Fast Text Access Methods for Optical and Large Magnetic Disks: Designs and Performance Comparison.

Christos Faloutsos [1]
Raphael Chan
Department of Computer Science
University of Maryland
College Park, MD 20742

ABSTRACT

High capacity disks, especially optical ones, are commercially available. These disks are ideal for archiving large text data bases. In this work, we examine efficient searching techniques for such applications. We propose a unifying framework, which reveals the similarities between signature files and an inverted file using a hash table. Then, we design methods that combine the ease of insertion of the signature files with the fast retrieval of the inverted files. We develop analytical models for their performance and we verify it through experimentation on a 2.8 Mb data base. The agreement between theory and experimentation is very good. The results show that the proposed methods achieve fast retrieval, they require a modest 10%-30% space overhead, (as opposed to 50%-300% overhead [13] for the inverted files), and they do not require re-writing; thus, they can handle insertions easily, they permit searches during an insertion and they can be used with write-once optical disks. Using our verified model, the performance predictions for the proposed methods on large data bases (e.g., 250 Mb) are very promising.

1. Introduction

Text retrieval methods have attracted much interest recently [3, 11, 12, 27, 30]. One of the main reasons is probably the development of optical disks. Text databases are traditionally large and have archival nature: there are insertions in them, but almost never deletions and updates. These two characteristics make the optical disks the ideal storage medium. Optical disks have high recording capacities (of the order of 1 Gigabyte for 12'' platters), low cost (≈300 dollars for a 12'' disk) and long archival life (at least 10 years [10]). The proposed methods compete well on all classes of optical disks: the read-only CD-ROMs, the write-once-read-many (WORM) ones, and the erasable ones. They also compete on the traditional magnetic disks, as well. We shall examine the WORM ones with more emphasis, because they are the only commercially available optical disks that can be written upon, and because the write-once restriction creates interesting problems, that the traditional access methods (e.g., B-trees) can not solve efficiently. Notice that the write-once restriction creates no problems for text databases, exactly because there are seldom deletions and updates. There are numerous applications involving storage and retrieval of textual data:

- Electronic office filing [30], [3].
- Computerized libraries. For example, the U.S. Library of Congress has been pursuing the "Optical Disk Pilot Program" [21], [18], where the goal is to digitize the documents and store them on an optical disk. A similar project is carried out at the National Library of Medicine [29].
- Automated law [14] and patent offices [15]. The U.S. Patent and Trademark Office has been examining electronic storage and retrieval of the recent patents on a system of 200 optical disks.
- Electronic storage and retrieval of articles from newspapers and magazines.
- Consumers' data bases, which contain descriptions of products in natural language.
- Electronic encyclopedias [4], [12].
- Indexing of software components to enhance reusability [26].

All of the above applications require an efficient search method. This is exactly the problem addressed in this work. We opt for the following characteristics:

1) The method should introduce a small space overhead.
2) The method should be fast, requiring a few disk accesses to respond to simple queries (e.g., single word ones).

---

[1] Also with University of Maryland Institute for Advanced Computer Studies (UMIACS).

This research was sponsored partially by the National Science Foundation under the grant DCR-86-16833.

3) The method should not require re-writing.

There are two advantages, if a method requires no re-writing of the index structure:

- The method can handle insertions easily, without the need to shut down the system. Even at the presence of (at most) one writer, readers can be allowed to continue searching the data base. The idea is that the readers can ignore the newly written items, which will **not** overwrite the old items in the data base, exactly because no re-writing is necessary. This is very important in archival environments with large insertion and query frequencies, such as the U.S. patent office [15].
- The method can work more easily on a WORM disk: Every change on the index will not require re-writing, which is impossible in WORMs.

In fact, WORM disks have the additional restriction that, once a page is written, it can not be changed, even if it is almost empty. Methods that require "append-only" operations can by-pass the problem by keeping on a magnetic disk those few pages of the index or the text file that are not completely full; the full pages can be dumped on the optical disk. Notice that methods that require dynamic rearrangements of the index, such as a B-tree inverted index, can not easily take advantage of such an arrangement, because they will require a large portion of the (already large) index to reside on the magnetic disk.

Signature files [8, 9] satisfy the insertion and space requirements, but may be slow for large data bases. In order to accelerate their search time, we examine bit-sliced storage of the signature file; in addition, we propose compression of each bit-slice, which achieves even better search times. Based on these ideas, we propose a family of methods that satisfy all of the above design goals, and we present analytical and experimental results for their performance.

The paper is organized as follows: In section 2 we describe the problem and its parameters. In section 3 we examine briefly older text retrieval methods, with emphasis on the signature files. In section 4 we propose a family of methods, based on compressed bit-sliced signature files and describe them in detail. In section 5 we provide the space-time analysis of the proposed methods. In section 6 we present experimental results on a 2.85 Mb data base, as well as the expected performance of the proposed methods on a large data base, based on our analysis. In section 7 we present the conclusions and future research directions.

2. Problem definition.

The general problem is defined as follows:

P1:

Given the description of the data base, (size in bytes, number of documents e.t.c.)

Find the best method, as well as the optimal values for the design parameters

To solve the above, the following sub-problem has to be solved:

P2:

Given - the description of the data base,
- the search method
- the (not necessarily optimal) values for the design parameters

Find - the performance of the method (space, response times e.t.c.)

Tables T2.1, T2.2 and T2.3 list the input, design and output parameters respectively. The input parameters describe the data base, the output parameters measure the performance of a given method in a specific setting, and the design parameters are to be chosen by the designer, to optimize the performance. Notice that we use a stop-list of 150 words - the most common English words, such as "the", "a", etc. This saves space, without penalizing the retrieval performance.

