Fast Information Retrieval Using Compressed Multi-Fragmented Signature Files

Seyit KOCBERBER[1] **Fazli CAN**[2*]

[1] Department of Computer Engineering and Information Science, Bilkent University
Bilkent, 06533 Ankara, Turkey

[2] Department of Systems Analysis, Miami University, Oxford, OH 45056, USA
e-mails: seyit@bilkent.edu.tr, canf@muohio.edu

BU-CEIS-9625

November 21, 1996

Abstract: A new file method, called Compressed Multi-Fragmented Signature File (C-MFSF), that uses a partial query evaluation strategy with compressed signature bit slices is presented. The experiments show that for queries with more than one term, C-MFSF obtains the query results with fewer disk accesses than the inverted file approach. For single term queries, which is rare for very large databases, it requires more disk accesses than the inverted file method while its response time is still satisfactory. The number of disk accesses is the same as the number of query terms for queries with more than two terms. The experiments with a real database of 152,850 records validate the simulation results and are used to project the response time for very large databases. For a database of one million records, the projected response times for single term and multi-term queries are less than 2 and 1.14 seconds, respectively.

1. INTRODUCTION

Signature file approach is a well-known indexing technique for information retrieval [FAL92]. In this approach, the content of a record (an instance of any kind of data will be referred to as a *record*) is encoded in a bit string called *record signature*. During the generation of signatures each term (an attribute of a record, without loss of generality, will be referred to as a *term*) is hashed into a bit string of size F by setting S bits to "1" (*on-bit*) where $F > S$. The result is called a *term signature*. Record signatures are obtained either by concatenating or superimposing the signatures of the record terms. In this paper we consider only superimposed signatures and conjunctive queries.

Several signature generation and signature file methods have been proposed to obtain a desirable response time and space overhead. A survey of the proposed methods can be found in [AKT93, FAL92]. In the superimposed signature approach the length of the record signature (F) and term signatures are the same and $F >> S$. Similar to records, query signatures are obtained by superimposing the query term signatures.

* To whom all correspondence should be addressed voice: (513) 529-5950, fax: (513) 529-1524

The Multi-Fragmented Signature File (MFSF) method provides promising results for conjunctive queries [KOC95, KOC96a]. In this paper, we propose the Compressed Multi-Fragmented Signature File (C-MFSF) method that stores the sparse bit slices of MFSF with large F values in a compressed form. A prototype information retrieval (IR) system is implemented based on the proposed method C-MFSF. Its performance is measured analytically and experimentally. A theoretical comparison with the inverted files in terms of disk accesses is also provided.

The organization of the paper is as follows. Section 2 explains query processing in MFSF. The simulation environment and the test databases used in the experiments are described in Section 3. In Section 4, a new method to code the positions of "1"s in the bit slices of a signature file is proposed. The results obtained with simulation runs are given in Section 5. In Section 6, the results of the experiments with real data are provided. Section 7 contains the projection results obtained for very large databases. Section 8 contains a theoretical comparison of C-MFSF and the inverted file method. Finally, Section 9 concludes the paper.

2. QUERY PROCESSING IN MFSF

The query evaluation with signature files is conducted in two phases. In the first phase of the query processing, the query signature is compared with the record signatures. The records whose signatures contain at least one "0" bit (off-bit) in the corresponding positions of on-bits of the query signature are definitely irrelevant to the query. If a record contains all of the query terms, i.e., the record is relevant to the query, the record signature will have on-bits in the corresponding bit positions of all on-bits of the query signature. Thereby in the first phase most of the irrelevant records are eliminated.

Due to hashing and superimposition operations used in obtaining signatures, the signature of some irrelevant records may match the query signature. These records are called *false drops*. The false drop probability is minimized when the *optimality condition* is satisfied, i.e., half of a record signature bits are on-bits [CHR84, ROB79]. In the second phase of the query processing, these possible false drop records are resolved by accessing the actual records.

For a database of N records, the signature file can be viewed as an N by F bit matrix. Off-bits of a query signature have no effect on query processing, since only the on-bits of the query signature are compared with the corresponding record signature bits. Therefore, signature file processing can be done by considering only the columns (bit slices) of the bit matrix corresponding to the on-bits of the query signature.

