Inverted Files Versus Signature Files for Text Indexing

JUSTIN ZOBEL
RMIT
ALISTAIR MOFFAT and KOTAGIRI RAMAMOHANARAO
The University of Melbourne

Two well-known indexing methods are inverted files and signature files. We have undertaken a detailed comparison of these two approaches in the context of text indexing, paying particular attention to query evaluation speed and space requirements. We have examined their relative performance using both experimentation and a refined approach to modeling of signature files, and demonstrate that inverted files are distinctly superior to signature files. Not only can inverted files be used to evaluate typical queries in less time than can signature files, but inverted files require less space and provide greater functionality. Our results also show that a synthetic text database can provide a realistic indication of the behavior of an actual text database. The tools used to generate the synthetic database have been made publicly available.

Categories and Subject Descriptors: E.5 [**Files**]; H.2.2 [**Database Management**]: Physical Design; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; I.7.3 [**Document and Text Processing**]: Index Generation

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Indexing, inverted files, performance, signature files, text databases, text indexing

1. INTRODUCTION

There are two principal indexing methods—inverted files and signature files—that have been proposed for large text databases. Both remain the subject of active research. However, although many researchers have evaluated the performance of one method or the other, there has been no

This work has been supported by the Australian Research Council, the Centre for Intelligent Decision Systems, and the Collaborative Information Technology Research Institute.

Authors' addresses: J. Zobel, Department of Computer Science, RMIT, GPO Box 2476V, Melbourne 3001, Australia; email: jz@cs.rmit.edu.au; A. Moffat, K. Ramamohanarao, Department of Computer Science, The University of Melbourne, Parkville, Victoria 3052, Australia; email: {alistair, rao}@cs.mu.oz.au.

detailed side-by-side comparison. Indeed, the absence of a comparison has meant that the question as to which is better has been a popular topic of debate in our research institute, in which, in the context of text indexing, several of us have proposed variations on both of these indexing schemes [Sacks-Davis et al. 1987; Kent et al. 1990; Bell et al. 1993; Moffat and Zobel 1996]. The debate has been further added to by the steady flow of papers we have been asked to review in which the authors espouse or improve upon one of these forms of indexing without regard for the existence of the other.

To resolve the question of which method is superior we have undertaken a detailed examination of inverted files and signature files, using both experimentation on realistic data and a refined approach to modeling of signature files. Our conclusion is that, for current architectures and typical applications of full-text indexing, inverted files are superior to signature files in almost every respect, including speed, space, and functionality. To allow others to reproduce our results, and to establish a benchmark against which they may in turn compare their methods, we have made available on the Internet our inverted file text database system and one of the databases used in our tests, a synthetic database developed expressly for experimentation.

We also used an actual test database in the comparison, consisting of over 250 megabytes of English-language articles from *The Wall Street Journal*; data which, for copyright reasons, cannot be distributed. The results on the two test databases were, however, remarkably similar, and a useful byproduct of our investigation is the observation that performance on a synthetic database can be a fairly accurate predictor of performance on an actual collection.

Text databases, the query classes we consider, and the test data we use for comparison are described in Section 2. Application of inverted files and signature files to text indexing is explained in Section 3, and in Section 4 we compare these schemes with respect to query evaluation speed, using direct argument, mathematical modeling, and experiment. Some other points of comparison are considered in Section 5, and in Section 6 we consider the likely performance of the two indexing methods for other applications. Conclusions are presented in Section 7.

2. TEXT DATABASES

We assume that a *text database* is a collection of *documents*. Each document is uninterpreted prose—that is, is not divided into fields—and is of arbitrary length. Thus a database is a large number of records, each of which is simply a list of words. Such a database might be used for abstracts, legal transcripts, or, as in our main test data, newspaper articles. Practical text databases will often include other fields in each record, such as date of creation; our definition of text database can be regarded as being records in which such auxiliary information has been removed, leaving only the part to be indexed by content.

2.1 Boolean Queries

There are two main forms of query for text databases, Boolean and ranked. In this investigation we have chosen to focus on Boolean queries, that is, the class of queries that can be constructed from query terms, disjunction ($\bigvee$), and conjunction ($\bigwedge$). A document d is an answer to a conjunctive query $t_1 \bigwedge t_2 \bigwedge \ldots \bigwedge t_q$ if it contains every t_i for $1 \leq i \leq q$; and d is an answer to a disjunctive query $t_1 \bigvee t_2 \bigvee \ldots \bigvee t_m$ if it contains any t_i for $1 \leq i \leq m$. Conjunction and disjunction can be nested to arbitrary depth. A typical form of a query to a text database is a conjunction of q disjunctions each of m_i terms, that is, of the form

$$(t_{11} \bigvee \ldots \bigvee t_{1m_1}) \bigwedge \ldots \bigwedge (t_{q1} \bigvee \ldots \bigvee t_{qm_q}).$$

This form is useful because queries are commonly used to identify documents referring to every one of a set of concepts; the conjunction expresses that each of the concepts must be present. In turn a concept is represented by several terms (often thesaural equivalents or variant forms of the same word) any one of which may be present; each disjunction expresses that any of the terms by itself represents the concept.

In English text, the use of variant word endings such as "ing" and "ed" mean that two words can be superficially distinct even though it is desirable that they be regarded as a match. This problem is addressed by *stemming* each word during index creation, that is, removing variant endings. Query terms should then also be stemmed. A widely used algorithm is that of Lovins [1968].

2.2 Other Query Types

The other main category of query for text databases is the ranked query, in which a similarity score is calculated between the query, which is a list of terms or even some sample text, and each document. Similarity is defined by a mathematical formula that approximates the likelihood that the record is an answer, and can involve a large number of parameters. The top-ranked documents according to this score are presented to the user as answers. Ranked queries are discussed further in Section 5.7.

A further class of query that might be considered in the context of text indexing is *proximity* queries, in which answers must contain the specified terms within a specified distance of each other. The special case of a proximity of 1 is known as *adjacency*, which is valuable because a term may only be of interest if it is part of a phrase. We do not use these query classes as a basis for quantitative comparison of the indexing techniques, but do discuss how well they can be supported by the different methods.

2.3 Experimental Collections

There are several large text collections to which we have access, in particular the TREC collection of 3 Gb of data [Harman 1992a]. TREC consists of several subcollections, including newspaper articles, patent

Table I. Statistics of Document Collections Used for Experiments

		WSJ	FIN
Collection statistics:	Size (megabytes)	266.8	233.7
	Number of records	98,732	100,000
	Distinct words	204,587	209,556
Record length (kilobytes):	Minimum	0.07	0.05
	Average	2.77	2.39
	Maximum	78.66	13.79
Words per record:	Minimum	5	7
	Average	432	430
	Maximum	8,023	2,496
Distinct words per record:	Minimum	5	6
	Average	228	265
	Maximum	2,257	1,091
Distinct words per record after removal of common terms:	Minimum	4	1
	Average	150	116
	Maximum	1,945	637

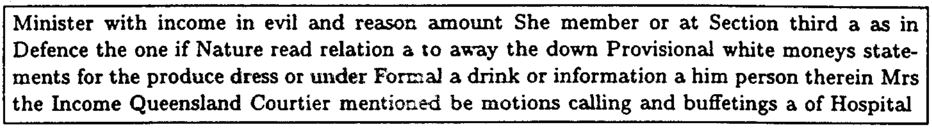
applications, journal articles, and abstracts. For our experiments we have used the WSJ collection, which consists of two and a half years of articles from *The Wall Street Journal*. Some statistics of this collection, and of the synthetic database FIN described in the following, are shown in Table I. As a collection of one kind of document from a single source, we believe WSJ to be a fair test of a text-indexing technique.

One aim of this work is to produce results that others can readily reproduce. Our inverted file text database system, described later, is available via ftp, but we cannot distribute the TREC data. Nor would it be feasible to fetch such large quantities of data over the Internet. We therefore implemented a database generator, FINNEGAN, that can create databases of any size with similar statistical properties to real text. The inputs to FINNEGAN are the file LEX, which is the combined vocabularies of a large number of books available on the Internet, together with the occurrence count of each word, and a distribution of record lengths measured in number of term occurrences per record. Record lengths were generated by the gamma distribution, which has parameters α and β; for integral $\alpha > 0$ and $t > 0$, the distribution is specified by

$$F(t) = \frac{t^{\alpha-1}e^{-t/\beta}}{\beta^{\alpha}(\alpha - 1)!},$$

where $F(t)$ is the probability that a record will have exactly t terms. We used $\alpha = 3$ and $\beta = 144$, so that the mean number of term occurrences per record is $\alpha\beta = 432$, approximately the number observed in WSJ.[1]

[1] FINNEGAN, LEX, and the query generator QUANGLE described in the following, are available via Internet from `ftp://munnari.oz.au/pub/finnegan`. This directory also contains the queries as described in the following, and the scripts used to create databases and run experiments.



Minister with income in evil and reason amount She member or at Section third a as in Defence the one if Nature read relation a to away the down Provisional white moneys statements for the produce dress or under Formal a drink or information a him person therein Mrs the Income Queensland Courtier mentioned be motions calling and buffetings a of Hospital

Fig. 1. A sample of the synthetic document database FIN.

A sample of FIN is shown in Figure 1. The difference in size between WSJ and FIN is due to the lack of markup and punctuation in the latter. It should be noted that FIN is by no means a substitute for a real database. It cannot be used to examine the effectiveness of ranking algorithms, for example, nor would it be a good subject for semantic analysis. But it is a useful basis for performance comparison, and, as our results show, provides a good indication of actual performance; and it is free of any copyright.

2.4 Query Generation

The TREC data has an associated set of queries, but these are primarily designed for ranked query evaluation. Queries 51 to 75 have been approximated into a Boolean form, originally for use by a "soft Boolean" query evaluator; these are the QUEENS queries [Kwok et al. 1992]. But we have no further source of actual queries, nor would actual queries be likely to have answers on FIN. Thus we chose to generate synthetic queries. In the absence of either a distribution for query terms (which is unlikely to resemble the distribution of terms in text—textually common words such as "the" are improbable query terms) or a distribution of query lengths we have developed a somewhat simplistic query generator. However, as shown later, the performance trends for the artificial queries are like those of the QUEENS queries.

Input to the query generator QUANGLE is the vocabulary of the database (except for 601 stopwords: frequently occurring words such as "the" and infrequently occurring closed-class words such as "furthermore") and the number of records f_t containing each term t. Also input are the number N of records in the database, the number A of answers desired per query, the number q of conjuncts to be generated, and the number m of disjuncts per conjunct. Queries are generated by QUANGLE as follows. First, q frequencies $F_1, \ldots, F_q$ are chosen so that $\Pi_{i=1}^{q}(F_i/N) \approx A/N$. Then m further frequencies $F_{i1}, \ldots, F_{im}$ are chosen for each F_i so that $\Sigma_{j=1}^{m} F_{ij} \approx F_i$. Finally, for each F_{ij} a term t_{ij} with $f_{t_{ij}} \approx F_{ij}$ is chosen from the vocabulary. The output query is

$$(t_{11} \vee \ldots \vee t_{1m}) \wedge \ldots \wedge (t_{q1} \vee \ldots \vee t_{qm}).$$

The frequencies F_i and $F_{i,j}$ were chosen according to a uniform distribution. It may be that the frequencies of real query terms are not distributed in this way, but we believe this method of query construction does not favor either inverted file or signature file indexing, and thus provides a fair test.