The **false drop probability** F_d is the probability that the signature of a non-qualifying document will qualify.

$$F_d = \frac{\text{false drops}}{N - \text{actual drops}}$$

We examine **single word** queries in this paper. General Boolean queries will be examined in the near future.

3. Survey.

Text retrieval methods form the following large classes [5]: Full text scanning, inversion, and signature files, which we shall focus next. Signature files work as follows: The documents are stored sequentially in the "text file". Their "signatures" (hash-coded bit patterns) are stored sequentially in the "signature file". When a query arrives, the signature file is scanned sequentially and many non-qualifying documents are discarded. The rest are either checked (so that the "false drops" are discarded) or they are returned to the user as they are. The method is faster than full text scanning but slower than inversion [22] on large data bases. It requires much smaller space overhead than inversion (≈10% [2], as opposed to 50%-300% that inversion requires [13]) and it can handle insertions easily.

Signature files typically use superimposed coding [17] to create the signature of a document. A brief description of the method follows; more details are in [5]. For performance reasons, that will be explained later,

Symbol	Definition
N	total number of documents
L	average length of a document in bytes
D′	number of distinct words per document (document vocabulary)
D	number of distinct NON-COMMON words per documents
N_w	total number of words in the whole collection of documents
V	total number of DISTINCT words in the collection of documents (vocabulary of the collection)
P	size of a disk page (=block) in bytes (typically: 1 Kb)
b	bits per byte (typically: 8)

Table T2.1
Input parameters (description of the data base).

Symbol	Definition
F	size of the document signature in bits
S	size of the hash table
B_p	size of a bucket for the postings lists in bytes
p	size of a pointer (4 bytes or less)
B_i	size of a bucket for the intermediate level (see the last two methods) in bytes

Table T2.2
Design parameters

Symbol	Definition
O_V	space overhead of the method
F_d	False drop probability (single word queries)
$d_{R,s}$	disk accesses on successful search
$d_{R,i,s}$	disk accesses on the index only (successful search)
$d_{R,u}$	disk accesses on unsuccessful search
$d_{R,i,u}$	disk accesses on the index only (unsuccessful search)
d_I	disk accesses on insertion of a document
A	number of actually qualifying documents

Table T2.3
Performance measures ("output parameters").

each document is divided into "logical blocks", that is, pieces of text that contain a constant number D of distinct, non-common words. Each such word yields a "word signature", which is a bit pattern of size F, with m bits set to "1", while the rest are "0" (see Figure F3.1). F and m are design parameters. The word signatures are OR-ed together to form the block signature. Block signatures are concatenated, to form the document signature. The m bit positions to be set to "1" by each word are decided by hash functions. Searching for a word is handled by creating the signature of the word and by examining each block signature for "1"'s in those bit positions that the signature of the search word has a "1".

Word	Signature
free	001 000 110 010
text	000 010 101 001
block signature	001 010 111 011

Figure F3.1.
Illustration of the superimposed coding method.
It is assumed that each logical block
consists of D=2 words only.
The signature size F is 12 bits. m=4 bits per word.

For the rest of this work, the above method will be called **SSF**, for Sequential Signature File. Figure F3.2 illustrates the file structure used: In addition to the text file and the signature file, we need the so-called "pointer file", with pointers to the beginnings of the logical blocks. One of the bits of a pointer can be used as a flag, to indicate whether the corresponding logical block belongs to a different document than the previous logical block.

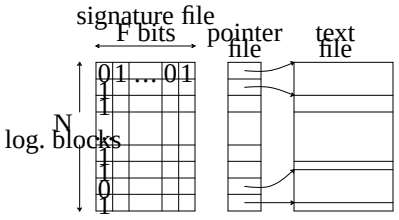


Figure F3.2
File structure for SSF

The signature file is an F×N binary matrix. Previous analysis showed that, for a given value of F, the optimal value of m is such that this matrix contains "1"'s with probability 50% [28]. This is the reason that documents have to be divided into logical blocks: Without logical blocks, a long document would have a signature full of "1"'s, and it would always create a false drop. To avoid unnecessary complications, for the rest of the discussion we assume that all the **documents span exactly**

one logical block.

4. Proposed methods

4.1. Framework

All of the forthcoming methods start from the signature file, which can be thought of as a bit-matrix, abstractly. The differences among the methods are in the way this matrix is stored (row-wise or column-wise) and whether compression is used or not.

storage / compression	row-wise	column-wise
No (m>=1)	seq. sig. files (SSF)	bit sliced sig. files (BSSF)
Yes (m=1)	VBC	compressed bit slices; CBS, DCBS, NFD (≈ inversion, with hash table)

Table T4.1
Proposed framework for text retrieval methods with false drops.

The classification of Table T4.1 encompasses all the text retrieval methods that allow false drops. The SSF method is naturally in the entry for row-wise storage, without compression. The VBC method (Variable Bit-block Compression) [9] suggests row-wise storage of the bit matrix, followed by compression of each row. There, it was showed that, whenever compression is applied, the best value for m is 1. VBC achieves better false drop probability for the same space overhead than SSF. In the present work, we are mainly focusing on the methods that store the bit matrix column-wise, because they are faster on retrieval. These methods are described in detail next. Table T4.2 gives a list of the names of the methods and their abbreviations.

SSF	Sequential Sign. Files
BSSF	Bit-Sliced Sign. Files
CBS	Compressed Bit Slices
DCBS	Doubly Compressed Bit Slices
NFD	No False Drops

Table T4.2
List of methods and abbreviations

4.2. Bit-Sliced Signature Files (BSSF)

To improve the search time of SSF, we can store the bit-matrix of the signature files column-wise, as shown in Figure F4.1.