To retrieve the record signature bits corresponding to a bit position without retrieving other bits, the signature file is vertically partitioned and the bits of a vertical partition are stored

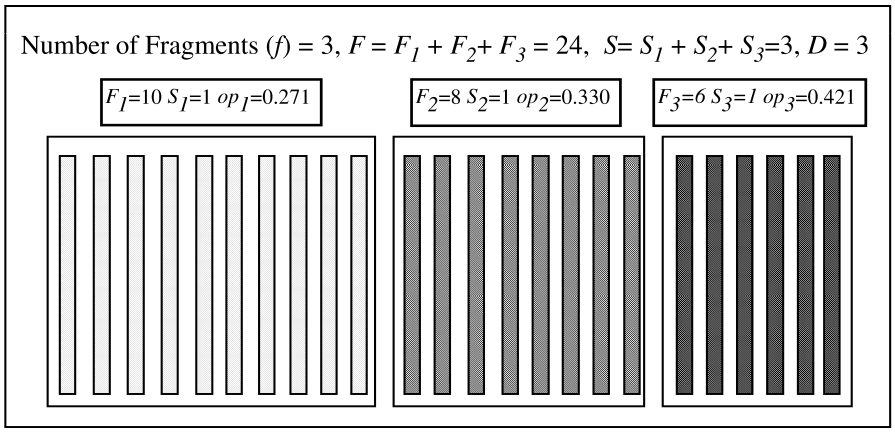
sequentially as in bit-sliced signature files (BSSF) [ROB79] and generalized frame-sliced signature files (GFSSF) [LIN92]. Vertical partitioning a signature file improves performance by reducing the amount of data to be read and processed.

In BSSF, especially for multi-term queries, the time required to complete the first phase of the query evaluation increases as the number of on-bits of the query signature, i.e., query weight, increases [ROB79]. MFSF solves this problem by employing a partial evaluation strategy and considering the submission probabilities of the queries with different number of terms in a multi-term query environment [KOC95, KOC96a]. The partial query evaluation technique employs a stopping condition that tries to complete the first phase of the query evaluation without using all on-bits of the query signature, i.e., by *partial evaluation*. The aim of the stopping condition is to reduce the number of expected false drops to the level that will also provide the lowest response time within the framework of the bit-sliced signature file environment [KOC95, KOC96a].

In MFSF a signature file is conceptually divided into f sub-signature files. The bits of a signature file is distributed among the sub-signature files, fragments, such that $F = F_1 + F_2 \ldots + F_f$ ($f \leq F$). Each term sets S_r bits in the rth fragment such that $S = S_1 + S_2 \ldots + S_f$ ($0 < S_r < F_r$, $1 \leq r \leq f$). Each sub-signature file is a BSSF with its own F (signature size) and S (number of bits set by each term) parameters and consequently, each fragment may have a different on-bit density value [KOC95, KOC96a].

In the bit sliced signature file approach, each processed bit slice eliminates a fraction of the false drops depending on the on-bit density (*op*) of the processed bit slice (*op* is the probability of a particular bit of a bit slice being an on-bit). Lower *op* values eliminate false drops more rapidly during signature file processing and the stopping condition is reached in fewer evaluation steps. In MFSF, since each term sets bit(s) in each fragment, more bit slices from the lower on-bit density fragments are processed in the query evaluation for increasing number of query terms. This property of MFSF is illustrated in Figure 1.

In the example MFSF of Figure 1, there are 24 bits in each record signature and these bits are distributed among three fragments. Since each term sets only one bit to "1" in each fragment and $F_1 > F_2 > F_3$, $op_1 < op_2 < op_3$ holds where op_i ($1 \leq i \leq 3$) denotes the on-bit density in the ith fragment. Since op_1 has the lowest value, processing a bit slice from the first fragment eliminates more false drops than processing a bit slice from the second and the third fragments. Similarly, processing a bit slice from the second fragment eliminates more false drops than processing a bit slice from the third fragment.



Number of On-Bits in Each Query Format:

$t = 1$	1	1	1
$t = 2$	2	2	2
$t = 3$	3	3	2*

t: number of query terms, *D*: number of distinct terms in a record
Different gray levels indicate different on-bit densities of the fragments
(*) More than one term may set the same bit position

Figure 1. The number of on-bits in the fragments of an example MFSF for various number of query terms.

At the bottom of Figure 1, the expected number of on-bits in the query signatures of the queries with one, two, and three terms are given (*t* denotes the number of query terms). For the following three examples, we assume that the stopping condition requires reducing the level of false drop probability to 0.075 in the first phase of query processing.

For $t = 1$, each fragment have one on-bit. Using two on-bits (one from the first and one from the second fragment) reduces the false drop probability to 0.089 ($0.271 \cdot 0.330 = 0.089$) which is insufficient to reach the stopping condition. Therefore, three bit slices (i.e., one on-bit from each fragment) are used in the query evaluation and the resulting false drop probability is 0.038 ($0.271 \cdot 0.330 \cdot 0.421 = 0.038$).

For $t = 2$, each fragment have two on-bits. The partial evaluation strategy of MFSF uses the on-bits of the lower on-bit density fragments first [KOC95, KOC96a]. The false drop probability after using two bit slices from the first fragment is 0.073 ($0.271^2 = 0.073$) which is sufficient to reach the stopping condition. Therefore, the response time for $t = 2$ is less than the response time for $t = 1$ since it requires fewer bit slice evaluations, i.e., the response time decreases for increasing number of query terms.

