The (finite) differential with respect to r_f is $p * f_{r_f} - N/b$. Since f_i is monotonically non-decreasing in i (by definition of rank), this differential is also monotonically non-decreasing in i. We must choose r_f to be the highest rank, such that this differential is still positive. That is,

$$f_{r_f} \approx N/(b * p) \tag{4.14}$$

Search time for inverted index

For all the analyses that involve page reads and writes, we assume that each page request results in a page fault, except for the root of the B-Tree, which we assume is pinned in memory. We believe that such an assumption is realistic in a typical environment with a large database and users who have a variety of interests. In an inverted index, the time required for a single term query on the i-th term is:

$$t_{1,i} = h * T_{rand} + (T_{rand} - T_{seq} + T_{seq} * LP_i) \tag{4.15}$$

This equation represents h random disk accesses to traverse the B-tree, and one random access followed by $LP_i - 1$ sequential accesses to read the postings list. If h_{btree} is the height of the B-tree, and if the top one or two levels of the B-tree are cached in memory, then h denotes the rest of the levels. Usually, $h = h_{btree} - 1$. The symbols T_{rand} and T_{seq} are the times for a random and sequential disk access, respectively. LP_i is the length of the postings list of the i-th term, in pages:

$$LP_i \approx \left\lceil \frac{f_i * p}{P} \right\rceil \tag{4.17}$$

The above equation is an exact equality if each postings list that spans across pages is guaranteed to be aligned to start at the beginning of a page.

If we know the distribution of queries ($q_i, i = 1 \cdots V$), we can use eq. (4.15) to calculate the average response time. Here, we provide the formulae according to eq. (3.1-2). Let us calculate the response time separately for frequent and for rare terms, so that $\overline{LP_{freq}}$ and $\overline{LP_{rare}}$ denote the average length of a postings list in pages, for frequent and rare terms, respectively. We can then develop equations for the expected time to execute a query against a single frequent term and a single rare term, respectively, as follows:

$$t_{1,rare} = h * T_{rand} + (T_{rand} - T_{seq} + T_{seq} * \overline{LP_{rare}}) \tag{4.18}$$

$$t_{1,freq} = h * T_{rand} + (T_{rand} - T_{seq} + T_{seq} * \overline{LP_{freq}}) \tag{4.19}$$

with

$$\overline{LP_{freq}} = 1/r_f \sum_{i=1}^{r_f} LP_i \tag{4.20}$$

$$\overline{LP_{rare}} = 1/(V - r_f) \sum_{i=r_f+1}^{V} LP_i \tag{4.21}$$

The search time t_1 for a single term query is the average of the search times for rare and frequent terms, weighted by the probability of a rare (respectively, frequent) term being specified. This probability is captured in the parameter q, the probability that the query term is frequent, so that we can write:

$$t_1 = q * t_{1,freq} + (1 - q) * t_{1,rare} \tag{4.22}$$

The search time for a multiple-term query with c terms is simply the time to perform c single term queries (plus some additional time to manipulate the postings lists, which we can ignore assuming that this manipulation is entirely in memory).

$$t_c = c * t_1 \tag{4.23}$$

Search time for hybrid method.

In the hybrid structure we propose, the time to perform a retrieval remains exactly the same as before when a single rare term is specified. When a single frequent term is specified, the time to traverse the B-tree is still the same, but now instead of retrieving a postings list, one has to retrieve a bit-slice from the bit-matrix. The time required is:

$$t_{1,freq} = h*T_{rand} + (T_{rand} - T_{seq} + T_{seq} * \lceil N/(b*P) \rceil) \quad (4.24)$$

By our choice of cut-off for frequent words, we know that $N/(b*p) \leq f_i$ for all $i < r_f$. We then have $\lceil N/(b*P) \rceil \leq \left\lceil \frac{f_i * p}{P} \right\rceil = LP_i$. Therefore, the time required to perform the retrieval for any term is no worse with our proposed hybrid method than for the standard inverted index, and could be better if the specified term is very frequent. In consequence, the expected time is also less for our technique than for standard inverted index.

Note also, that if we selected r_f to minimize the expected time to perform a single-term retrieval, we would get exactly the same answer as we obtained when we selected r_f to minimize the storage. In other words, the optimum cut-off point is the same for both objectives!