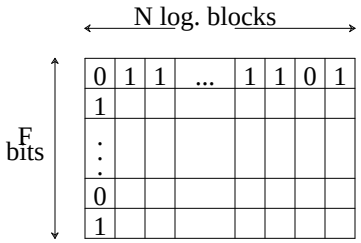


Figure F4.1
Transposed bit matrix

To allow insertions, we propose is to use F different files, one per each bit position, which will be referred to by "bit-files". The method will be called **BSSF**, for "Bit Sliced Signature Files". Figure F4.2 illustrates the proposed file structure.

Searching for a single word requires the retrieval of m bit vectors (instead of all of the F bit vectors) which are subsequently ANDed together. The resulting bit vector has N bits, with "1"'s at the positions of the qualifying logical blocks.

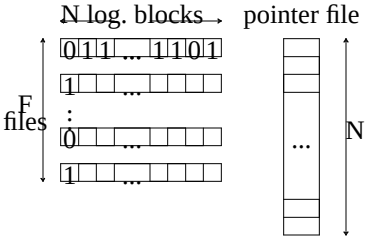


Figure F4.2
File structure for Bit-Sliced Signature Files.
The text file is omitted.

An insertion of a new logical block requires F disk accesses, one for each bit-file, but **no rewriting**! Thus, the proposed methods is applicable on WORM optical disks. As mentioned in the introduction, commercial optical disks do not allow a single bit to be written; thus, we have to use a magnetic disk, that will hold the last page of each file; when they become full, we will dump them on the optical disk. Using the results of the forthcoming analysis, we can predict the size of each bit file and allocate enough space for it on the optical disk from the very beginning. E.g., if the design suggests using F=1000, for an overhead of 10% on a 300Mb disk, then each bit file will require at most 30 Kb. Figure F4.3 shows how the bit files could be actually stored on the surface of the WORM disk.

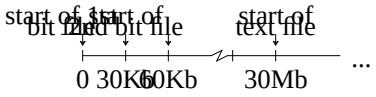


Figure F4.3
Pre-allocation of space for the bit files.

4.3. Compressed Bit Slices (CBS).

Although the bit sliced method is much faster than SSF on retrieval, there may be room for two improvements:

1) On searching, each search word requires the retrieval of m bit files, exactly because each word signature has m bits set to "1". The search time could be improved if m was forced to be "1".

2) The insertion of a logical block requires too many disk accesses (namely, F, which is typically 600-1000)

If we force m=1, then F has to be increased, in order to maintain the same false drop probability (see the formulas in section 5). For the next three methods, we shall use S to denote the size of a signature, to highlight the similarity of these methods to inversion using hash tables. The corresponding bit matrix and bit files will be sparse and they can be compressed. The easiest way to compress each bit file is to store the positions of the "1"'s. However, the size of each bit file is unpredictable now, subject to statistical variations. Therefore, we store them in buckets of size B_p, which is a design parameter. As a bit file grows, more buckets are allocated to it on demand. These buckets are linked together with pointers. Obviously, we also need a directory (hash table) with S pointers, one for each bit slice.

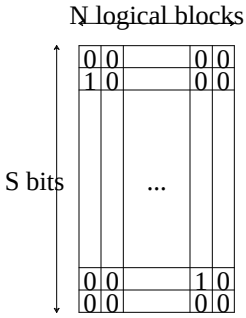


Figure F4.4
Sparse bit matrix

Notice the following:

1) There is no need to split documents into logical blocks any more. This is true for every method that has m=1.

2) The pointer file can be eliminated. Instead of storing the position of each "1" in a (compressed) bit file, we can store a pointer to the document in the text file.

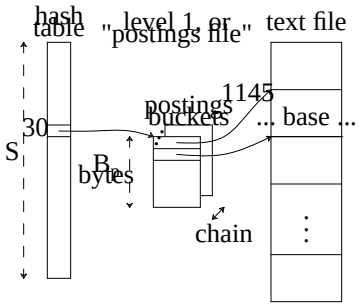


Figure F4.5
Illustration of CBS

Thus, the compressed bit files will contain pointers to the appropriate documents (or logical blocks). The set of all the compressed bit files will be called "level 1" or "postings file", to agree with the terminology of inverted files [25].

The postings file consists of postings buckets, of size B_p bytes (B_p is a design parameter). Each such bucket contains pointers to the documents in the text file, as well as an extra pointer, to point to an overflow postings bucket, if necessary.

Figure F4.5 illustrates the proposed file structure, and gives an example, assuming that the word "base" hashes to the 30-th position (h("base")=30), and that it appears in the document starting at the 1145-th byte of the text file.

Searching is done by hashing a word to obtain the postings bucket address. This bucket, as well as its overflow buckets, will be retrieved, to obtain the pointers to the relevant documents. To reduce the false drops, the hash table should be sparse. The method is similar to hashing. The differences are the following:

(a) The directory (hash table) is sparse; Traditional hashing schemes require loads of 80-90%.

(b) The actual word is stored nowhere. Since the hash table is sparse, there will be few collisions. Thus, we save space and maintain a simple file structure.

The similarities between CBS and hash-based inverted files illustrate the generality of our framework.

4.4. Doubly Compressed Bit Slices (DCBS).

The motivation behind this method is to try to compress the sparse directory of CBS. The file structure we propose consists of a hash table, an intermediate file, a postings file and the text file as in Figure F4.6.

The method is similar to CBS. It uses a hashing function $h_1()$, which returns values in the range $(0, (S-1))$ and determines the slot in the directory. The difference is that DCBS makes an effort to distinguish among synonyms, by using a second hashing function $h_2()$,

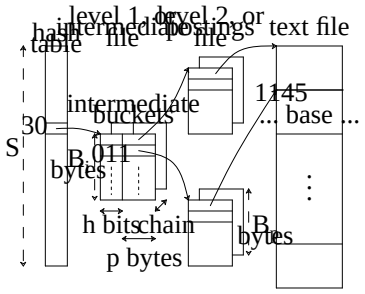


Figure F4.6
Illustration of DCBS.

which returns bit strings that are h bits long. These hash codes are stored in the "intermediate file", which consists of buckets of B_i bytes (design parameter). Each such bucket contains records of the form (hashcode, ptr). The pointer ptr is the head of a linked list of postings buckets.