FIRST	Celtic
SECOND	Ms ∧ George ∧ double
THIRD	(doubling ∨ task ∨ Pickens ∨ Iraqi) ∧ (leased ∨ negotiated ∨ Lloyd ∨ integrity) ∧ (CHM ∨ targets ∨ distributes ∨ acquires)
FOURTH	payment ∧ impact ∧ Intermedics ∧ receive ∧ infringement
FIFTH	FOREIGN ∧ South ∧ All ∧ CNG
QUEENS	(weather ∨ lightning ∨ avalanche ∨ tornado ∨ typhoon ∨ hurricane ∨ heat ∨ wave ∨ flood ∨ snow ∨ rain ∨ downpour ∨ blizzard ∨ storm ∨ freezing) ∧ (fatal ∨ death ∨ die ∨ killed ∨ dead ∨ injuries ∨ fatalities)

Fig. 2. Sample queries for WSJ.

For each collection we used six sets of queries. The first three query sets were constructed using QUANGLE. In the first set, FIRST, each of the 25 queries was designed to have one conjunct and approximately 10 answers; each term in this query set is relatively rare in the database. In the second set, SECOND, each of the 25 queries was designed to have three conjuncts and approximately 10 answers; each term in this query set is thus fairly common in the database. In the third set, THIRD, each of the 25 queries was designed to have three conjuncts of four disjuncts each and approximately 10 answers; the query terms in this set range from rare to common. Note that in these first three query sets QUANGLE was restricted in its choice to only select terms that occurred in fewer than 1 record in 20 of the database, for reasons given in the following. Given this threshold, the average number of distinct query-eligible terms remaining in each document is shown in the final section of Table I.

The last three query sets were constructed differently. In the set FOURTH each of the 25 queries is a list of five distinct words randomly selected from a single database record, with 25 documents scattered uniformly in the collection used as the seed records. This generation method guarantees that each query has at least one answer. In the fifth set, FIFTH, there is one query, a conjunct of the most common indexed terms; the conjunct is the longest list, starting from the most common indexed term, that had an answer. The last query set is QUEENS, from which we removed common words to ensure that queries could be evaluated via the index.

A query from each set is shown in Figure 2. Statistics for the query sets are shown in Table II. Of these sets, SECOND and THIRD and of course QUEENS are most typical of real queries.

3. INVERTED FILE INDEXES AND SIGNATURE FILE INDEXES

There are two principal indexing methods, and a wide range of variations thereof, that are suitable for large text databases: *inverted files* and *signature files*. In this section we describe the two methods, particularly

Table II. Statistics of Query Sets and Numbers of Answers per Query

Query	Number of	WSJ			FIN		
set	queries	Min	Av	Max	Min	Av	Max
FIRST	25	1	9	61	1	9	39
SECOND	25	7	25	71	3	11	29
THIRD	25	10	32	103	4	12	18
FOURTH	25	1	1	2	1	1	1
FIFTH	1	11	11	11	2	2	2
QUEENS	25	0	414	4,946	0	141	2,056

with respect to their application to text, and consider some of their drawbacks.

3.1 Inverted Files

An inverted file index [Fox et al. 1992] has two main parts: a search structure or *vocabulary*, containing all of the distinct values being indexed; and for each distinct value an *inverted list*, storing the identifiers of the records containing the value. Queries are evaluated by fetching the inverted lists for the query terms, and then intersecting them for conjunctive queries and merging them for disjunctive queries. To minimize buffer space requirements, inverted lists should be fetched in order of increasing length; thus, in a conjunctive query, the initial set of candidate answers are the records in the shortest inverted list, and processing of subsequent lists only reduces the size of this set. Once the inverted lists have been processed, the record identifiers must be mapped to physical record addresses. This is achieved with an *address table*, which can be stored in memory or on disk.

An effective structure for storing vocabularies is a B$^+$-tree. The high branching factor typical of these trees means that the internal nodes are only a small percentage of the total vocabulary size. For example, suppose that in a B$^+$-tree leaves contain pointers to inverted lists, that the vocabulary of some database contains 1,000,000 distinct 12-byte terms, and that the disk being used operates with 8-kilobyte blocks and 4-byte pointers. Then at most 64 kilobytes are required for the internal nodes. Given this much memory, at most one disk access is required to fetch a vocabulary entry. Since the exact address of the inverted list is then known, a second access suffices to retrieve the corresponding inverted list. Other structures that are suitable for storing vocabularies include arrays and hash tables, with comparable performance.

3.2 Compressed Inverted Files

The inverted lists themselves are sequences of record identifiers, sorted to allow fast query evaluation. Sorting of identifiers within inverted lists has another important benefit: the identifiers can be represented using variable-length codes that, for large text databases, compress the index by a factor of about six [Bell et al. 1993], to around 5 to 10% of the data size. This approach has the disadvantage that inverted lists must be decoded as they are retrieved, but such decompression can be fast. Moreover, by inserting a small amount of additional indexing information in each list a large part of the decompression can be avoided, so that on current hardware the limiting factor is transfer time, not decompression time [Moffat and Zobel 1996]. Indeed, the performance achieved by our implementation of inverted lists was one of the factors that spurred this investigation, and we assume inverted lists to be compressed throughout this article.

An interesting feature of compressed inverted lists is that the best compression is achieved for the longest lists, that is, the most frequent terms. In the limit (which, in the case of text indexing, is a term such as "the" that occurs in almost every record) at most one bit per record is required. There is thus no particular need to eliminate common terms from the index: the decision as to whether to use the inverted lists for these terms to evaluate a query can be made, as it should be, at query evaluation time.

There are several widely held beliefs about inverted files that are either fallacious, or incorrect once compression of index entries is taken into account:

—the assumption that sorting of inverted lists during query evaluation is an unacceptable cost (illusory, because inverted lists should be maintained in sorted order);

—the assumption that a random disk access will be required for each record identifier for each term, as if inverted lists were stored as a linked list on disk (They should be stored contiguously or at the very least in a linked list of blocks.);

—the assumption that, if the vocabulary is stored on disk, log N accesses are required to fetch an inverted list, where N is variously the number of documents in the collection or the number of distinct terms in the collection (only true if none of the nodes in the tree storing the vocabulary can be buffered in memory; moreover the base of the log is usually large, perhaps 1,000, so the true cost is one or at most two disk accesses);

—the assertion that inverted files are expensive to create (Previous work in our research group has shown this to be fallacious [Moffat 1992; Moffat and Bell 1995], and the experiments reported in the following confirm that they are cheaper to build than signature file indexes.);

—the assertion that inverted files are large, an oft-repeated claim being that they occupy between 50 and 300% of the space of the text they index [Haskin 1981] (With current techniques inverted files are stored in around 10% of the space of the text they index [Witten et al. 1994].).

3.3 Bitstring Signature Files

In signature file indexes [Faloutsos 1992], each record is allocated a fixed-width signature, or bitstring, of w bits. Each word that appears in the record is hashed a number of times to determine the bits in the signature that should be set, with no remedial action taken if two or more distinct words should happen (as is inevitable) to set the same bit. Conjunctive queries are similarly hashed, then evaluated by comparing the query signature to each record signature; disjunctive queries are turned into a series of signatures, one per disjunct. Any record whose signature has a 1-bit corresponding to every 1-bit in the query signature is a potential answer. Each such record must be fetched and checked directly against the query to determine whether it is a *false match* (a record that the signature indicates may be an answer, but in fact is not) or a *true match*. Again, an address table is used to convert record numbers to addresses.

3.4 Bitslice Signature Files

To reduce query-time access costs, the set of signatures can be transposed into a set of *bitslices*, so that a signature file contains one fixed-length slice for each bit position in the original string; the length of each slice is the number of records being indexed, that is, N bits for a database of N records.

For bitsliced signature files, then, conjunctive query evaluation consists of: hashing the query into a signature; for some or all of the 1-bits in the query signature, fetching the corresponding bitslice; ANDing the fetched slices together to form a bitmap of potential answers; and, for each 1-bit in the bitmap, retrieving the corresponding record and checking whether it is a true match. If each bitslice is sufficiently sparse, and the hash function used to set the bits sufficiently randomly, only a few false matches will remain after a subset of the bitslices has been processed. At this point it may be cheaper to cease processing of bitslices and start retrieving and checking records. That is, the number of bit slices actually fetched in a multi-term query might be only a little larger than the number of slices processed for a single-term query.

For a given bitsliced signature file index, the minimum number of bitslices that should be processed for a conjunctive query is fixed at s, typically in the range 6 to 8 [Sacks-Davis et al. 1987; Kent et al. 1990]. The index will have been created with parameters chosen to ensure that processing s bitslices will, in a probabilistic sense, reduce the number of false matches to a specified level. If the number q of query terms in a conjunctive query exceeds s, then at least q slices should be fetched, since otherwise one or more of the query terms play no part in the selection of answers. For a query of m disjuncts, at least sm slices are required, since these queries are processed as m independent queries and the answer sets merged. As for inverted files, either one or two disk accesses are then required to fetch each answer, depending on whether the address table is in memory.

For text indexing, an application in which queries might have as few as one term, signatures are formed by letting each word in the record set s bits. To keep the number of false matches to a manageable level, signature width is such that each bitslice is fairly sparse. For example, Kent et al. [1990] suggest that, to achieve good overall performance, approximately one bit in eight should be set. Note that false-match checking can be expensive in document databases, as it involves fetching a record (and thus either one or two disk accesses), parsing the record into words (often including stemming each word), and then evaluating the query directly against the list of terms. Fast query processing is thus only possible if the number of false matches is kept low.

Combining these recommendations for signature density and s means that each distinct word in each record requires a notional space in the signature of about 50 bits, roughly the length of an average-length word represented in ASCII. This allows an initial estimate of signature file size—if each distinct term appears on average twice per record, then about 25 bits per word occurrence are required by the index, corresponding to approximately 50% of the space occupied by the input text. Note that standard bitstring signature files are claimed to be substantially more compact than this [Faloutsos 1992], since there is no disk access penalty for having a higher bit density. In fact, as shown in the following, the difference is negligible; for a given false-match rate bitstring signature files are only slightly smaller than bitsliced signature files. Moreover, query processing costs are much greater, since the entire index must be scanned to determine candidate answers.

As for inverted file indexes, one processing heuristic is to select slices in increasing density, so that sparse slices are preferred to dense. Implementation of this technique requires that each slice be tagged with a density indication, which must be stored separately from the slice itself if the number of disk accesses is to be kept small. The selection process must also use knowledge of which query term corresponds to each bit in the query signature, since nothing is gained if all of the sparse slices correspond to the same query term.

3.5 Blocked Signature Files

A particular problem with bitsliced signature files is of scale: databases with large numbers of records have long slices, several of which must be retrieved in full regardless of the properties of the query. That is, despite that fact that the index is stored, transposed, and only a few bit positions in each signature must be inspected, index processing costs are guaranteed to rise linearly in the size of the database. For example, a database of one million records would have bitslices of one megabit each, and processing of even the simplest of queries would require transfer of approximately one megabyte of index data. This problem can be addressed by grouping records into blocks, so that each bit in each slice corresponds to B records, where B is the blocking factor. Slice length is reduced by a factor of B; to keep slice densities low, signature width must be increased by a similar factor. To reduce the potential for block-level false matches (in which a block contains all the query terms, but no record in the block is a match) a different mapping from record number to block number can be used in each slice. Thus a record may be in block 6 in the first slice, block 11 in the second slice, block 9 in the third slice, and so on. The number of different mappings is K, the number of slices (i.e., signature width) is a multiple of

K, and each mapping is applied to w/K slices. This *multi-organizational scheme* [Kent et al. 1990] reduces but does not eliminate the potential for block-level false matches.