The expected time for a single term query can be computed from $t_{1,freq}$ and $t_{1,rare}$ as before, as well as the time for a multi-term query (eq. (4.22-23)).

$$t_1 = q*t_{1,freq} + (1-q)*t_{1,rare} \quad (4.22)$$

$$t_c = c*t_1 \quad (4.23)$$

Arithmetic example

To estimate the benefit due to the hybrid method quantitatively we use parameter values from our sample 30MB database, as shown in tables T1, T2 and T3: V = 71,814 distinct terms and N=10,075 documents. From measurements on the database we have that the cut-off rank from eq. (4.14) is

$$r_f = 1214 \quad \text{and that} \quad \sum_{i=1}^{r_f} f_i = 875{,}951 \; ; \; \sum_{i=1}^{r_f} V = 1{,}792{,}079$$

Plugging in these values into the formulae above, we get the results shown in Table T4. We see that the hybrid indexing method achieves significant space savings, compared to the standard inverted index method. In addition, it achieves some smaller savings on the search time.

	traditional	hybrid	savings (%)
Space [Mb]	8.10	6.13	32
$t_{1,freq}$ [msec] (magn.disk)	96.99	90.6	7
$t_{1,freq}$ [msec] (CD-ROM)	400	400	0

Table T4: Summary of arithmetic example for the static case.

5. DYNAMIC CASE

Thus far we have not considered any updating of the database. In this section, we include this consideration as well. We assume that the environment is archival, that is, there are insertions of new documents, but deletions and updates are very rare. This is a realistic assumption; most of the text retrieval applications have archival nature, such as library automation [14], patent and law office [9], electronic office filing [3], newspaper articles, *etc.*)

Stability of Frequency Distribution

In order to apply our hybrid method beneficially, the set of frequent terms has to be stable despite the growth of the database. We believe that most archival databases do not radically alter their nature all of a sudden. Therefore, it is reasonable to assume that terms that are frequent will remain frequent as more documents are added to the database, and terms that are rare will remain rare. After a large number of insertions, divergence may begin to appear. In that case we will be forced to re-adjust the indices. This should happen only rarely, and should therefore be acceptable. Moreover, clever incremental reorganization schemes may require rewriting of only a small portion of the index.

To verify our claim above, we performed a small experiment. We considered the most frequent 300 terms in a small sample (representing 5%) of the database we had on hand. We found that over 80% of these terms were in the 300 most frequent terms in the entire database, and almost all of them were within the 600 most frequent. In other words, even as a database grew by a factor of 20, the most frequent terms determined from a small sample remained more or less the most frequent terms in the full database.

We also considered a less extreme situation in which instead of growing the database by such a large factor, we simply doubled the size of the database. Now, the 300 most frequent terms in one half of the database were more than 95% within the 300 most frequent in the full database, and all were within the first 350.

We repeated the above experiments, considering the first 1000, and the first 2000 most frequent terms from a sample (rather than the first 300). The results remained substantially similar. Most frequent terms from the sample remained frequent, provided that the number of terms considered frequent was significantly less than the total vocabulary of the sample (say, less than 10% of it).

Assumptions Regarding Data Organization

In order to allow growth of the database, the postings lists are organized in chains of fixed-length **postings records**. Each postings record consists of s postings pointers, plus an extra pointer that points to the next postings record (or is null). This organization will add to the space overhead, because of the link pointer and because the last postings record of a chain will be approximately half empty. The choice of the best value for s is an open problem; from the experiments on the 30Mb database, we found that the value of $s=6$ minimizes the space overhead.

An alternative method of organizing the postings lists is to let the size of the postings records to be adaptive – with a larger postings record obtained on each overflow. Determining the size of each overflow postings record is a non-trivial task, which we are currently studying. Since we do not have a concrete design for the method yet, we cannot study it further.

The retrieval of each postings record requires one random access, *i.e.*, the system does NOT try to cluster the postings records of a chain, because such an attempt would make insertions even more expensive. Moreover, the B-tree cannot be packed, as opposed to the situation in the static environment. Therefore, the utilization factor is lower. Assuming a deferred splitting algorithm (as in a B^*-tree), we have

$$U = U_{dynamic} \simeq 0.8 \tag{5.0}$$

The space overhead and retrieval time criteria remain the same as before, even in a dynamic situation. However, some parameter values change, as discussed above.