Figure F4.6 illustrates an example, where the word "base" appears in the document that starts at the 1145-th byte of the text file. The example also assumes that h=3 bits, h_1("base")= 30 and h_2("base")= $(011)_2$.

Searching for the word "base" is handled as follows:

Step 1 h_1("base")= 30: The 30-th pointer of the directory will be followed. The corresponding chain of intermediate buckets will be examined.

Step 2 h_2("base")= $(011)_2$: the records in the above intermediate buckets will be examined. If a matching hash code is found (at most one will exist!), the corresponding pointer is followed, to retrieve the chain of postings buckets.

Step 3 The pointers of the above postings buckets will be followed, to retrieve the qualifying (actually or falsily) documents.

Insertion is omitted for brevity.

OBSERVATION 1: Notice that the postings buckets will be exactly the same as if we had the setting of CBS, with a hash table of size $S\,2^h$ (see Figure F4.7), and a hash function $h(x) = h_1(x)||h_2(x)$. The symbol "||" stands for concatenation of binary strings. This observation shows how DCBS achieves compression of the sparse hash table of CBS. More important is the fact that we can re-use the analytical formulas of CBS with the appropriate changes.

4.5. No False Drops method (NFD).

Here we propose a method to avoid false drops completely, without storing the actual words in the index. The idea is to modify the intermediate file of the DCBS,

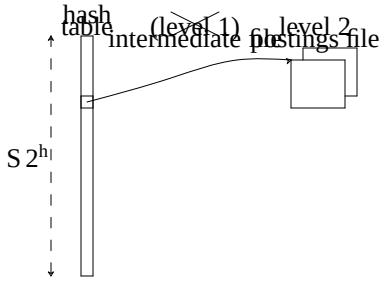


Figure F4.7
Illustration of the equivalence between CBS and DCBS.

and store a pointer to the word in the text file. Specifically, each record of the intermediate file will have the format (hashcode, ptr, ptr_to_word), where ptr_to_word is a pointer to the word in the text file.

This way each word can be completely distinguished from its synonyms, using only h bits for the hash code and p (=4 bytes, usually) for the ptr_to_word. The advantages of storing ptr_to_word instead of storing the actual word are two: (1) space is saved (a word from the dictionary is ≈8 characters long [19]), and (2) the records of the intermediate file have fixed length; thus, there is no need for a word delimiter and there is no danger for a word to cross bucket boundaries.

Searching is done in a similar way with DCBS. The only difference is that, whenever a matching hashcode is found in Step 2, the corresponding ptr_to_word is followed, to avoid synonyms completely.

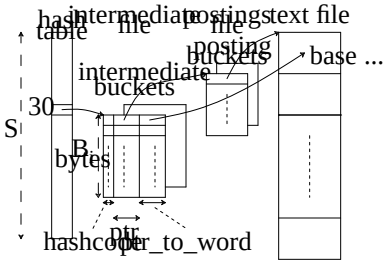


Figure F4.8
Illustration of NFD

5. Analysis.

For every method, we shall examine the performance measures ("output parameters") for a given setting ("input parameters") and a given selection of the "design parameters". The measures we are interested in are listed in T3.1:

- the space overhead O_V
- the false drop probability F_d

• $d_{R,s}$, $d_{R,u}$ number of disk accesses on retrieval (successful and unsuccessful, respectively). They include the disk accesses to search the index, as well as to retrieve the (actually or falsily) qualifying documents.
• d_I number of disk accesses on insertion of a document

Intermediate quantities that we are interested in, are
• $d_{R,i,s}$, $d_{R,i,u}$: the number of disk accesses to search the index for a single word query (for successful and unsuccessful search, respectively). They account for the disk accesses needed to find the list of pointers to the (actually or falsily) qualifying documents (but not to retrieve them).
• $A = \bar{o}$: average number of actual drops, or average number of documents each word occurs in.

For all the methods, the following formulas hold:

$$d_{R,s}=d_{R,i,s}+\left(F_d(N-A)+A\right)\frac{L}{P} \text{ disk accesses} \quad (5.0.1)$$

$$d_{R,u}=d_{R,i,u}+F_dN\frac{L}{P} \text{ disk accesses} \quad (5.0.2)$$

For the last three methods, that use compressed bit sliced signatures, we have:

$$d_I=D\,(d_{R,i,s}+1) \quad (5.0.3)$$

Due to space limitations, we omit the details of the derivation of the formulas (see [7]).

5.1. Sequential Signature Files (SSF).

Overhead.

$$O_V=\left\lceil\frac{F\,N}{P\,b}\right\rceil+\left\lceil\frac{N\,p}{P}\right\rceil \text{ pages} \quad (5.1.1)$$

False drop probability.

$$F_d=(\frac{1}{2})^m, \qquad \text{with } m=\frac{F\ln 2}{D} \quad (5.1.2)$$

Unsuccessful Retrieval.

$$d_{R,i,u}=\left\lceil\frac{F\,N}{P\,b}\right\rceil+F_dN \quad \text{disk accesses} \quad (5.1.3)$$

Successful Retrieval.

$$d_{R,i,s}=\left\lceil\frac{F\,N}{P\,b}\right\rceil+F_d(N-A)+A \quad \text{disk accesses} \quad (5.1.4)$$

Insertion.