Queries are evaluated as for conventional bitsliced signature files, except that the bitslices must be decoded into record numbers as they are retrieved. It is possible to intersect bitslices by having a list of record numbers that are potential matches, and to use each subsequent bitslice to eliminate records that are not matches, by analogy with the query evaluation mechanism for inverted files; but the large number of records (about one in eight) that each slice implies is a match, and the disorder in each slice due to the complexities of the mappings, make this approach undesirable. It is also possible to decode each blocked bitslice into a full bitslice, then directly AND the decoded slices, but the cost of decoding is high and dominates query evaluation time.

These problems are addressed in the ATLAS [Sacks-Davis et al. 1995] text database system as follows. The number of mappings K is set equal to s, the number of bits set per term, and the index is divided into K partitions, each of w/K slices. Then each term is allocated one slice in each partition, thus guaranteeing that each term uses all of the available mappings. The first few mappings (typically three) are identical, so that the same blocking scheme is used in the first few partitions. Then, when queries are evaluated, the first slices fetched are from these partitions because such slices can be ANDed without further processing. Assuming that the selected slices are sufficiently sparse, the result of this operation is a slice of length N/B with a relatively small number of 1-bits. For each of these 1-bits, the corresponding record numbers are computed and the remaining slices probed to determine whether they have 1-bits for these records. The main disadvantage of this approach is that there is less flexibility in slice selection: it decreases the likelihood of being able to use a sparse slice to reduce the number of candidates. During the development of ATLAS, this *K-block* approach was experimentally found to increase the number of block-level false matches, but greatly improved query response time. It is the implementation we assume in this article.

The larger the blocking factor, the more slices must be fetched to eliminate block-level false matches. Thus increasing the blocking factor increases the number of disk accesses, but reduces the amount of data to be transferred. It follows that choice of parameters needs to take actual disk characteristics into account.

3.6 Compression of Signature Files

Compression brings considerable benefit to inverted file indexes, and it is natural to ask if the same improvements can be achieved with signature files. The answer is no.

By inverted file standards each slice in a bitslice of blocked signature file is very dense, with approximately one bit in eight set. The entropy of a binary probability distribution [1/8, 7/8] is 0.55 bits per symbol, and so compression of such an index roughly halves the space. However the need to decompress a set of slices means that query processing becomes much slower. For example, in the WSJ database the average compressed inverted list is 105 bytes long; but the average compressed bitslice is over 6,000 bytes long. Even in a blocked signature file significantly more decompression time will be required than by the corresponding inverted file index. And the use of compression erodes one of the advantages of a signature file, namely, that no "slice address table" is required when all of the slices are of the same length. Instead, the address of each variable length slice must be maintained in an address table, a small extra use of main memory.

Moreover, although we have noted previously that decompression time for compressed inverted lists can be fully offset by reduced seek and transfer times, a similar tradeoff is not observed for compressed bitslices because of the poor compression ratio. That is, the reduction in the volume of data transferred provides a window of time for decompression; only if the volume is greatly reduced will this window be large enough for decompression to take place. The required size of the window is, however, dependent on architecture: as the gap in speed between disk and processor grows, the necessary window size increases.

3.7 Records of Varying Lengths

In an inverted file index records of widely differing lengths pose no special problems, since long problems merely result in index entries in a relatively large number of the inverted lists.

The same is not true for signature files. To maintain even density and thus good performance for signature file indexes it is important that the records being indexed be of similar length. If signature width is set to cater to the longest record, index size will be unacceptable. On the other hand, if signature width is suited to average records (or even to all but 10% of the longest records), then the signatures for the longest records will have almost all of their bits set. In WSJ, for example, average record length is about 2.8 kilobytes but the longest is 79 kilobytes, and the chance of having more than a handful of 0-bits in the signature for this record is small. It follows that the longest records will be identified as potential answers to a high proportion of queries and will be retrieved and checked; and it is quite possible that the time taken to retrieve and false-match check the longest candidates will exceed all other query processing costs combined. In essence, long records remove the independence assumption of 1-bits in bitslices—a 1-bit in some position of a bitslice is indicative of a long record, and hence the conditional probability that bits in the same position of other slices will also be set is higher than might otherwise be expected.

The difficulties presented by databases of records of widely varying length is a well-known drawback of signature files [Faloutsos 1985; Kent et al. 1990], but the gravity of the problem has not, to our knowledge, previously been recognized. We analyze the impact of variable-length records on false-match rates in Section 4.

Methods for coping with databases of widely differing record length have been outlined but not, to our knowledge, developed or implemented. One suggested solution is to partition the database into subdatabases of records of roughly similar length, each subdatabase having its own index [Kent et al. 1990]. Since the signature width would vary from subdatabase to subdatabase, each index must be processed separately, multiplying the number of slices to be fetched and making query processing time unacceptably high. Another proposed solution is to have very wide signatures but apply compression so that similar overall space is used [Faloutsos 1985]; but this solution is only suitable for bitstring signature files, which have the immediate disadvantage that the entire index must be retrieved and decompressed to answer a query. A third possible solution is to break the longer records into shorter fragments, but this requires the use of a mapping that converts signature number to record number; not only must this mapping must be applied to every bit when ANDing slices together, but also it has no beneficial effect when only one slice is fetched for each query term.

The problem of long records is less acute for blocked signature files, since each record is blocked in several different ways, signature width is much greater, and the blocking means that length variation is more constrained. Nonetheless a long record can still cause difficulties. This effect is quantified in Section 4. In TREC, the longest record is 2.52 megabytes, a thousand times larger than the average record. If this record is retrieved, the cost of fetching and checking it completely dominates query processing time. Unfortunately its length implies that it is the record most likely to be retrieved as a false match.

In summary, false matches and long records are a serious problem for signature files applied to text databases, and cannot be neglected.

3.8 Handling of Common Terms

Common terms (those that occur in a high proportion of records) pose a similar problem for signature file indexes, as all of the bitslices to which common terms hash will have a high proportion of 1-bits. That is, if a common term and a rare term share a bitslice, queries on the rare term will have an increased number of false matches because of the reduced filtering effect of the shared slice. Moreover, 100 (say) common terms generate 600 or 800 unusually dense slices, so it is clear that a large fraction of the slices in the index (which is typically a few thousand slices wide) are shared between rare terms and common terms. Common terms are a particular problem for the multi-organizational scheme. Consider a signature file in which 8 bits are set for each term, and a conjunctive query of four independent and equally likely terms, each with probability of 0.1; such a query would be expected to have 10 answers on WSJ or FIN. Then the probability of one term not appearing in any record is $1 - 0.1 = 0.90$. In a block of 8 documents, the probability of the term not appearing is $(0.90)^8 = 0.430$, and so the probability of the term appearing somewhere in the block is $1 - 0.430 = 0.570$. To process a query, 8 slices are fetched, and each bit in each slice is set with probability at least 0.570. Hence, the probability that any particular record is accessed as a match is at least $(0.570)^8 = 0.011$. That is, 1.1% of the database—over 1000 records in our test databases—might be accessed as false matches.

One solution to this problem is to not index common terms by applying an extended stoplist. This makes queries on other terms faster, but also requires that queries involving common terms be answered by scanning the database. In WSJ, for example, there are many terms that occur in a high proportion of the records (e.g., "Washington") that are certainly not stopwords in the usual sense and, despite their frequency, provide useful discrimination. Another solution is to have a separate vocabulary of common terms, each of which has a dedicated bitslice or inverted list [Kent et al. 1990; Sacks-Davis et al. 1987]. That is, a "signature file method" should incorporate an inverted file to handle common terms.

From the point of view of comparing inverted files and signature files, this suggestion is awkward, since there is no straightforward basis for partitioning the distinct terms of the database into "common" and "not common" and hence no clear boundary between the two indexing schemes. It might be proposed, for example, that the common terms be as many as will fit in a vocabulary of some predetermined size, so that, with sufficient memory for the vocabulary, no bitslices are required at all.

In the face of this ambiguity we make our position clear: we compare inverted files to the component of a signature file index that consists of slices in which bits are set by hashing. That is, we assume that all terms but those explicitly defined to be common are so indexed, and that common terms are simply not indexed at all. Then to avoid any suggestion that our experiments were biased in favor of inverted file indexes, we arbitrarily declared "common" all words that appeared in more than 5% of the documents of the test collections. These words were also excluded from the inverted file indexes and from all of the queries, including the QUEENS queries, in which some of the common words appeared.

As noted, common terms are handled particularly economically by compressed inverted files, and there is no need to stop them at all, since, even with every word and number indexed, index sizes are still typically about 10 to 15% of the source text size [Bell et al. 1993].

3.9 Choice of Parameters

In a compressed inverted file the main choices that must be made are of the structure of the vocabulary and of the mechanism used to compress the inverted lists. Experience has shown that a B^+-tree style index is appropriate for a very wide range of vocabulary sizes, and that several different compression methods all give excellent behavior on inverted lists [Bell et al. 1993]. That is, there is little need for parameter or process selection on the part of the database manager.

The situation is in contrast to that for a signature file index. It is necessary to choose the number of bits s per term, a signature width w, and, in the case of the multi-organizational scheme, a blocking factor K. Moreover, an incorrect choice of parameters can lead to unnecessarily poor performance in speed, space, or both. Indeed, tuning signature files for best results was an ongoing difficulty with the experiments described in the following, since parameters that worked well for one query set and signature file scheme tended not to work well for other combinations. A lesson we learned very quickly is that it is difficult to choose good signature file parameters without detailed knowledge of the type of queries to be handled.

3.10 Other Signature File Methods

One of the difficulties in the comparison of inverted files and signature files is that many variants of signature file techniques have been proposed, and it is possible that some combination of parameters and variants will result in a better method. But we believe that the methods considered here are at least as good as the best signature file techniques, and are fair representatives.

The principal alternatives to bitsliced signature files are partitioned signature files and hybrid organizations. The partitioned signature file [Zezula et al. 1991; Ciaccia and Zezula 1993; Ciaccia et al. 1996] is a modification of the bitstring method in which each signature is allocated to a bucket. This method should be most effective when the query signature has a large number of bits set. The results of Zezula et al. suggest that bitslice signature files are faster than partitioned signature files when the number of bits set in the query signature is less than 70 or so (the exact crossover point depends on factors such as the number of buckets), so for our test queries we would expect bitslice signature files to be superior. We are, however, not aware of any experimental comparison of partitioned signature file techniques to other indexing techniques on a large data set.

There are several hybrid schemes that aim to combine the benefits of signature files and inverted files. One approach, in which common terms are indexed by bitmaps and rare terms by a signature file [Kent et al. 1990; Sacks-Davis et al. 1987], was discussed previously; the motivation for this scheme is that, as we indicated, indexing of common words in a signature file organization increases the false-match rate. Another hybrid approach that has been advocated is a blend of bitmaps for common terms and variable-length posting lists for more discriminating terms [Faloutsos and Jagadish 1992]. A related approach is to blend signature files with posting lists [Chang et al. 1989]. The motivation for these schemes is that, for rare terms, inverted files have better performance than signature files because signature file costs are always linear in the size of the database, whereas for common terms a signature file or bitmap is more efficient than using a fixed number of bits per record pointer. However, although the cost of storing and retrieving posting lists for common terms is one of the major drawbacks of traditional inverted list implementations, compression eliminates this problem as a compressed posting list is never longer than the equivalent bitmap. That is, compression automatically provides a smooth transition from posting lists for rare terms to bitmaps for common terms, with no administrative intervention or controlling parameters.