Modification to the Hybrid Structure

In most text indexing situations, the primary update activity performed is the insertion of a new document into the database. As such, we shall focus on the effort to insert a new document as the primary cost of update. Consider the hybrid structure that we have been using thus far. In this structure, every time a new document is added to the database, a new row has to be added to the bit matrix corresponding to the frequent terms, and for every other term in the document appropriate entries have to be made in the postings lists, and the B-tree updated if required.

One can readily see that adding a bit to the end of every column bit-vector is a slow procedure. For this reason, in a dynamic environment we propose that the bit matrix be stored row-wise (as a set of document signatures), so that when a new document is added, a new row-vector can be written in one swoop. Of course the rare terms have still to be inserted into the postings lists in the usual way. When a query is specified, instead of reading just a single bit-slice, now all the document signatures will be read, if the specified query term is frequent. (See [12]. for a "group-slice" technique that is a compromise between row-wise and column-wise storage).

Space

The space required for the standard inverted index structure, is computed once more as:

$$O_{inv} = O_{tree} + O_{post} \tag{4.1}$$

with the space for the tree being dominated by the leaves and being obtained as:

$$O_{tree} \simeq (l_v + 1 + p) * V/U \quad \text{bytes} \tag{4.2}$$

The space for the postings file has increased, due to the dynamic linking of small groups of postings:

$$O_{post} = (s+1) * p * V * \bar{C} \quad \text{bytes} \tag{5.1}$$

s is the number of postings per postings record, and the additional 1 is for the pointer to the next record. (We have assumed these pointers to be the same size as pointers in the postings list. If not, the formula above can easily be modified accordingly). $\bar{C}$ is the average length of a postings chain (in number of postings records):

$$\bar{C} = \frac{1}{V} \sum_{i=1}^{V} \left\lceil \frac{f_i}{s} \right\rceil \tag{5.2}$$

The quantity inside the summation, the number of postings records for the i-th term, is written as C_i:

$$C_i = \left\lceil \frac{f_i}{s} \right\rceil \tag{5.3}$$

Then, the average number $\overline{C_{freq}}$ of postings records for the frequent terms is

$$\overline{C_{freq}} = \frac{1}{r_f} \sum_{i=1}^{r_f} C_i \tag{5.4}$$

and the respective formula for the rare terms $\overline{C_{rare}}$ is

$$\overline{C_{rare}} = \frac{1}{V - r_f} \sum_{i=r_f+1}^{V} C_i \tag{5.5}$$

Once more, the space required by the postings list in the inverted index method can be expressed in two parts as in eqn. (4.8)

$$O_{post\_freq} = (s+1) * p * r_f * \overline{C_{freq}} \tag{5.6}$$

$$O_{post,rare} = (s+1)*p*(V-r_f)*\overline{C_{rare}} \tag{5.7}$$

As in the static case, the space difference between the hybrid method and the inverted method is expressed by

$$O_{hyb}=O_{inv}+O_{bitmat,freq}-O_{post,freq} \tag{4.11}$$

where $O_{bitmat,freq}$ is the same as for the static case, and $O_{post,freq}$ is the overhead of the postings of the frequent terms, as shown above.

Once more, the cut-off rank r_f that minimizes the space overhead can be computed by setting the differential improvement to zero, and works out as follows:

$$C_{r_f}=N/(b*p*(s+1)) \tag{5.8}$$

The cut-off that satisfies the above equation will be called $r_{f,2}$ for the rest of this paper. The $r_{f,2}$ cut-off obtained here is close to, but larger than the cut-off in the static case.

Search Time

Compared to the static case, in the standard inverted index the search time for a single term will be longer, because we have one random disk access for every postings record†. Thus, for a query on the i-th term, we have

$$t_{1,i}=h*T_{rand}+T_{rand}*C_i$$

As in the static case, if the query frequencies q_i are known, t_1 can be calculated as the average of the $t_{1,i}$'s $i=1\cdots V$. If the query frequencies are unknown, then we can use the assumptions implied in eqs. (3.1-2). Then, we have

$$t_{1,freq}=h*T_{rand}+T_{rand}*\overline{C_{freq}} \tag{5.9}$$

$$t_{1,rare}=h*T_{rand}+T_{rand}*\overline{C_{rare}} \tag{5.10}$$

and

$$t_1=q*t_{1,freq}+(1-q)*t_{1,rare} \tag{4.22}$$

A multi-term query is given by eq. (4.23)