$$d_I=1+\left\lceil\frac{F}{Pb}\right\rceil \quad (5.1.5)$$

5.2. Bit-sliced signature files (BSSF).

Overhead.

$$O_V=\left\lceil\frac{FN}{Pb}\right\rceil+\left\lceil\frac{Np}{P}\right\rceil \text{ pages} \quad (5.2.1)$$

False drop probability. The formula for it is exactly the same with the one for SSF (Eq. 5.1.2)

Unsuccessful Retrieval.

$$d_{R,i,u}=\left\lceil\frac{N}{Pb}\right\rceil m+F_dN \text{ disk accesses} \quad (5.2.2)$$

Successful Retrieval

$$d_{R,i,s}=\left\lceil\frac{N}{Pb}\right\rceil m+F_d(N-A)+A \text{ disk accesses} \quad (5.2.3)$$

Insertion.

$$d_I=1+F/2 \quad (5.2.4)$$

5.3. Compressed Bit Slices (CBS).

Each word will occur in $\bar{o}$ documents on the average, where

$$\bar{o}=\frac{ND}{V}=A \quad (5.3.1)$$

Space overhead:

$$O_V=\left\lceil\frac{Sp}{P}\right\rceil+S\,\bar{c}\,\frac{B_p}{P} \quad (5.3.2)$$

where

$$\bar{c}=\sum_{w=0}^{V}c(w)p(V,S,w) \quad (5.3.3)$$

is the average length of chains of pages in the first level, with

$$c(w)=\left\lceil(w\bar{o})\frac{p}{B_p-p}\right\rceil \text{ buckets} \quad (5.3.4)$$

and

$$p(V,S,w)=\binom{V}{w}\left(\frac{1}{S}\right)^w\left(1-\frac{1}{S}\right)^{V-w} \quad (5.3.5)$$

False drop probability.

$$F_d=1-\left(1-\frac{1}{S}\right)^D\simeq\frac{D}{S} \quad (5.3.6)$$

Unsuccessful Retrieval.

$$d_{R,i,u}=1+\bar{c} \quad (5.3.7)$$

Successful Retrieval.

$$d_{R,i,s} = 1 + \bar{c}_s \tag{5.3.8}$$

where

$$\bar{c}_s = \sum_{w'=0}^{V-1} p(V-1,S,w') c_s(w') \tag{5.3.9}$$

and

$$c_s(w') = \left\lceil (w'+1)\bar{o}\frac{p}{B_p - p} \right\rceil \tag{5.3.10}$$

5.4. Doubly Compressed Bit Slices (DCBS).

Space overhead.

$$O_V = \left\lceil \frac{Sp}{P} \right\rceil + O_{V1} + O_{V2} \tag{5.4.1}$$

where

$$O_{V1} = S\,\bar{c}_1\,\frac{B_i}{P} \quad \text{pages} \tag{5.4.2}$$

$$O_{V2} = S\,2^h\,\bar{c}_2\,\frac{B_p}{P} \quad \text{pages} \tag{5.4.3}$$

with

$$\bar{c}_1 = \sum_{w=0}^{V} p(V,S,w)\, c_1(w) \tag{5.4.4}$$

$$c_1(w) = \left\lceil w\,\frac{p+h/b}{B_i - p} \right\rceil \tag{5.4.5}$$

and

$$\bar{c}_2 = \sum_{w=0}^{V} p(V,S\,2^h,w)\, c_2(w) \tag{5.4.6}$$

$$c_2(w) = \left\lceil w\bar{o}\frac{p}{B_p - p} \right\rceil \tag{5.4.7}$$

False drop probability.

$$F_d = 1 - \left(1 - \frac{1}{S\,2^h}\right)^D \approx \frac{D}{S\,2^h} \tag{5.4.8}$$

Unsuccessful Retrieval.

$$d_{R,i,u} = 1 + \bar{c}_1 + \bar{c}_2 \tag{5.4.9}$$

Successful Retrieval.

$$d_{R,i,s} = 1 + \frac{1+\bar{c}_{s,1}}{2} + \bar{\bar{c}}_{s,2} \tag{5.4.10}$$

with

$$\bar{\bar{c}}_{s,1} = \sum_{w'=0}^{V-1} p(V-1,S,w') \left\lceil (w'+1)\frac{h/b+p}{B_i - p} \right\rceil \tag{5.4.11}$$

$$\bar{\bar{c}}_{s,2} = \sum_{w'=0}^{V-1} p(V-1,S\,2^h,w') \left\lceil (w'+1)\frac{\bar{o}p}{B_p - p} \right\rceil \tag{5.4.10}$$

5.5. No False Drops method (NFD).

Space overhead.

$$O_V = \left\lceil \frac{Sp}{P} \right\rceil + O_{V1} + O_{V2} \quad \text{pages} \tag{5.5.1}$$

with

$$O_{V2} = V \left\lceil \bar{o}\frac{p}{B_p - p} \right\rceil \frac{B_p}{P} \quad \text{pages} \tag{5.5.2}$$

and

$$O_{V1} = S \sum_{w=0}^{V} p(V,S,w) \left\lceil w\frac{2p+h/b}{B_i - p} \right\rceil \frac{B_i}{P} \quad \text{pages} \tag{5.5.3}$$

False drop probability.

$$F_d = 0 \tag{5.5.4}$$

Unsuccessful Retrieval.

$$d_{R,i,u} = 1 + \bar{c}_1 + \frac{V}{S\,2^h} \tag{5.5.5}$$

Successful Retrieval.

$$d_{R,i,s} = 1 + \frac{1+\bar{c}_{s,1}}{2} + \frac{1}{2}\frac{V-1}{S\,2^h} + 1 + \left\lceil \bar{o}\frac{p}{B_p - p} \right\rceil \tag{5.5.6}$$

with

$$\bar{\bar{c}}_{s,1} = \sum_{w'=0}^{V-1} p(V-1,S,w') \left\lceil (w'+1)\frac{h/b+p+p}{B_i - p} \right\rceil \tag{5.5.7}$$

6. Experimental results.

To validate our models, we implemented the CBS, DCBS and NFD methods and we run simulation experiments on a set of technical reports. To allow comparisons with previous experiments on signature file methods, we divided the data base into "logical blocks", each of which was treated as a separate document. The words "logical block" and "document" are considered identical. The data base had the following characteristics:

L = 1024 Kbytes per document
D = 58 distinct non-common words per document
N = 2,784 documents
V = 7,765 vocabulary-size of the data base
N L = 2.8 Mbytes size of the data base
N_w = 436,904 total number of words in data base.