3.11 Experimental Text Database Systems

We used two text database systems for the experiments in this article. The MG system [Witten et al. 1994; Moffat and Zobel 1996] uses compressed inverted files for indexing.[2] The vocabulary is maintained on disk and is accessed through a small main-memory index, and the address table is also held on disk. Hence query processing requires two disk accesses per query term, and two disk accesses per answer.

The system used for the signature file experiments supports several signature file indexing schemes, including bitstring, bitslice, and blocked bitslice. It also incorporates a data analysis tool that chooses suitable parameters for signature width and so on, according to the criteria discussed by Kent et al. [1990]. It maintains its address table on disk.[3] Both of these systems were developed jointly at RMIT and the University of Melbourne.

Both of our database systems support storage and querying of compressed text, but, since the compression schemes are not compatible, in this article we use both in uncompressed mode, to ensure that the same amount of data is being fetched and to allow direct comparison of retrieval times. Note that for many typical classes of query the use of compression can actually decrease retrieval time through reduced seek and transfer costs [Zobel and Moffat 1995]. Similarly, both systems use Lovin's stemming algorithm [Lovins 1968], but there were small differences in the implementations that led to different documents, and different numbers of documents, being retrieved on some of the queries. To avoid this inconsistency we turned off stemming in both systems. This change is to the benefit of signature files, since with a stemmed index each record selected by the signature file must be completely restemmed during false-match checking, a nontrivial requirement.

We note the existence of other public retrieval systems based upon inverted files: the SMART system developed at Cornell University by Salton and others [Salton and McGill 1983], and the GLIMPSE system developed at the University of Arizona by Manber and others [Manber and Wu 1994]. Also, most commercial text retrieval systems and Internet search engines

[2] MG is available from `ftp://munnari.oz.au/pub/mg`. A tutorial introduction to MG appears as an appendix in Witten et al. [1994].

[3] The system actually used was ATLAS [Sacks-Davis et al. 1987, 1995; Kent et al. 1990], a nested relational system with text support. To avoid the problems caused by common terms, ATLAS makes use of dedicated bitslices (a form of compressed inverted list) for terms that appear in sufficiently many of the documents that they should not be indexed through signatures. To allow the comparison, we disabled this feature, thereby achieving a "pure" signature file implementation, and, as noted, terms that appeared in more than 5% of the records were declared to be "common," and neither indexed nor queried in either database system. ATLAS is not publicly available.

employ inverted files. We are unaware of any public domain text retrieval systems that use signature files.

4. COMPARISON OF INVERTED FILES AND SIGNATURE FILES

In this section and the next we compare inverted files and signature files. This section focuses on query evaluation speed, using direct argument, modeling, and experiment; in Section 5 we compare other aspects of the two methods, such as disk space and extensibility. We assume that processing is carried out on a uniprocessor workstation; the issue of parallelism is explicitly addressed in Section 5.4. We also assume that disk transfers have a fixed nontrivial startup cost to fetch the first byte of accessed data, but that subsequent bytes fetched in the same read operation can be transferred quickly, and that these costs are independent of file size. This latter assumption is an approximation as all our experiments have been run on a UNIX system, in which large files are organized as a tree of index and data blocks, and random access to a byte of a large file can be several times slower than random access to a byte of a small file.

Note that our description of the relative merits of inverted files and signature files has been entirely based on application of these techniques to document indexing. Some of the following arguments and results may have broader implications for the relative performance of these techniques, but we make no claims with respect to other applications.

4.1 Direct Argument

Inverted file indexes with in-memory search structures require no more disk accesses to answer a conjunctive query than do bitsliced signature files. This can be seen from the following inductive argument, an elaboration of a claim presented by Zobel et al. [1992].

Initially, all of the records of the collection are candidate answers to the query. The query is resolved by fetching bitslices, in the signature file case, or inverted lists, in the inverted file case. Now an inverted list can be regarded as a representation of a bitslice—a list of the ordinal positions of documents with a 1-bit. So we can refer to "bits set in an inverted list", meaning bits set in the bitslice represented by an inverted list; and we can "AND together inverted lists", meaning that we form their intersection. We can also refer to "a word's bitslices", meaning the slices corresponding to a bit set by that word.

For each word in a query, there can be bits set in the word's signature file bitslices that are not set in the word's inverted lists, but the converse does not hold; the word's inverted list is never denser than any of its signature file bitslices. Thus, for any bitslice that can be selected to AND with the list of candidate records in a signature file index, an inverted list that is at least as sparse can be selected in the corresponding inverted file index (and if the signature file bitslice is for a word that has already been processed in the inverted file case, then no action is required). For a query involving q words, the selection of q inverted file entries is sufficient to guarantee that no false matches remain in the inverted file case; but after q bitslices have been processed in the signature file case there can still be candidate records that are false matches (and possibly a large number if q is small, less than six, say). These false matches can be resolved only after further disk accesses, either to retrieve more bitslices or to retrieve text.

From the statistics of our text collections it can be seen just how much more dense are bitslices than inverted lists. For standard bitslices and WSJ, signature width is around 10,000 bits, and each word sets bits in eight slices; but each word sets bits in only one of over 100,000 inverted lists. It is thus virtually impossible for all of the bits in any bitslice to correspond to a single term.

If, in the signature file case, only $p < q$ slices are retrieved before false-match checking begins (on the assumption that it will be cheaper to retrieve and check false matches than to continue processing with the index), then in the inverted file case the same "short-circuit" evaluation strategy can be adopted, and a false-match checking phase can likewise be commenced after p inverted file entries have been merged.

Provided that inverted file entries have been selected for merging in order of increasing length, the number of records to be retrieved and false-match checked in the inverted file case cannot exceed the number of records accessed in the signature file case. Hence, the only situation when the number of disk accesses in the inverted file index can exceed the number of disk accesses with a signature file index is when the inverted file vocabulary cannot be held in memory. Such an arrangement gives rise to q "unaccounted for" disk accesses, one for each of the query terms. In this situation the signature file can perform fewer disk accesses than the inverted file if q is larger than the number of false matches (or, if the document address table is held on disk, the number of false matches divided by two). This is most likely to happen when q is large, since fetching a large number of bitslices also reduces the false-match probability. For typical small values of q (less than three or four) the inverted file index performs fewer accesses, even if the vocabulary is held on disk.

In practice, the relative performance of signature files and inverted files is affected by other factors, in particular, the lengths of bitslices and inverted lists. An unblocked bitslice is always at least as long as the longest compressed inverted list, so performance cannot be better. Blocked bitslices typically have 1 bit per 4 to 32 records, and with the latter ratio blocked WSJ bitslices are just a few hundred bytes long. The inverted lists for WSJ vary in length, from a few bytes to about 3 kilobytes (recall that common terms have been removed), with an average of 105 bytes. A typical query might be expected to favor inverted lists—one imagines that users try to specify highly selective terms—but it is also conceivable that blocked signature files transfer less data and are therefore faster.

However, even for the complete TREC collection, which is an order of magnitude larger, the average compressed inverted list occupies just a few hundred bytes. To reduce the false-match rate to a manageable level signature file methods must examine some minimum number of slices, even for single term queries. For nonconjunctive queries this effect further favors inverted files. A disjunction of m terms is processed by a signature file index as m single-term queries, and then the answer lists merged; this means that on the order of sm slices must be fetched if the false-match rate is to be reasonable.

For conjunctive queries, it is questionable whether queries can have large numbers of terms. Our experience is that it is unlikely that, for text, a conjunctive query of even 10 terms will have an answer in the first 2 Gb of the TREC collection, unless the terms are words such as “the” and “of”. For example, if query terms appear randomly in 5% of the records, the threshold previously assumed, then 5 words provide a selection rate of about 1 in $(20)^5 = 3{,}200{,}000$. In real queries, terms may be semantically related and thus the queries may have more matches, but even so the supposition that conjunctive queries can have large numbers of terms is at best dubious. Ranked queries, discussed in more detail in the following, can have large numbers of terms, but are not conjunctive.

Once matches have been identified, retrieval of those matches is strictly cheaper in the inverted file case than in the signature file case because of the need for false-match checking of the records returned by the signature file index. Thus, even if there are no false matches for a given query, fewer disk accesses in the signature file case, and less data transferred (an unlikely combination of circumstances in view of the preceding arguments), query evaluation can nonetheless be slower for signature files because the absence of false matches cannot be known until every true match has been checked.

4.2 Mathematical Modeling—Signature Files

In this section we describe a mathematical model for bitsliced and multi-organization signature files, and show in particular that violation of one constraint on the use of signature files (that records should be of near-equal length) can be catastrophic. This model can be used to derive signature file size from expected false-match rates and a distribution of record lengths.

Suppose a record in a database has t distinct terms to be indexed, that s signature bits are set for each term, and that the signature is w bits wide. That is, the signature for the document is formed by setting st randomly chosen bits (with replacement) out of w. After this, the probability of a randomly chosen bit in the signature being set to one is given by

$$1 - \left(1 - \frac{1}{w}\right)^{st}.$$

A single-term query will cause s bits in the signature of each document to be inspected, and if all s bits are one, the record must be processed as a false match. Hence, the probability $p(w, s, t)$ that a nonanswer record of t terms is inspected as a false match is given by

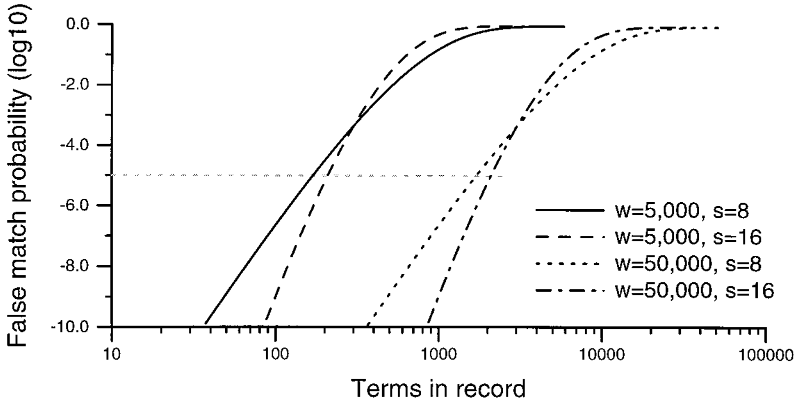


Fig. 3. False-match probability for $w = 5{,}000$ and $w = 50{,}000$, and $s = 8$ and $s = 16$; single-term queries.

$$p(w, s, t) = \left(1 - \left(1 - \frac{1}{w}\right)^{st}\right)^{s}.$$

Using this expression, we can calculate, for example, that with $s = 8$ bits set per term and exactly $t = 150$ distinct noncommon terms per record (the average for WSJ), a false-match rate of 1 in 100,000 is achieved when $w = 4{,}443$.