$$t_c=c*t_1$$

For the hybrid structure, $t_{1,rare}$ remains unaltered, and $t_{1,freq}$ can be computed as:

$$t_{1,freq}=h*T_{rand}+t_{bitmap} \tag{5.11}$$

where t_{bitmap} is the time to scan the bit-map sequentially

$$t_{bitmap}=(T_{rand}-T_{seq}+T_{seq}*N*r_f/(b*P))$$

To optimize the search time for single-term queries, the cut-off rank r_f should obey the equation:

$$t_{bitmap}<T_{rand}*C_{r_f} \tag{5.12}$$

or, equivalently,

$$T_{rand}-T_{seq}+T_{seq}*N*r_f/(b*P)<T_{rand}*C_{r_f} \tag{5.13}$$

Let $r_{f,1}$ be the largest value that satisfies eq. (5.13). Eq. (5.13) expresses the requirement that, for the last frequent term, following the chain of postings records will be slower than scanning the bit matrix. Choosing $r_{f,1}$ as the cut-off rank guarantees that the hybrid method will give a better response time for any frequent term, as compared to the traditional method. Of course, for the rare terms, the response

† There is a slight probability that two successive postings records of interest are on the same page. This probability is typically low enough that we have chosen to ignore it (assumed zero) in the formulae derived here.

times are the same for both methods.

In the static case, by having a bit-slice structure for the bit matrix, we were able to guarantee that the time to read the bit-slice would be less than the time to read a corresponding postings record. In fact, the choice of r_f that minimized space was exactly the same as the choice of r_f that minimized expected retrieval time. Unfortunately, we do not have the same situation here. There are several additional factors, such as the respective times for random and sequential access to disk, the number of documents in the database, *etc.* Since we are forced to scan the entire bit-matrix for every query involving a frequent term, purely in terms of retrieval time, $r_{f,1}$ usually has a lower value. For our example database, this optimum value $r_{f,1}$ is around 690 rather than around 1200. Notice, however, that the hybrid method still improves the space overhead, even with a lower cut-off. Guidelines on how to choose between the two cut-offs are discussed in section 6.

Regardless of the final choice of the cut-off rank, the response time is given by eq. (4.22), assuming that eqs. (3.1-2) hold:

$$t_1 = q * t_{1,freq} + (1-q) * t_{1,rare} \tag{4.22}$$

The calculation of the response time for multi-term queries is the most complicated derivation in this paper. The complication, as well as the savings of the hybrid method, stem exactly from the fact that, if x terms in the query are frequent, the bit matrix has to be scanned only once, instead of x times. Formally:

The response time for a c-term query, under the assumptions of eqs. (3.1-2) is given by

$$t_c = h * c * T_{rand} + (1-(1-q)^c) * t_{bitmap} + (1-q) * c * \overline{C_{rare}} * T_{rand} \tag{5.14}$$

We see that for multi-term queries, there is an even greater benefit to using the hybrid method relative to a standard inverted index, than in the case of single term queries.

Insertion Cost

For the standard inverted index structure, the time for insertion is

$$t_{IN} = h * \overline{V_d} * T_{rand} + \overline{V_{freq}} * (\overline{C_{freq}} + 1) * T_{rand} + \overline{V_{rare}} * (\overline{C_{rare}} + 1) * T_{rand} + t_{new} \tag{5.15}$$

because we need to traverse the B-tree $\overline{V_d}$ times; for each frequent term ($\overline{V_{freq}}$ of them per document) we need $\overline{C_{freq}}$ reads to traverse a chain of frequent postings plus one disk access to write back the results; and symmetrically for the rare terms. ($\overline{V_{freq}}$ is the average number of frequent terms per doc; $\overline{V_{rare}}$ is the average number of rare terms per doc; $\overline{V_{rare}} + \overline{V_{freq}} = \overline{V_d}$, the average number of terms per doc.). The term t_{new} reflects the cost of introducing a new term, including potential splits of the B-tree this might cause, weighted by the probability of a new term being introduced. This term is common to both this and the hybrid structure, and we do not evaluate it any further.