The parameters we used for the following experiments are as follows:

postings bucket size, B_p = 68 bytes (16 document

pointers per bucket)
pointer size, p = 4 bytes
byte size, b = 8 bits
hash code, h = 8 bits (for DCBS and NFD only)
Size of a disk page, P = 1024 bytes

The size of the hash table was S=32,000 and 64,000 for the CBS method, and S=1,000 and 2,000 for DCBS and NFD. The size of intermediate blocks for DCBS was B_i= 36 and 24 bytes for S=1000 and 2000 respectively. For NFD, the values of B_i were B_i= 76 and 40 bytes respectively.

We asked 1000 successful and 1000 unsuccessful queries. The query words were chosen randomly from the dictionary of the "spell" utility of UNIX.

Notice that some of the design parameters are not "fine tuned". The main goal in this section is to validate our analytical model, and not to exhibit the best performance that the methods can achieve. Ways of fine tuning the parameters are discussed in [7].

Graphs G6.1-2 compare the theoretical and experimental false drop probability F_d and the search time for the index ($d_{R,i,s}$ and $d_{R,i,u}$), for CBS. The horizontal axis is the size of the hash table S. Graphs G6.3-4 do the same for the DCBS method, and G6.5 for the NFD method. In all the graphs, the dashed lines stand for the theoretically expected values, while the solid ones for the experimental values.

We measured other parameters, too, such as the overhead O_V and the disk accesses on insertion for each of the three methods. To save space, we do not provide graphs for them, but we discuss the results next. The conclusion from the experiments is that the theoretical and experimental values agree very well. Table T6.1 gives a list of the relative errors for the predictions of our analysis.

	CBS	DCBS	NFD
retrieval $d_{R,i,s}$, $d_{R,i,u}$	<5%	<7%	7%
F_d	<10%	<40%	N/A
O_V	0.4%	1%	2%
d_I	≈ 40%		

Table T6.1
Relative errors between theoretical and experimental performance.

The major observations are:

- The model expects ≈ 40% fewer disk accesses d_I on insertion, than actually required. This was the only significant problem of our model. The reason is that words are not distributed uniformly; High frequency words appear in almost any document, and require a long search, exactly because the postings chain for this word is long.
- The model gives pessimistic, but very close estimates for the search performance. The explanation is again the non-uniform distribution of words. Since the query words were chosen randomly from the dictionary of the "spell" utility of UNIX, it was more probable to search for a word with few occurrences, and therefore a shorter postings chain.
- The average number of qualifying documents was 19.05, very close to the theoretically expected number (20.7).

6.1. Arithmetic examples

Since our analysis has been verified, we can use it to predict the performance of our methods on several environments. First, we consider the data base of our experiments. Using only p=3 bytes per pointer and B_p=66, we obtain the graph G6.6. Notice that 3 bytes can hold pointers for data bases of up to 2^{24}= 16 Mb in size. Graph G6.6 plots the total number of disk accesses $d_{R,s}$ on successful search as a function of the overhead O_V for all the signature methods.

Notice that all the proposed methods require very small space overhead (12% for BSSF, ≈ 20% for the rest), with very good search performance: To retrieve ≈ 20 qualifying documents, BSSF requires $d_{R,s}$=55 disk accesses in total (2.75 per document, for 12% overhead), CBS requires 27 disk accesses (1.35 per document, for 22% overhead), DCBS requires 24 disk accesses (1.20 per document, for 22% overhead) and NFD requires 25 disk accesses (1.25 per document for 22% overhead).

A larger data base could have the following characteristics: Consider a 5 1/4 inch optical disk, with 300 Mb capacity, where we plan to store technical papers. Each paper has L=30 Kb size, with a vocabulary of D = 1000 words (these estimates are based on measurements on actual technical reports). The vocabulary of the collection is pessimistically estimated as V=100,000, which is the vocabulary of the best dictionaries [19]. Leaving 50 Mb for the index and other overheads, we have 250 Mb, which can hold 250 Mb / 30 Kb ≈ 8,300 = N documents. Graph G6.7 plots the analytical results for this setting.

Notice that

- the proposed methods require even smaller overhead: BSSF needs 8%, CBS needs 20%, and DCBS and NFD need 17%.
- BSSF is slower on searching, requiring ≈ 100 more disk accesses than the other methods.

7. Discussion - Conclusions.

All of the proposed methods (BSSF, CBS, DCBS, NFD) fulfill the design goals. They require small overhead (≈10% for BSSF, 20-30% for the rest), they are fast on retrieval and they require no rewriting: Thus, they work well with optical disks and they handle insertions easily, allowing readers to access the data base, even when a new document is inserted concurrently.

The proposed methods combine the best of both worlds (signature files and inversion), and can compete against both. Thus, they can be applied in any situation where the traditional SSF have been applied, such as rule indexing in Prolog data bases [1], [23], for formatted attributes [20], [24], or combinations of attributes and text [6] etc..