Figure 3 shows the variation in $p(w, s, t)$ as a function of t for four different pairs of parameters w and s. Note the steep increase in false-match probability as t increases; note also that $s = 16$ allows slightly more terms per record than $s = 8$ for some given false-match probability, but with a much greater penalty when records have more than the expected number of terms. The horizontal dashed line corresponds to a false-match rate of 1 in 10^5.

Suppose further that the distribution of record lengths (where length means the number of distinct indexed terms) is governed by probability density function $F(t)$. For example, one assumption that might be made during the analysis of signature file methods is that all records in the collection are of some length l, corresponding to the function

$$F_E(t) = \begin{cases} 1 & \text{if } t = l \\ 0 & \text{otherwise} \end{cases}$$

Given a density function $F(t)$, the probability $M(F, w, s)$ that a random nonanswer document is accessed as a false match is given by

$$M(F, w, s) = \sum_{t=1}^{\infty} p(w, s, t) \cdot F(t).$$

If $L(t)$ is the ratio of total term occurrences to distinct terms in a record of t distinct terms (e.g., taking an average over WSJ we might suppose that

Table III. False-Match Rates $M(F, w, s)$ for Density Functions F_E and F_W and $s = 8$; Single-Term Queries

w	F_E (equal)	F_W (actual)
3,164	1.0×10^{-4}	58×10^{-4}
4,443	1.0×10^{-5}	138×10^{-5}
6,142	1.0×10^{-6}	308×10^{-6}

$L_E(t) = 432/150 = 2.87$), then the expected volume $V(F, w, s)$ of false-match checking is given by

$$V(F, w, s) = \sum_{t=1}^{\infty} p(w, s, t) \cdot F(t) \cdot L(t).$$

These are both "per random nonanswer record" expected values. The expected number of false matches per single-term query in a collection of N documents is given by

$$(N - a) \cdot M(F, w, s) \approx N \cdot M(F, w, s),$$

where a is the number of true matches, and the approximation is appropriate when $a \ll N$. Likewise, the expected volume of text (measured in terms) that must be checked in a single-term query is approximated by

$$N \cdot V(F, w, s).$$

As a more realistic example, suppose that $F_W(t)$ is the fraction of records in WSJ containing exactly t distinct terms (after the removal of common terms), and that $L_W(t)$ is the average term multiplicity over t-term documents in WSJ. Table III shows some values of $M(F, w, 8)$ for F_E and F_W. As can be seen, the difference between the "equal" length distribution F_E and the "actual" length distribution F_W increases the false-match rate more than a hundredfold.

Moreover, if a signature file index is used for variable length records, the false-match records are highly likely to be long. Figure 4 plots predicted false-match volume V as a function of w, the signature width, for the WSJ collection. The two outermost lines (labeled "Equal-length" and "BS") show false-match text volume $V(F, w, s)$, calculated using F_E and F_W, respectively. Over a large range of w the actual expected volume of false-match records is more than a thousand times greater than the amount predicted by the "equal-length" assumption embodied in F_E. For example, a bitsliced WSJ index that checks 432 false-match terms per query (the length of an average WSJ record) requires a signature of $w = 19{,}600$ bits, very much larger than the index of 4,443-bit signatures indicated by the original F_E-based analysis summarized in Table III.

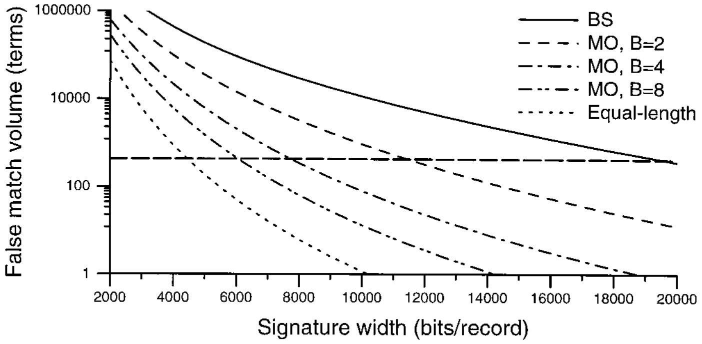


Fig. 4. Predicted false-match volume as a function of w for WSJ and $s = 8$; single-term queries.

Note that this model can also be applied to single-term queries on bitstring signature file indexes. Evaluation of the model indicates that the minimum signature width for a bitstring index on WSJ with a false-match rate of 1 in 10^{-5} is at least 12,200 bits (this minimum occurs when $s = 9$, and is only a few bits less than the value of w required for $s = 8$ for the same false-match rate), giving an index that is around 50% of the size of the indexed data, a marked contrast to the 10% that has been claimed elsewhere [Faloutsos 1985]. This somewhat surprising result is caused by the conflict between the need for large values of s if w is to be reduced, and the steepness of the false-match penalty for long records when s is large (Figure 3).

The situation with multi-organization indexes is more complex. By blocking records together and widening the signature accordingly, variations in record lengths are, to a certain extent, smoothed out. Indeed, the ability of multi-organization signature files to cope with routine vagaries of record length is a strong point in their favor. Nevertheless, false-match rates and false-match volumes can still be high. If we assume that each term is indexed by a single bit in each of $K = s$ different organizations, that the signature width for a block of B records is w bits, that each organization is allocated a total of w/s bits, and, as before, that F is a density function on record lengths, then the nonanswer false-match rate for a single-term query is bounded above by

$$M_{\mathrm{MO}}(F, w, s, B)$$

$$= \sum_{t_1=1}^{\infty} \sum_{t_2=1}^{\infty} \sum_{t_3=1}^{\infty} \cdots \sum_{t_B=1}^{\infty} \left[\prod_{i=1}^{B} F(t_i) \right] \cdot \left[\prod_{i=1}^{s} \left(1 - \left(1 - \frac{s}{w} \right) \right)^{\sum_{i=1}^{B} t_i} \right]$$

$$= \sum_{t=1}^{\infty} \left[F^B(t) \cdot \left(p\left(\frac{w}{s}, 1, t \right) \right)^s \right],$$

where $F^B(t)$ is the probability density function that arises when B random variables are selected according to $F(t)$ and summed, the length of a block of B records, if it can be assumed that the terms in records are disjoint. In practice there will be overlap of terms in blocks, and so this estimate is pessimistic. A more precise calculation would use an actual density function for the number of distinct terms in blocks of B records.

Figure 4 plots the corresponding false-match volume function $V_{\mathrm{MO}}(F_W, w, s, B)$ for $B = 2$, $B = 4$, and $B = 8$, all with $s = 8$. Large values of B clearly reduce the false-match volume for single-term queries toward the level expected under an "equal length" assumption, but by no means eliminate the discrepancy.

On the other hand, blocked signature files handle multi-term queries relatively poorly, especially when B is large. The larger the block, the more likely it is that two query terms appear together in the block but not in any single record of the block. These *block-level* false matches add to the record-level false matches estimated by the preceding model.

In summary, the need for false-match checking, and the need to minimize false matches, is a serious problem for signature files, a problem for which inverted files have no equivalent.

4.3 Mathematical Modeling—Inverted Files

A mathematical model can also be applied to inverted file indexes, in this case to calculate an upper bound on index size. For inverted files, there are no considerations such as false matches and no parameters to vary, so estimation of size is straightforward.

One compression method that we have used for representing inverted files is Golomb coding [Golomb 1966; Bell et al. 1993]. Golomb codes have the interesting property that a set of p positive integers summing to at most N are represented in at most

$$G(N, p) \approx p \cdot \left(1.5 + \log_2 \frac{N}{p}\right)$$

bits. Moreover, the concavity of the log function means that the worst case is when the p values are all roughly equal, and on nonuniform distributions the number of bits required decreases. This bound can be immediately applied to the statistics reported in Table I to calculate an upper bound on index size for inverted lists stored as gaps: for WSJ, the index need not be larger than

$$G(98732 \times 204587,\ 98732 \times 150),$$

which is 21.0 megabytes, or 11.9 bits per pointer. This compares very favorably with the signature file sizes calculated previously. For example, the minimum width calculated (using the equal length assumption) was $w = 4{,}443$, which corresponds to an index of 52.3 megabytes, more than twice the size. More realistic signature widths produce even larger indexes.

If the WSJ false-match rate is to be 1 in 100,000, then $w = 12{,}204$ is required and the index grows to 143.6 megabytes, the equivalent of 80 bits per index pointer.

Moreover, the Golomb codes can be applied independently to the individual inverted lists, thus improving compression. Over a large range of databases we have found that Golomb codes consistently require about six bits per pointer.

It should also be noted that it is not necessary to decompress the whole of each index list to process conjunctive queries [Moffat and Zobel 1996]. If each compressed inverted list is partially indexed by the insertion of *skips* it is only necessary for a small fraction of each inverted list to be decoded, and for typical queries the dominant cost when processing each inverted list is the cost of transferring the list from disk to memory. The insertion of the skips does increase the length of each inverted list by as much as 10% of their compressed size, but in the context of an already small inverted index the actual cost is negligible.

4.4 Experiment

Our experimental framework is as follows. We have the two text database systems implemented within our research group over a period of several years, which we use to compare the performance of inverted files and signature files. In both cases we are confident that the implementation is of good quality, and both systems have inbuilt instrumentation allowing reliable measurement of query evaluation time. We also have our two test document collections WSJ and FIN and for each collection six sets of test queries. We built inverted file and signature file indexes for both collections on the same physical disk drive on the same machine. The hardware is a four-processor Sun SPARC 10 model 514 with 256 megabyte of memory. The operating environment was Solaris; both programs were written in C, and are uniprocessing, executing on a single processor and making no attempt to exploit parallelism. The experiments were carried out in an alternating manner, to minimize the likelihood of external factors disrupting the timings; and, as far as possible, the machine and disk drives were otherwise idle throughout the process. Each query was executed five times, with individual executions of each query separated by a flush of the operating system cache to ensure a cold start to the query process.

The inverted file indexes, denoted by IF in the results, were, as discussed previously, compressed. They were not, however, skipped, since we preferred to base our results upon a straightforward implementation of compressed inverted files.

Two sets of signature file indexes were created, in each case using $s = 8$, that is, 8 bits per term. In the first indexes, denoted by BS, the signature width was chosen to probabilistically yield one false match per query assuming the actual distribution of record lengths in that database. In the second set, denoted by MO, the multi-organizational scheme was used, with $K = 8$ and $B = 8$, and, as for BS, the actual record lengths were used; but

Table IV. Index Parameters (Signature Width, Size in Megabytes, Size as Percentage of Size of Original Data)

		WSJ			FIN		
		w	megabytes	%	w	megabytes	%
IF	Inverted file	—	16.2	6.1	—	14.5	6.2
BS	Bitslice	12,204	143.6	53.8	6,178	73.6	31.5
MO	Blocked bitslice	44,492	65.5	24.5	30,667	45.7	19.6

to make the analysis tractable we pessimistically assumed that no terms were duplicated between records, so the signature widths in this case are greater than an exact analysis would imply. In each case—IF, BS, and MO—words appearing in more than 5% of the documents were not indexed, nor were they used in queries. Statistics of these indexes are shown in Table IV.