The above insertion time is extremely long, exactly because we have to scan sequentially all the postings records of a term, to locate and updated the last record. It can easily be shortened, if, *e.g.*, for each term, we store a pointer to its last postings record, in addition to its first postings record. An alternative design would be to link the postings records in the reverse order. Regardless of the low-level details, with negligible or even zero additional space overhead, we can reach the last postings record of each term in one random disk access. Thus, the insertion time becomes

$$t_{IN} = h * \overline{V_d} * T_{rand} + \overline{V_{freq}} * (1+1) * T_{rand} + \overline{V_{rare}} * (1+1) * T_{rand} + t_{new} = (h+2) * \overline{V_d} * T_{rand} + t_{new} \tag{5.16}$$

because we need to descend h levels of the B-tree; one disk access to read the last postings record, and one more to write it back.

For the hybrid structure, the insertion cost is:

$$\begin{aligned} t_{IN} &= h*\overline{V_d}*T_{rand} + t_{new} && \textit{btree} \\ &+ T_{rand} - T_{seq} + \left\lceil r_f/(b*P) \right\rceil * T_{seq} && \textit{bitmap} \\ &+ (\overline{C_{rare}}+1)*\overline{V_{rare}}*T_{rand} && \textit{postings} \end{aligned} \tag{5.17}$$

because we need to descend the B-tree $\overline{V_d}$ times, we need to append the new row at the end of the bitmap and to update the postings lists for $\overline{V_{rare}}$ rare terms. If the last postings record of each term is accessible in one disk access, then, using the same justification as with eq. (5.16), the last term of eq. (5.17) changes, to become

$$\begin{aligned} t_{IN} &= h*\overline{V_d}*T_{rand} + t_{new} && \textit{btree} \\ &+ T_{rand} - T_{seq} + \left\lceil r_f/(b*P) \right\rceil * T_{seq} && \textit{bitmap} \\ &+ (1+1)*\overline{V_{rare}}*T_{rand} && \textit{postings} \end{aligned} \tag{5.18}$$

Comparing equations (5.16) and (5.18), we find the saving in insertion cost due to the hybrid method to be

$$2*\overline{V_{freq}}*T_{rand} - T_{rand} - \left\lceil r_f/(b*P) \right\rceil * T_{seq} + T_{seq} \tag{5.19}$$

If a particular term occurs f_i times in N documents, then the probability that it will occur in a new document is roughly f_i/N. Using this, and ignoring the ceiling function, we can write the differential savings due to including the i-th term in the frequent set as being roughly

$$2*f_i*T_{rand}/N - T_{seq}/(b*P) \tag{5.20}$$

Let r_f be the largest rank that satisfies (5.22). Once more, since f_i is monotonically non-increasing in i, this function is monotonically non-increasing in i, indicating that the largest benefits are likely to be obtained by marking the most frequent terms as frequent, with diminishing benefits as we include more and more terms. For large enough N, there may be an $r_{f,3}$ cutoff beyond which the quantity in eqn. (5.20) becomes negative indicating a worsening instead of a saving. For smaller databases, due to the large page size and large differential between random and sequential disk access time, no such cutoff maybe found, and it may actually be preferred to treat every term as frequent. This is exactly the case in our 30Mb database: eq. (5.22) gives that every term that occurs in more than f_i=0.06 documents, should be treated as frequent. This is not a very surprising result. Indeed insertion cost is the one criterion on which signature files win handsomely over inverted indices. What our analysis here indicates is that if the insertion cost is to be minimized in our hybrid approach, then the optimum cutoff is very large, making the hybrid technique "closer" to a signature file than an inverted index. A cut-off as large as $r_{f,3}$ will optimize insertion time, imposing heavy penalties on the storage and search time. Thus, we shall not consider the minimization of insertion cost further in this paper. Suffice to say that it can be done, if necessary, as shown above.

Arithmetic Example

Once more, we use data from our example 30MB database, with all parameters values as shown in Tables T1, T2, and T3. As discussed earlier, there is no single optimum value of r_f. The cut-off $r_{f,2}$ that minimizes the space is $r_{f,2}$=1420; the cut-off $r_{f,1}$ that minimizes the search time for a single term query is $r_{f,1}$=690 (assuming a magnetic disk as the medium). The cut-off assumes intermediate values when we wish to minimize multi-term retrieval queries.