The proposed methods can also compete against traditional inverted files on magnetic disks, or on erasable optical disks: They give fast responses, for smaller space overhead, allowing searches even during an insertion. They only suffer from the lack of alphabetical ordering on the words of the vocabulary. However, using order preserving hashing functions [16], this disadvantage can be eliminated.

Finally, the proposed methods can be used easily on CD-ROMs. There, the whole data base is available during the design time; accurate statistics can be gathered, and the design parameters can be fine-tuned to the specific data base and the specific performance requirements.

The contributions of this work are the following:

- The framework. It encompasses all the old signature-based methods, and it reveals the similarities between signature files and inverted files, leading to the design of methods that combine the best characteristics of both sides. Moreover, using the framework, any new variation of the SSF gives immediately rise to 3 new methods, one for each entry of Table T4.1.
- The detailed design of the proposed methods and the development of accurate analytical models for their performance study.

Future research can deal with the following topics:

- Simulation of the BSSF method.
- Performance analysis on general, Boolean queries
- Derivation of approximate, simpler formulas for the analysis.
- Performance comparison with B-tree indices.

Acknowledgments.

The authors would like to thank Timos Sellis and Leo Mark for providing some of the technical reports for the experiments.

References

1. Berra, P.B., S.M. Chung, and N.I. Hachem, ‘‘Computer Architecture for a Surrogate File to a Very Large Data/Knowledge Base,’’ *IEEE Computer Magazine*, vol. 20, no. 3, pp. 25-32, March 1987.
2. Christodoulakis, S. and C. Faloutsos, ‘‘Design Considerations for a Message File Server,’’ *IEEE Trans. on Software Engineering*, vol. SE-10, no. 2, pp. 201-210, March 1984.
3. Christodoulakis, S., F. Ho, and M. Theodoridou, ‘‘The Multimedia Object Presentation Manager in MINOS: A Symmetric Approach,’’ *Proc. ACM SIGMOD*, May 1986.
4. Ewing, J., S. Mehrabanzad, S. Sheck, D. Ostroff, and B. Shneiderman, ‘‘An Experimental Comparison of a Mouse and Arrow-jump Keys for an Interactive Encyclopedia,’’ *Int. Journal of Man-Machine Studies*, vol. 24, no. 1, pp. 29-45, Jan. 1986.
5. Faloutsos, C., ‘‘Access Methods for Text,’’ *ACM Computing Surveys*, vol. 17, no. 1, pp. 49-74, March 1985.
6. Faloutsos, C., ‘‘Integrated access methods for messages using signature files,’’ *IFIP WG 8.4 Working Conference on Methods and Tools for Office systems*, pp. 135-157, Pisa, Italy, October 1986.
7. Faloutsos, C. and R. Chan, ‘‘Fast Text Access Methods for Optical and Large Magnetic Disks: Designs and Performance Comparison,’’ UMIACS-TR-87-66 CS-TR-1958, Univ. of Maryland, College Park, Dec. 1987.
8. Faloutsos, C. and S. Christodoulakis, ‘‘Signature Files: An Access Method for Documents and its Analytical Performance Evaluation,’’ *ACM Trans. on Office Information Systems*, vol. 2, no. 4, pp. 267-288, Oct. 1984.
9. Faloutsos, C. and S. Christodoulakis, ‘‘Description and Performance Analysis of Signature File Methods,’’ *ACM TOOIS*, vol. 5, no. 3, pp. 237-257, 1987.
10. Fujitani, L., ‘‘Laser Optical Disk: The Coming Revolution in On-Line Storage,’’ *CACM*, vol. 27, no. 6, pp. 546-554, June 1984.
11. Gonnet, G.H., ‘‘Unstructured Data Bases,’’ Tech. Report CS-82-09, Univ. of Waterloo, 1982.

12. Gonnet, G.H. and F.W. Tompa, ‘‘Mind your grammar: a new approach to modelling text,’’ *Proc. of the Thirteenth Int. Conf. on Very Large Data Bases*, pp. 339-346, Brighton, England, Sept. 1-4, 1987.

13. Haskin, R.L., ‘‘Special-Purpose Processors for Text Retrieval,’’ *Database Engineering*, vol. 4, no. 1, pp. 16-29, Sept. 1981.

14. Hollaar, L.A., ‘‘Text Retrieval Computers,’’ *IEEE Computer Magazine*, vol. 12, no. 3, pp. 40-50, March 1979.

15. Hollaar, L.A., K.F. Smith, W.H. Chow, P.A. Emrath, and R.L. Haskin, ‘‘Architecture and Operation of a Large, Full-Text Information-Retrieval System,’’ in *Advanced Database Machine Architecture*, ed. D.K. Hsiao, pp. 256-299, Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

16. Knott, G.D., ‘‘Expandable Open Addressing Hash Table Storage and Retrieval,’’ *Proc. SIGFIDET*, pp. 187-206, San Diego, Calif, 1971.

17. Mooers, C., ‘‘Application of Random Codes to the Gathering of Statistical Information,’’ Bulletin 31, Zator Co, Cambridge, Mass, 1949. based on M.S. thesis, MIT, January 1948

18. Nofel, P.J., ‘‘40 Million Hits on Optical Disk,’’ *Modern Office Technology*, pp. 84-88, March 1986.

19. Peterson, J.L., ‘‘Computer Programs for Detecting and Correcting Spelling Errors,’’ *CACM*, vol. 23, no. 12, pp. 676-687, Dec. 1980.

20. Pfaltz, J.L., W.H. Berman, and E.M. Cagley, ‘‘Partial Match Retrieval Using Indexed Descriptor Files,’’ *CACM*, vol. 23, no. 9, pp. 522-528, Sept. 1980.