In addition to the experiments described here we explored the performance of signature files, by varying parameters, to try to achieve the best possible performance and to test our models. In one set of experiments we chose signature width to probabilistically yield one false match per query assuming records of equal length; performance was as poor as our analysis had predicted. We also varied s, trying 6, 12, and 16, as well as the experiments with $s = 8$ reported here. Increasing s consistently led to slower query evaluation and only slightly smaller indexes; the value $s = 8$ was chosen as the best representative value for the full set of results. Another parameter we varied was the number of common words. This had surprisingly little effect on performance for bitslices (although index size grew rapidly as more words were indexed), but the presence of common words was disastrous for the multi-organizational scheme, with query response time deteriorating by a factor of hundreds when the number of common words was halved.

Based on the preceding analyses, we would expect the relative performance of signature files to be best on the query sets SECOND and FOURTH, since no disjuncts are involved and for signature files the query can be resolved (to a given level of false-match probability) in the same number of disk accesses as the query set FIRST.

Performance is shown in Tables V and VI. Table V shows the time spent retrieving and processing index information, that is, the time spent fetching and resolving inverted lists or bitslices in order to generate a list of possible or actual (for IF) answer documents. Table VI shows the total time (including the index processing times listed in Table V) spent retrieving answers, and, in the case of signature files, time spent retrieving, processing, and eliminating false matches. In this latter table the number of matches is shown in square brackets against each entry, and so the number of false matches can be inferred by subtracting the number of (true) matches recorded for the inverted file index.

Perhaps the most notable result is that, although signatures are in several cases slightly faster with respect to index processing time, for all but a few of the several hundred queries we tested inverted files completed

Table V. Index Processing Time (Elapsed Time; Seconds)

	WSJ		
	IF	BS	MO
FIRST	0.09	0.35	0.16
SECOND	0.21	0.38	0.28
THIRD	1.33	3.76	4.22
FOURTH	0.28	0.46	0.24
FIFTH	1.12	0.83	0.54
QUEENS	0.50	2.50	1.94

	FIN		
	IF	BS	MO
FIRST	0.08	0.33	0.15
SECOND	0.24	0.37	0.29
THIRD	1.32	3.60	4.11
FOURTH	0.27	0.43	0.20
FIFTH	1.14	0.76	0.59
QUEENS	0.43	2.40	2.18

Table VI. Query Evaluation Speed (Average Elapsed Time in Seconds) and Number of Documents Retrieved (i.e., Average False and True Matches; Shown in Square Brackets)

	WSJ		
	IF	BS	MO
FIRST	0.49 [9]	1.24 [11]	0.52 [9]
SECOND	1.16 [25]	4.11 [36]	5.69 [88]
THIRD	2.52 [32]	12.26 [92]	10.80 [109]
FOURTH	0.33 [1]	0.85 [2]	0.37 [1]
FIFTH	1.68 [11]	3.62 [19]	4.14 [54]
QUEENS	5.21 [414]	18.60 [433]	16.28 [430]

	FIN		
	IF	BS	MO
FIRST	0.46 [9]	0.64 [10]	0.42 [9]
SECOND	0.66 [11]	1.42 [19]	2.89 [72]
THIRD	1.77 [12]	7.37 [83]	7.84 [120]
FOURTH	0.32 [1]	0.52 [1]	0.29 [1]
FIFTH	1.24 [2]	1.22 [4]	1.36 [10]
QUEENS	2.82 [141]	6.86 [160]	6.54 [168]

processing faster, and often 10 times as fast, as did signature files, particularly on the more realistic query sets SECOND, THIRD, and QUEENS. Another general trend is the retrieval cost per document for each indexing method. In the inverted file case, between 30 and 60 documents are retrieved per second once the index processing is complete. In the signature file case, around 10 to 15 documents are retrieved per second, the slower rate being due to false-match checking. This discrepancy was accentuated when stemming was used: the number of documents processed per second fell to around 8.

As expected, query set FIRST is handled relatively well by each of the indexing methods. The inverted file index is extremely fast at processing queries involving a single rare term, but the signature file indexes are also acceptable.

For query set SECOND, the relative performance of BS and IF is like that of query set FIRST, but MO performs poorly, most probably because of the manner in which QUANGLE was driven to make use of relatively common words (most with probability of occurrence between 0.025 and 0.05) to achieve the desired number of answers, so that block-level false matches dominate. We have nonetheless reported the results we observed, having been told by advocates of signature files that inverted files are poor on queries involving a relatively large number of common terms. For query set SECOND the bitsliced signature file implementations do have reasonable index processing times compared to inverted files (Table V), but even so, the cost of false-match checking—almost regardless of the width of the index—makes overall query processing costs substantially higher (Table VI).

Query set FOURTH, with five terms per query, was designed to allow the signature file implementations to show their best relative performance, and this is evident in Table V. For both databases the MO index was faster than the IF index at determining answers, and on FIN was faster overall as well. Surprisingly this was not true for the BS index; the MO index clearly benefits from manipulating shorter slices.

Query FIFTH, containing the most common indexed terms, puts paid to the fallacy that inverted files are inefficient for queries containing common terms. They are slower than for rare terms, and the index processing cost is higher than for signature files. But common terms also bring out the worst in signature file indexes, and the inverted file indexes completed the processing of the query as fast as signature files.

Finally, we had expected that the two complex query sets, THIRD and QUEENS, would be handled poorly by the signature file system. The inclusion of disjunction increases false-match rates, since the probabilities are additive rather than multiplicative. But overall these query sets are not markedly worse than SECOND, although again the multi-organizational scheme performed particularly badly. Note that the signature file system always converts queries to disjunctive normal form. To be fair in our comparison, we forced MG to also convert queries to this form, which meant in effect that the queries in THIRD were evaluated as if they had $3 \cdot 4^3 = 192$ terms. This is why index processing costs were high. For unconverted THIRD queries to WSJ, the MG index-time component was 0.62 seconds per query, and for the QUEENS queries the corresponding time was 0.47 seconds.

One important conclusion to be drawn from our experiments is that signature file performance is much less predictable than inverted file performance. Given two terms occurring in the same number of documents of the same total length, with an inverted file the time to identify and fetch the matching documents containing either of these terms will be roughly the same; with a signature file the time can differ by a factor of hundreds, particularly if the terms are relatively rare.

Another important observation is that in the search for improved performance, proponents of signature files have widened signatures, to reduce "on-paper" false-match probabilities; shortened the length of the bitslices, to reduce transfer times; used noncompressed representations, to avoid decompression costs; and shortcircuited index processing, to avoid disk seeks. But a single crucial fact has been neglected: false-match checking is by itself sufficiently expensive to make signature-based schemes uncompetitive. From our experiments it is clear that the cost of a signature file should be measured primarily in terms of the volume of match processing (for true and false matches) required. The number of disk seeks, the number of slices transferred, and the length of the slices, are all relatively unimportant when it takes as long as 0.1 seconds to fetch and check each retrieved record.

5. FURTHER POINTS OF COMPARISON

In this section we compare inverted files and signature files on several other criteria, including disk space, memory requirements, ease of index construction, ease of update, scalability, and extensibility. The reader is referred to Zobel et al. [1996] for a justification of the various areas examined.

5.1 Disk Space

For inverted files, reported index sizes are from 6 to 10% of the indexed data [Bell et al. 1993]. These figures include storage of in-document frequencies (used in processing of ranked queries) and indexing of all terms, including stop-words and numbers. They do not allow for the space wastage that a B^+-tree implementation would imply, and such an implementation is necessary for a dynamic collection (rather than the packed representation that can be used for a static database). For a dynamic collection perhaps 14% of the indexed data size is an upper limit on index size. The cost of storing the vocabulary must be added to this, but for large databases the vocabulary comprises a small fraction of the stored system. The vocabulary of the 3 Gb TREC collection has just over one million distinct terms, and occupies less than 30 Mb even assuming 12 characters and a pointer per word and substantial space wastage—that is, less than 1% of the indexed data. A similar ratio applies to the 132-Mb Commonwealth Acts of Australia, whose vocabulary occupies around 1 Mb [Bell et al. 1993]. Hence, we can regard 15% of the indexed text as being a reasonable upper bound on the amount of space required by the index, even for a dynamic collection.

The space required by bitsliced signature files depends on the signature width, and on whether blocking is used. Quoted sizes are 25 to 40% of the indexed data [Kent et al. 1990; Sacks-Davis et al. 1987], a figure that is broadly borne out by our experiments (Table IV). Signature files do not require the storage of a vocabulary, and for a static collection the slices can again be densely packed. Dynamic indexes are, however, subject to the same problems as dynamic inverted file indexes, and slices must be overallocated to allow for growth. Unless slices are stored as linked lists of blocks on disk or there is extensive copying of the index, a similar 30 to 40% average space overhead must be accepted, taking the index size to around 30 to 55% of the text.

Note that the sizes reported in Table IV do not include the additional inverted lists that would (for both signature files and inverted files) be required for common terms; for WSJ, these additional lists would occupy about 2% of the space of the indexed data.

Inverted files have often been judged through reference to two papers published over a decade ago: Haskin [1981], which estimated that inverted files require 50 to 300% of the space required for the data; and Cárdenas [1975], which estimated that each term occurrence requires up to 80 to 110 bits of index. These papers no longer reflect the capabilities of inverted files, and should not be used as a basis of comparison, just as signature files should not be condemned on the basis of the performance of bitstring techniques.

5.2 Memory Requirements

In the context of text indexing, memory can be used for vocabulary or a part thereof, the address table, buffers for retrieved inverted lists or bitslices, working space for recording the current set of candidates (as a slice or as a set of document identifiers), and buffers for answer records.

We believe that today it would be rare for a machine to have insufficient memory to store the vocabulary of a database indexed by that machine. That is, for example, we would be surprised if a machine with a 1-Gb database were unable to hold 10 Mb in memory. However, there is no great disadvantage to holding some of the vocabulary on disk; as indicated previously, 1 Mb would in this case still be sufficient to allow an inverted list to be fetched with a single disk access. Nor, as the experiments indicate, do two disk accesses per inverted list appear to be an important cost. Furthermore, even for a database of several gigabytes the other buffers amount to less than 1 Mb. The one exception to this is the need to hold answers in memory: there must be room to hold the whole of the longest document if any postretrieval processing is to be carried out or if answers are to be presented in an order different from that in which they are fetched. For TREC, this buffer must be 2.52 Mb; for WSJ 80 Kb suffice. Signature files have the advantage of not requiring space for a vocabulary, but all other costs are comparable, and for both systems the performance in the preceding experiments could have been achieved in around 100 Kb in addition to buffers for answers.

5.3 Index Construction

Inverted file indexes and signature file indexes have similar requirements during index construction. The text must be processed in document number order, and a “transposed” matrix of information written to disk in bitslice or term order. The requirement that reasonable amounts of memory be available is the only constraint. For example, if enough main memory is available that the entire index can be held, then direct techniques (in which the skeleton of the index is sketched in memory and then filled in as documents are processed) can be used. Such an approach is obvious for signature files, where all of the slices are the same length, provided the number of documents is known in advance; and similar methods can also be used to generate compressed inverted files [Moffat 1992]. Direct techniques yield fast indexing performance. If there is insufficient main memory for the direct approach to be used, the text must be partitioned, and a sequence of partial indexes built. This is logically easier for signature files, since each bitslice is readily partitioned into segments of the same length, but is also possible for compressed inverted files [Witten et al. 1994]. Moreover, the generally smaller size of a compressed inverted index compared to a bitsliced or blocked signature file index means that it is more likely that the direct approach can be exploited. Sort-based methods for inverted index construction are also possible, and need not be expensive regarding time, memory space, or temporary storage space [Moffat and Bell 1995].