We have chosen r_f=800 as a representative value for our computations. The size of the postings record is

s=6 pointers. Assume q=0.33. Finally, from statistics on the database, we have:

$\sum_{i=1}^{V} f_i$= 1,792,079 postings pointers for all terms

$\sum_{i=1}^{r_f} f_i$= 721,531 postings pointers for frequent terms

$\sum_{i=1}^{V} C_i$ = 343,320 postings records for all terms

$\sum_{i=1}^{r_f} C_i$ = 120,603 postings records for frequent terms

Table T5 summarizes the results of the comparison using the values discussed above. The main observation is that the proposed method improves every performance measure.

	traditional	hybrid	savings (%)
Space [Mb]	10.7	8.4	27
$t_{1,freq}$ [sec]	4.58	3.0	52
t_1 [sec]	1.61	1.09	47
t_4 [sec]	6.44	3.04	112
t_{IN} [sec]	21.36	17.07	25

Table T5: Summary of arithmetic example for the dynamic case.

6. IMPLEMENTATION CONSIDERATIONS

In this paper we have proposed a hybrid technique for indexing large text databases. A central requirement for the proposed method is that the index terms be divided into two groups: frequent and rare. Here we briefly discuss how a practitioner could do this in a real application.

The classification of terms into two groups hinges on the determination of the occurrence frequencies of the terms. In a static database, these frequencies could be determined by direct measurements on the database. However, in the light of the experimental results presented in Sec. 5, one could get by even with a small sample (5-10%) of the database. The cut-off between frequent and rare terms is then determined analytically as shown in Sec. 4, eq. (4.14).

In an archival situation, the database designer can never have access to the entire database. As documents are added to the database, term frequencies will undoubtedly change. But in view of the experiments described in Sec. 5, we expect that by and large frequent terms will remain frequent and rare terms will remain rare, as the database grows. Thus, by using a snapshot of the database (or even a sample of a snapshot), occurrence frequencies can be determined.

In an archival situation there is the additional complication that the cut-off rank $r_{f,2}$ that minimizes the space (eq. (5.8)) is different (usually, larger) than the cut-off rank $r_{f,1}$ that optimizes the search time (eq. (5.13)). The safe, conservative solution is to choose $r_{f,1}$, the smallest cut-off, which optimizes the response time; this choice achieves sub-optimal, but, nevertheless, positive savings for the space and the insertion time, too. Such a choice guarantees that the response time for any single-term query will be at least as good as in the traditional method.

A larger value for the cut-off rank will improve the space overhead, the insertion time and the search time for multi-term queries, at the expense of a slower performance for single-term queries on the least

frequent of the frequent terms. It is up to the designer of the database to decide which value between the two extremes $r_{f,1}$ and $r_{f,2}$ to use as a cut-off; the formulae in section 5 will help to predict the impact of each decision on the performance measures.

Another design decision in the archival case is the choice of the size s of the postings records. Our measurements in the 30Mb database showed that the space overhead is relatively insensitive to s in the vicinity of $s=6$. We also performed the same measurements in two other text databases, that were available on-line: the first was the book, *Wuthering Heights*, and the second was a collection of Grimm stories. The sizes were 659Kb and 1.4Mb, respectively, and the optimal values for s were 4 and 7, respectively. For both databases, the increase on the space overhead was small, for small deviations from the optimal values of s. Thus, we recommend the value of $s=6$ as a good (if not optimal) choice, for databases with English text. It is interesting to examine whether $s=6$ is still a good choice for non-English text databases.

Our experience is summarized in the following pseudo-algorithm, which is intended as a practitioner's guide:

Design Guidelines

1 Obtain a reasonably large sample of the database (10% or more)

2 Sort the words on decreasing occurrence frequencies

3 Decide the list of "noise" words (''I'', ''the'', *etc.*, as well as words specific to the application, like ''est' in our Associated Press database)

4 Decide the cut-off rank r_f for the frequent terms:

  4.1 For a static database, use eq. (4.14).

  4.2 For a dynamic database, choose a value of s ($s \simeq 6$) to minimize the space of the traditional method; then, use eq. (5.13) and (5.8) to calculate the two cut-off ranks $r_{f,1}$ and $r_{f,2}$, the search-time optimal cut-off and the space-wise optimal cut-off, respectively. Then, try values of r_f in this range, estimate the performance measures (space, search time, insertion effort) and pick the value that gives the best results for the specific application. If in doubt, choose r_f to be the smallest cut-off $r_{f,1}$ (eq. (5.13)); such a choice will give sub-optimal, but positive savings for all the performance measures.