21. Price, Joseph, ‘‘The Optical Disk Pilot Project At the Library of Congress,’’ *Videodisc and Optical Disk*, vol. 4, no. 6, pp. 424-432, Nov.-Dec. 1984.

22. Rabitti, F. and J. Zizka, ‘‘Evaluation of Access Methods to Text Documents in Office Systems,’’ *Proc. 3rd Joint ACM-BCS Symposium on Research and Development in Information Retrieval*, Cambridge, England, 1984.

23. Ramamohanarao, K. and J. Shepherd, ‘‘A Superimposed Codeword Indexing Scheme for Very Large Prolog Databases,’’ *Third Intern. Conf. on Logic Programming*, Springer Verlag, London, 1986.

24. Sacks-Davis, R. and K. Ramamohanarao, ‘‘A Two Level Superimposed Coding Scheme for Partial Match Retrieval,’’ *Information Systems*, vol. 8, no. 4, pp. 273-280, 1983.

25. Salton, G. and M.J. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill, 1983.

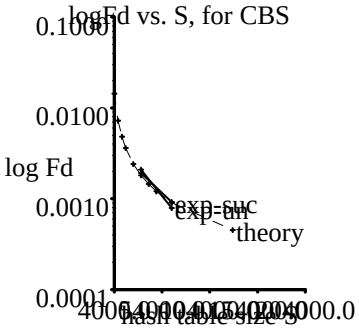
26. Standish, T.A., ‘‘An Essay on Software Reuse,’’ *IEEE Trans. on Software Engineering*, vol. SE-10, no. 5, pp. 494-497, Sept. 1984.

27. Stanfill, C. and B. Kahle, ‘‘Parallel Free-Text Search on the Connection Machine System,’’ *CACM*, vol. 29, no. 12, pp. 1229-1239, Dec. 1986.

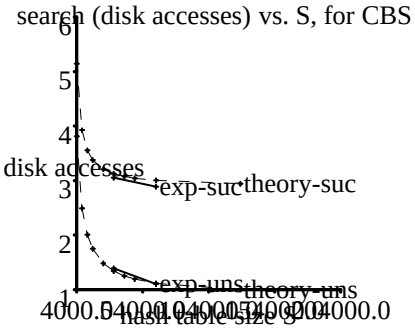
28. Stiassny, S., ‘‘Mathematical Analysis of Various Superimposed Coding Methods,’’ *American Documentation*, vol. 11, no. 2, pp. 155-169, Feb. 1960.

29. Thoma, G.R., S. Suthasinekul, F.A. Walker, J. Cookson, and M. Rashidian, ‘‘A Prototype System for the Electronic Storage and Retrieval of Document Images,’’ *ACM TOOIS*, vol. 3, no. 3, July 1985.

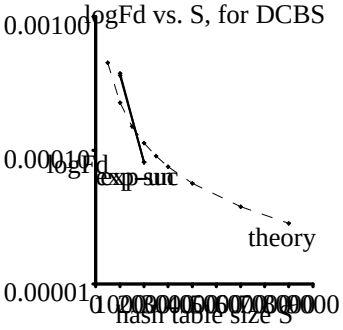
30. Tsichritzis, D. and S. Christodoulakis, ‘‘Message Files,’’ *ACM Trans. on Office Information Systems*, vol. 1, no. 1, pp. 88-98, Jan. 1983.



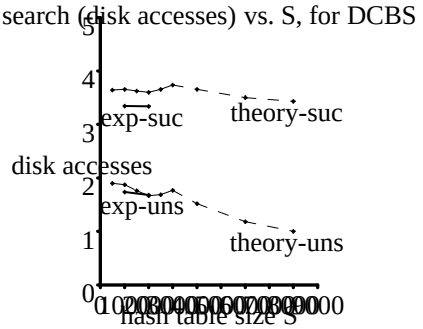
Graph G6.1
F_d for the CBS method;
theory and experiments.



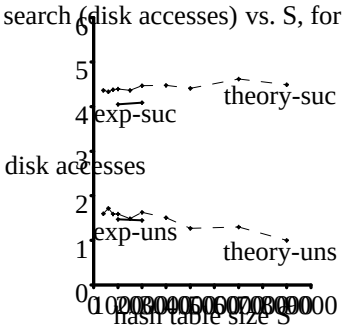
Graph G6.2
Search time for the CBS method;
theory and experiments.



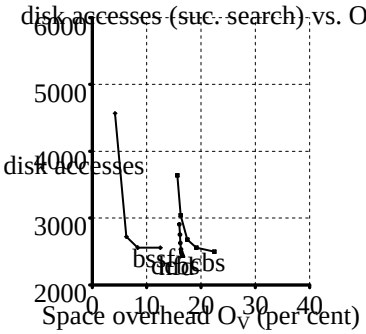
Graph G6.3
F_d for the DCBS method;
theory and experiments.



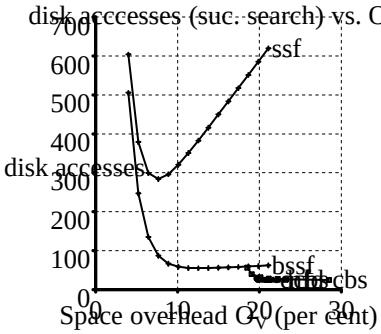
Graphs G6.4
Search time for the DCBS method;
theory and experiments.



Graph G6.5.
Search time for the NFD method;
theory and experiments.



Graph G6.7
Total disk accesses on successful
search versus space overhead
for a 250Mb data base. Analytical results.
Squares correspond to the CBS method,
circles to DCBS and triangles to NFD.



Graph G6.6
Total disk accesses on successful search $d_{R,s}$
versus space overhead. Analytical results
for the 2.8 Mb data base, with p=3 bytes.
Squares correspond to the CBS method,
circles to DCBS and triangles to NFD.