Index construction times for the two retrieval systems we tested are approximately 40 minutes for inverted files and 86 minutes for signature files. This does not include the cost of the data analysis phase for the signature file indexes, required since the data must be inspected to allow appropriate choice of parameters. A complete inspection of a 500-Mb database takes about 20 minutes. The signature file construction times are greater because first, each term in each record must be hashed eight times, or equivalently eight random numbers must be generated from a hash seed. In contrast, for inverted files the main operation per term is a table lookup. The second reason that signature files take longer to construct is simply that there is more data to be created and written, since the resultant index is larger. Hence, we would expect a compressed signature file to be somewhat faster to build than an uncompressed one, since fewer flushes to disk would be required for a given amount of memory.

5.4 Parallelism

Most current architectures provide some form of parallelism. Our experiments have considered queries running sequentially on a single-user machine but it is reasonable to consider whether in a parallel processing environment the relative performance of inverted files and signature files would change.

Consider how the respective index processing algorithms could be parallelized. One approach is through user parallelism, in which each query is processed sequentially but many queries are processed at once. Our experiments apply to such parallelism, since they measure resources in each case, and the use of resources would not be affected by parallel execution of other queries; recall that, aside from clearing the system cache prior to each full run of queries, we made no attempt to eliminate system caching during each full run of queries. This is the usual model for parallelism in a database: what is measured is not per-query elapsed time, but throughput.

The other approach is to take advantage of any parallelism in the query evaluation mechanism. However, in this respect neither mechanism has a particular advantage. Query evaluation involves three corresponding stages in each case: multiple fetches are required to retrieve and process lists or slices, these lists or slices must be intersected to identify candidate documents, and the documents must be retrieved and, if necessary, checked for false matches. The single relative advantage of signature files is that more distinct entities must be fetched and processed in the first and third stages. In an environment in which there are more processors than inverted lists or true matches (so that some processors would be idle during query processing on an inverted file) there may be a relative speed-up for signature files. Such a relative speed-up would be obviated by any user parallelism, and is only available because in the inverted file case there is less index information to be processed.

The remaining consideration is whether architectural changes resulting from parallelism might affect performance. There are several forms of parallelism that might apply. One is disk parallelism: use of a disk array, for example, to provide faster access to index information. The main effect of disk parallelism is to improve seek times and transfer rates (a well-configured disk array can, say, double data throughput) thus relatively increasing the cost of index decompression. Another form of parallelism is processor parallelism, which provides the opposite effect of relatively decreasing the cost of index decompression. Thus the question of the impact of parallelism depends on the choice of architecture.

However, the question of parallelism is to some extent not relevant: in the context of a multi-user system with user parallelism, the principal issue is the resources used to resolve each query. As our experiments unequivocally show, inverted files use less resources than do signature files.

5.5 Update

Inverted lists are of varying length, and in a dynamic system change in length as they are updated. This effect is exacerbated by the use of compression, since as little as one bit might be added to a given list during record insertion. Thus some care must be taken to manage the lists on disk. As was noted previously, when stored as the leaves of a B^+-tree-like structure, average space overheads are around 45% of the space required for the lists themselves (because B-trees have 69% average space utilization), which is unfortunate but certainly not a disaster in the context of the small initial space. Alternatively, direct space management can reduce the space overhead to just a few percent [Zobel et al. 1992, 1993]. However, in the interests of having an uncontroversial basis of comparison, we regard 45% as a fair indication of space overhead.

Update is also a problem in signature files. Bitstring signature files are easily extended as records are appended to the collection. However, in the absence of any buffering or batch update strategy, adding one record to a database indexed with a bitsliced signature file requires a disk access to each slice where the signature has a 1-bit. Moreover, although all of the slices are exactly the same length, there is still the need for careful disk management: either each slice must be stored as a chained list of disk blocks, or the entire index must be periodically copied onto a larger extent of disk. In the former case query processing will be slowed, as each slice may require multiple seeks; and in the latter case the index space needs to be overallocated at each expansion, again with an average overhead through each full expansion of around 45%. That each slice is of exactly the same length in bits is of no advantage, and the problems caused by dynamic databases are essentially the same for inverted indexes and for signature files.

Insertion of new records into the multi-organizational signature file index is particularly problematic, as the bits that must be changed are scattered throughout the index, meaning that there is little advantage in batching update. Moreover, if the expansion is persistent, the blocking factor must also be periodically revised and the index completely rebuilt. Modification of records presents even greater difficulties: to check whether a given bit in a slice should be 0 or 1, it is necessary to check every record that can set that bit.

For both signature files and inverted files update costs can be dramatically reduced by batching. A simple scheme that makes updates available immediately is to maintain a small in-memory index for new documents. This index can be periodically merged with the main index in a single pass through the structures on disk, so that the costs of the update are amortized into one bulk update operation. If it is not acceptable for the main index to be unavailable during the update pass, it can be implemented as a rolling merge in which the index is updated one chunk at a time; that is, the update is still single-pass but the pass is spread out to allow query processing to continue simultaneously.

Being more compact, inverted files are more efficient with regard to such update. An in-memory inverted file can index more documents than can an in-memory signature file, so that the update passes can be less frequent; and the whole of the on-disk signature file must be moved to accommodate change in slice length, whereas only the lists that are modified must be moved in the case of the inverted file. However, it is also true that fewer read and write operations are required to update the signature file because there are fewer slices than there are inverted lists.

5.6 Scalability

For a given query, the number of disk accesses to an inverted file or signature file index is independent of scale, so the asymptotic index processing cost is the amount of data to be transferred from the index. For signature files, therefore, index processing cost is linear in the size of the database regardless of the query, even unique-key retrieval requires that several complete slices be fetched from disk. In contrast, with inverted files average inverted list length may be sublinear in database size, because new records introduce new terms as well as additional occurrences of existing terms.

For signature files, accesses to the data are for both answers and false matches; for inverted files, the accesses are for answers only. Since the percentage of false matches is independent of the number of records, for signature files the cost of query evaluation is in all cases linear in database size. For inverted files, the worst case is linear in database size, but in the best case, key retrieval, the cost is almost constant. Thus inverted files should scale better than signature files.

5.7 Ranking

For a ranked query, typically a list of terms, a similarity heuristic is used to assign to each document in the collection a score relative to the query [Salton and McGill 1983; Salton et al. 1983; Croft and Savino 1988; Harman 1992b]. The top *r* documents according to this score are then presented as the answers to the query, where *r* is a parameter set by the user. Most of the successful similarity heuristics combine several different statistics in order to estimate how closely a given document matches the list of terms in the query, and are based upon a formulation often referred to as the TF*IDF method. These statistics include the probability that a random document will contain the given term (the IDF, or inverse document frequency component), and the number of times the term appears in this document (the TF, or term frequency component). The final statistic is a normalizing factor based upon some measurement of document "length," so as to enable the score of long documents to be discounted.

To evaluate a ranked query with an inverted file is straightforward. Each document pointer in each inverted list is augmented by the inclusion of the corresponding TF value. Since the TF values are usually small, compression is effective, and the typical average cost is one to two bits per pointer [Bell et al. 1993]. That is, an inverted index that includes TF values is still just 10% or so of the text that it indexes [Witten et al. 1994]. The vocabulary of the collection is similarly augmented by the addition of an IDF value for each term.

Processing a ranked query then consists of the following steps [Harman and Candela 1990]. First, an array of accumulators, one per document, is initialized to zero. Then, for each query term, the corresponding inverted list is fetched and processed. Processing a list consists of stepping through it, and, for each document pointer, multiplying the embedded TF value by the global IDF value applicable to this list, and adding this similarity contribution to the appropriate accumulator. Finally, when all of the terms have been processed, the accumulator values are normalized by the corresponding document lengths, and the top *r* accumulator values determined by a partial sort [Witten et al. 1994]. For typical ranked queries of 30 to 50 terms (the query is often a piece of text, essentially a request to "find the documents that are like this text") and a database of one or two gigabytes, ranked queries returning perhaps 20 to 50 documents can be evaluated using an inverted file in approximately 5 to 10 seconds [Moffat and Zobel 1996], roughly the same time as for a disjunctive Boolean query of the same terms.

Consider now how the same calculation might be undertaken with a signature file. First, a vocabulary structure must be added, in order for the IDF values for terms to be known. This is straightforward. Much more problematic, however, is the TF component, since the presence or otherwise of a term in a document is "smeared" over several bits in a signature file index. One solution to this difficulty proposed recently is to index terms for which TF $= i$ in the ith of a set of several distinct signature files [Lee and Ren 1996]. Then, when querying, the matching of a term's signature with a record's entry in the ith signature file is strong evidence that the term does appear in that document, with TF $= i$. Query processing is, however, extremely expensive, since slices in every one of the signature files must be checked for every one of the query terms; that is, for a query of q terms and indexes with s bits per term, up to sq slices must be accessed from every one of a large number of signature files, which in practice means that the entire index will be scanned for most queries. Furthermore, several of the problems already noted with regard to signature files are exacerbated by this structure, including the problems caused by variable record lengths, and the need to set a large number of parameters. In particular, a signature width must now be selected for as many signature files as there are distinct TF values in the document collection.

The only advantage of the arrangement proposed by Lee and Ren (compared to the application of signature files to Boolean queries) is that one can dispense with false-match checking. In a ranked query the list of answers presented to a user often contains surprising documents because of the complexity of the similarity calculation, and the return of extraneous documents because of false matches will, provided there are sufficiently few of them, have little effect upon a user's qualitative perception as to the usefulness of the system, and only limited impact upon the system's quantitative retrieval performance [Lee and Ren 1996]. False-match checking is, of course, not an issue for inverted files.

For these reasons it is clear that the ease and simplicity with which inverted file indexes can simultaneously support both Boolean and ranked queries make them the method of choice for such mixed query applications, and for this reason we have not included ranked queries in our experiments.

5.8 Extensibility

Signature files are not as easily extensible as inverted files. One of the main drawbacks of signature files is that they are essentially binary: they can identify which records have a given property, but nothing further. The standard use of signature files is to identify which records contain a conjunction of given terms, and we have already noted their relatively poor performance on disjunctive queries and for ranking. It is fairly straightforward to extend signature files to support term adjacency, that is, to record which documents contain which terms as adjacent pairs; this functionality can be provided by hashing each pair of terms to set one or two additional bits, and in the ATLAS database system this extension increases signature file size by only around 4% of the size of the indexed data [Sacks-Davis et al. 1987, 1995]. But extensions beyond this functionality are difficult.

There are also problems with signature files that limit their applicability. As previously discussed, long records present serious difficulties, so that signature files are best suited to databases of records of similar length, such as abstracts. Signature files cannot easily identify how many answers there are to a query without actually fetching them, since all candidates must be retrieved to eliminate false matches. Signature files cannot be used to resolve negations, queries to find records not containing a given term. For nontext applications, signature files have other limitations; in particular, the cost of key retrieval is linear in the database size, and range queries are not supported.

In contrast, extension of inverted files is straightforward. Within-record frequencies can be inserted between record identifiers in the inverted lists at a typical cost of around 2 to 5% of the size of the indexed data [Bell et al. 1993; Moffat and Zobel 1996]. Positional information (recording the word positions at which each term occurs in each record) can also be inserted, at a further cost of around 15 to 20% of original data size, allowing not just adjacency queries but general proximity queries. A word-level index allows query terms to be highlighted in answers without the need for a postretrieval scan to locate them again. Even when augmented with positional information, total inverted index size is still no more than that of a bitsliced signature file. Inverted files can be further extended to support queries on document structure [Linoff and Stanfill 1993].