7. CONCLUSIONS

The thrust of this paper is to capitalize on the highly skewed distribution of term occurrence to design better text retrieval methods. We suggest a hybrid approach, that uses a B-tree inverted index for the rare terms, and a signature-like method for the frequent ones. Based on this concept, we fine-tune our designs for two environments: one static and the other archival. We show that, with a careful choice of the frequent terms, the proposed methods *improve all* the performance measures of interest (space, search time and insertion time), when compared to the standard inverted index method. Additional contributions of the paper include:

- the conclusion that even a small sample of the database can be used to determine the frequent terms with good accuracy. This conclusion was drawn after experimenting on a real, text database. The result is essential, because it gives us confidence to propose the hybrid method even in a dynamic

environment.

- the development of analytical models for the space overhead and the response time of the traditional and the hybrid methods for both environments, as well as the study of the insertion time for the dynamic environment. The response time analysis includes multi-term queries, in addition to single-term ones.
- the presentation of statistical data from a real database (30MB) that supports our assertion of a highly skewed term distribution. Based on the above data, the paper presents arithmetic examples that illustrate the superiority of our designs over a plain inverted index.
- Finally, the paper provides guidelines to a practitioner on how to choose the parameters for the hybrid method.

References

1. Bourne, C.P, "Frequency and Impact of Spelling Errors in Bibliographic Databases," *Information Processing and Management* **13**(1), pp. 1-12 (1977).
2. Cardenas, A.F., "Analysis and Performance of Inverted Data Base Structures," *CACM* **18**(5), pp. 253-263 (May 1975).
3. Christodoulakis, S., M. Theodoridou, F. Ho, M. Papa, and A. Pathria, "Multimedia Document Presentation, Information Extraction and Document Formation in MINOS: A Model and a System," *ACM TOOIS* 4(4) (Oct. 1986).
4. Comer, D., "The Ubiquitous B-Tree," *Computing Surveys* **11**(2), pp. 121-137 (June 1979).
5. Faloutsos, C., "Access Methods for Text," *ACM Computing Surveys* **17**(1), pp. 49-74 (March 1985).
6. Faloutsos, C. and R. Chan, "Fast Text Access Methods for Optical and Large Magnetic Disks: Designs and Performance Comparison," *Proc. 14th International Conf. on VLDB* , Long Beach, California, pp. 280-293 (Aug. 1988).
7. Faloutsos, C., "Signature-Based Text Retrieval Methods: A Survey," *IEEE Data Engineering* **13**(1), pp. 25-32 (March 1990).
8. Haskin, R.L., "Special-Purpose Processors for Text Retrieval," *Database Engineering* **4**(1), pp. 16-29 (Sept. 1981).
9. Hollaar, L.A., K.F. Smith, W.H. Chow, P.A. Emrath, and R.L. Haskin, "Architecture and Operation of a Large, Full-Text Information-Retrieval System," pp. 256-299 in *Advanced Database Machine Architecture*, ed. D.K. Hsiao, Prentice-Hall, Englewood Cliffs, New Jersey (1983).
10. King, D. R., "The Binary Vector as the Basis of an Inverted Index File," *J. Lib. Autom.* 7(4), p. 307 (1974).
11. Lesk, M.E., "Some Applications of Inverted Indexes on the UNIX System," UNIX Programmer's Manual, Bell Laboratories, Murray Hill, New Jersey (1978).
12. Lin, Z. and C. Faloutsos, "Frame Sliced Signature Files," CS-TR-2146 and UMIACS-TR-88-88, Dept. of Computer Science, Univ. of Maryland (Dec. 1988).
13. Rothnie, J.B. and T. Lozano, "Attribute Based File Organization in a Paged Memory Environment," *CACM* **17**(2), pp. 63-69 (Feb. 1974).
14. Salton, G. and M.J. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill (1983).
15. Schuegraf, E.J., "Compression of Large Inverted Files with Hyperbolic Term Distribution," *Information Processing and Management* **12**, pp. 377-384 (1976).
16. Zipf, G.K., *Human Behavior and Principle of Least Effort: An Introduction to Human Ecology*, Addison Wesley, Cambridge, Massachusetts (1949).