Moreover, inverted files support a broader range of text query types. Inverted files are substantially faster for disjunctive queries, as they only require a fraction of the number of disk accesses, and are better able to support ranking. Because there is a vocabulary it is possible to support queries on stems, patterns, and substrings [Hall and Dowling 1980]. Finally, since inverted files do not have false matches, the number of answers can be determined without accessing the data.

6. OTHER APPLICATIONS

Given the weight of evidence against signature files for text indexing, it is interesting to ask if there is any application at all in which they would be the method of choice. As a result of our investigations we can offer only a guarded “maybe” to this question.

It is clear that uniform-length records are necessary if a signature file index is to be acceptably compact for a given false-match rate; and equally clear that the vocabulary must be rich if an inverted file index is to be large relative to the text size. Similarly, queries should be conjunctive, to avoid the problems of disjunction; and should have only a few answers, so that false-match checking costs cannot dominate. Query terms should be common, so that an inverted file index would have relatively poor performance.

Finally, all of these conditions should be guaranteed for all data and queries the system might be applied to, since signature files must be preparameterized and so are especially vulnerable to mismatch between the data that are anticipated and those that are actually stored.

One application that perhaps meets these constraints is a library catalogue. In a catalogue records tend to be short and of similar length, and the vocabulary of author names is likely to be large; in such an application inverted file size is increased by the large vocabulary, but inverted file evaluation times are fast because indexed terms are not common. In this case, therefore, signature files may become more nearly competitive in size, but less competitive in speed.

Another application that may suit signature files is multimedia data, that is, data such as images in which it is possible for each stored item to have many attributes and for many of the attributes to have only a few distinct values. Queries to such databases could involve large numbers of attributes, thus giving query signatures in which many bits are set, potentially providing an opportunity for alternative signature methods such as partitioned signature files to work well [Zezula et al. 1991]. However, it seems unlikely that the majority of queries will only involve common terms, and the linearity of query performance as the database grows remains a drawback. Signature files may be competitive to inverted files for such applications, but in the absence of well-defined query mechanisms for such data (currently an active area of research), index performance must remain a matter of speculation.

Moreover, note that the belief that signature files can index some kinds of data that cannot be indexed with inverted files is false. Both inverted files and signature files map index terms to records—if a term can be hashed to set a bit in a signature then it can be used to access a vocabulary. The problem of extracting index terms from multimedia data such as images and video is the same for both mechanisms.

7. CONCLUSIONS

Having participated in many informal discussions as to the relative merits of inverted files and signature files for text indexing, we felt it timely to use our collective experience to undertake a careful and thorough investigation. Our evaluation has two major components: detailed experimental comparison of implementations of both indexing schemes; and a refined model of false-match probability—the source of the worst inefficiencies of the signature file scheme—that shows that for document indexing the cost of false matches can be orders of magnitude greater than would be estimated by analysis undertaken in the absence of detailed information about the distribution of record lengths.

Our conclusions are unequivocal. For typical document indexing applications, current signature file techniques do not perform well compared to current implementations of inverted file indexes. Signature files are much larger; they are more expensive to build and update; they require that a variety of parameters be fixed in advance, involving analysis of the data and tuning for expected queries; they do not support proximity queries other than adjacency; they support ranked queries only with difficulty; they are expensive for disjunctive queries; they are highly intolerant of range in document length; their response time is unpredictable; they do not allow easy addition of functionality; they do not scale well; and, most important of all, they are slow. Even on queries expressly designed to favor them, signature files are slower than inverted files. The current trends in computer technology, in which the ratio of processor speed to disk access time is increasing, further favor inverted files.

As a byproduct of our investigation, we have demonstrated that a synthetic database can be a valuable tool in experimental research. The results obtained for FIN closely mirrored those of the real database WSJ, and in making FINNEGAN and QUANGLE publicly available we hope that we have provided a resource that others will find of use in their work.

ACKNOWLEDGMENTS

We are grateful to Alan Kent for help with the signature file system, Tim Shimmin for help with MG, and Despina Psaradellis for programming assistance. We are also grateful to Ron Sacks-Davis, and to the anonymous referees for their valuable feedback.

REFERENCES

BELL, T., MOFFAT, A., NEVILL-MANNING, C., WITTEN, I., AND ZOBEL, J. 1993. Data compression in full-text retrieval systems. *J. Am. Soc. Inf. Sci. 44*, 9 (Oct.), 508–531.

CARDENAS, A. 1975. Analysis and performance of inverted data base structures. *Commun. ACM 18*, 5 (May), 253–263.

CHANG, J., LEE, J., AND LEE, Y. 1989. Multikey access methods based on term discrimination and signature clustering. In *Proceedings of the ACM-SIGIR International Conference on Research and Development in Information Retrieval* (Cambridge, Mass., June). ACM Press, New York, 176–185.

CIACCIA, P. AND ZEZULA, P. 1993. Estimating accesses in partitioned signature file organisations. *ACM Trans. Inf. Syst. 11*, 2 (April), 133–142.

CIACCIA, P., TIBERIO, P., AND ZEZULA, P. 1996. Declustering of key-based partitioned signature files. *ACM Trans. Database Syst. 21*, 3 (Sept.), 295–338.

CROFT, W. AND SAVINO, P. 1988. Implementing ranking strategies using text signatures. *ACM Trans. Office Inf. Syst. 6*, 1, 42–62.

FALOUTSOS, C. 1985. Access methods for text. *ACM Comput. Surv. 17*, 1, 49–74.

FALOUTSOS, C. 1992. Signature files. In *Information Retrieval: Data Structures and Algorithms*, W. Frakes and R. Baeza-Yates, Eds., Prentice-Hall, Englewood Cliffs, NJ, Chapter 4, 44–65.

FALOUTSOS, C. AND JAGADISH, H. 1992. Hybrid index organizations for text databases. In *Proceedings of the Third International Conference on Extending Database Technologies* (Vienna, Austria, March). A. Pirotte, C. Delobel, and G. Gottlob, Eds., LNCS 580, Springer-Verlag, Berlin, 310–327.

FOX, E., HARMAN, D., BAEZA-YATES, R., AND LEE, W. 1992. Inverted files. In *Information Retrieval: Data Structures and Algorithms*, W. Frakes and R. Baeza-Yates, Eds., Prentice-Hall, Englewood Cliffs, NJ, Chapter 3, 28–43.

GOLOMB, S. 1966. Run-length encodings. *IEEE Trans. Inf. Theor. IT-12*, 3 (July), 399–401.

HALL, P. AND DOWLING, G. 1980. Approximate string matching. *ACM Comput. Surv. 12*, 4, 381–402.

HARMAN, D., ED. 1992a. *Proceedings of the TREC Text Retrieval Conference* (Washington, D.C., November). National Institute of Standards Special Publication 500-207.

HARMAN, D. 1992b. Ranking algorithms. In *Information Retrieval: Data Structures and Algorithms*, W. Frakes and R. Baeza-Yates, Eds., Prentice-Hall, Englewood Cliffs, NJ, Chapter 14, 363–392.

HARMAN, D. AND CANDELA, G. 1990. Retrieving records from a gigabyte of text on a minicomputer using statistical ranking. *J. Am. Soc. Inf. Sci. 41*, 8, 581–589.

HASKIN, R. 1981. Special purpose processors for text retrieval. *Database Eng. 4*, 1, 16–29.

KENT, A. J., SACKS-DAVIS, R., AND RAMAMOHANARAO, K. 1990. A signature file scheme based on multiple organisations for indexing very large text databases. *J. Am. Soc. Inf. Sci. 41*, 7, 508–534.

KWOK, K., PAPADOPOULOS, L., AND KWAN, K. 1992. Retrieval experiments with a large collection using PIRCS. In *Proceedings of the TREC Text Retrieval Conference* (Washington, D.C., November). D. Harman, Ed., National Institute of Standards Special Publication 500-207, 153–172.

LEE, D. AND REN, L. 1996. Document ranking on weight-partitioned signature files. *ACM Trans. Inf. Syst. 14*, 2 (April), 109–137.

LINOFF, G. AND STANFILL, C. 1993. Compression of indexes with full positional information in very large text databases. In *Proceedings of the ACM-SIGIR International Conference on Research and Development in Information Retrieval* (Pittsburgh, Pa., July). R. Korfhage, E. Rasmussen, and P. Willett, Eds., ACM Press, New York, 88–97.

LOVINS, J. 1968. Development of a stemming algorithm. *Mech. Transl. Comput. 11*, 1–2, 22–31.

MANBER, U. AND WU, S. 1994. GLIMPSE: A tool to search through entire file systems. In *Proceedings of the USENIX Winter 1994 Technical Conference* (San Francisco, Calif., January). 23–32.

MOFFAT, A. 1992. Economical inversion of large text files. *Comput. Syst. 5*, 2 (Spring), 125–139.

MOFFAT, A. AND BELL, T. 1995. In-situ generation of compressed inverted files. *J. Am. Soc. Inf. Sci. 46*, 7 (Aug.), 537–550.

MOFFAT, A. AND ZOBEL, J. 1996. Self-indexing inverted files for fast text retrieval. *ACM Trans. Inf. Syst. 14*, 4 (Oct.), 349–379.

SACKS-DAVIS, R., KENT, A. J., AND RAMAMOHANARAO, K. 1987. Multikey access methods based on superimposed coding techniques. *ACM Trans. Database Syst. 12*, 4 (Dec.), 655–696.

SACKS-DAVIS, R., KENT, A., RAMAMOHANARAO, K., THOM, J., AND ZOBEL, J. 1995. Atlas: A nested relational database system for text applications. *IEEE Trans. Knowl. Data Eng. 7*, 3 (June), 454–470.

SALTON, G. AND MCGILL, M. 1983. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York.

SALTON, G., FOX, E., AND WU, H. 1983. Extended Boolean information retrieval. *Commun. ACM 26*, 11, 1022–1036.

WITTEN, I., MOFFAT, A., AND BELL, T. 1994. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Van Nostrand Reinhold, New York.

ZEZULA, P., RABITTI, R., AND TIBERIO, P. 1991. Dynamic partitioning of signature files. *ACM Trans. Inf. Syst. 9*, 4, 336–369.

ZOBEL, J. AND MOFFAT, A. 1995. Adding compression to a full-text retrieval system. *Softw. Pract. Exper. 25*, 8 (Aug.), 891–903.

ZOBEL, J., MOFFAT, A., AND RAMAMOHANARAO, K. 1996. Guidelines for presentation and comparison of indexing techniques. *ACM SIGMOD Rec. 25*, 3 (Oct.), 10–15.

ZOBEL, J., MOFFAT, A., AND SACKS-DAVIS, R. 1992. An efficient indexing technique for full-text database systems. In *Proceedings of the International Conference on Very Large Databases* (Vancouver, B.C., Canada, September). L.-Y. Yuan, Ed., 352–362.

ZOBEL, J., MOFFAT, A., AND SACKS-DAVIS, R. 1993. Storage management for files of dynamic records. In *Proceedings of the Australian Database Conference* (Brisbane, Australia, February). M. Orlowska and M. Papazoglou, Eds., World Scientific, Singapore, 26–38.

Received July 1995; revised December 1996; accepted December 1997