For $t = 3$, the first and the second fragments have three on-bits while the third fragment has only two on-bits. Three query terms set three bit positions to "1" in the third fragment but two

of them overlap. Processing only two on-bits from the first fragment is sufficient to reach the stopping condition. However, to guarantee the contribution of each query term to the query evaluation the number of processed bit slices must be greater than or equal to the number of query terms. Therefore, three bit slices from the first fragment are used in the query evaluation and the resulting false drop probability is 0.020 ($0.271^3 = 0.020$). Note that although three bit slices are processed for both $t = 1$ and $t = 3$, the false drop probability of $t = 3$ is less than the false drop probability of $t = 1$. Therefore, the response time of $t = 3$ will be less than the response time of $t = 1$.

3. SIMULATION AND TEST ENVIRONMENT

To estimate the performance of C-MFSF a simulation and test environment is designed. The values of the parameters used in the simulation runs were determined experimentally and reflect a real computing environment. This provides validating the results obtained by simulation runs with experiments based on real data.

3.1 Computing Environment

A 33 MHz, 486 DX personal computer with a hard disk of 360 Mbyte running under DOS 5.0 is used to test the performance of the proposed method. We prefer to use the DOS environment since it provides exclusive control of all resources. In DOS, non-interrupting execution of user programs provides the accurate measure of the response time and produces consistent and reproducible results. The physical layout of a signature file on the disk affects the time required to process the signature file and in the DOS environment it can be controlled using Norton Disk Doctor or similar tools. We provide the values of important system parameters in Table I. We expect that a multi-user system can offer computing power and I/O speed equivalent to our experimental environment if not better [KOC96b]. So the results of the experiments can be achieved in multi-user environments without a performance degradation.

Table I. System Parameter Values of the Computing Environment

B_{size}, size of a disk block (bytes)	8192
P_{size}, size of a record pointer (bytes)	4
T_{byteop}, time required to perform bit operations between two bytes (milliseconds, ms)	0.00127
T_{read}, time required to read a disk block (ms)	5.77
T_{scan}, average time required to match an actual record with a query for false drop resolution (ms)	4.5
$T_{farseek}$, average time required to position the read head of disk to the desired block for the record file (includes rotational latency time) (ms)	30
$T_{nearseek}$, average time required to position the read head of disk to the desired block for the signature file (includes rotational latency time) (ms)	23

3.2 Test Database

We used MARC (MAchine Readable Cataloging) records of the Bilkent University Library collection as the test database (BLISS-1). MARC records are widely used to store and distribute the bibliographic information about various types of materials such as books, films, slides, videotapes, etc. Also, MARC records are basic record structure of many library systems such as Melvyl and OCLC. Additionally, other researchers can obtain MARC records easily for test and comparison purposes.

The number of unique terms in the records of the test database expose a normal distribution. For BLISS-1 the standard deviation for the "number of unique terms per record" is 11.12. In the test database MARC records are aligned according to disk block boundaries such that reading of each record during false drop resolution requires only one disk block access ($RB = 1$) unless the MARC record is larger than a disk block. This alignment increases the size of the data file by 4.34%. The record statistics of BLISS-1 are given in Table II.

Table II. Record Statistics of the Test Database BLISS-1

N, number of records	152,850
D_{avg}, average number of terms in a record	25.7
STD, standard deviation of D values	11.12
D_{max}, maximum number of terms in a record	166
V, number of distinct terms in the database	166,216
total number of terms ($N \cdot D_{avg}$)	3,916,856
average record length (bytes)	613
database size with 4.34% alignment overhead (MB)	93.24
RB, average number of disk block accesses to retrieve a record	1

3.3 Query Cases

To measure the performance of C-MFSF by the experiments with real data we considered three different query cases: Low Weight (LW), Uniform Distribution (UD), and High Weight (HW) queries. The values of P_t ($1 \le t \le 5$), where P_t denotes the probability of submitting a t term query, for these query cases are given in Table III. In the simulation runs and in the experiments with real data we limited the maximum number of query terms, t_{max}, to five.

Table III. P_t Values for LW, UD, and HW Query Cases

Query Case	P_1	P_2	P_3	P_4	P_5
Low Weight (LW)	0.30	0.25	0.20	0.15	0.10
Uniform Distribution (UD)	0.20	0.20	0.20	0.20	0.20
High Weight (HW)	0.10	0.15	0.20	0.25	0.30

For each query case, we generated a query set containing 500 zero hit queries by considering the occurrence probabilities of the number of query terms. For example, since the

occurrence probability of a one term query is 0.10 in the HW query case, the HW query set contains 100 ($0.10 \cdot 1000$) one term queries. In our experiments we also consider the execution time of queries with a specific number of terms. To measure the performance of C-MFSF optimized according to a query case for specific number of query terms, we used additional five query sets: T1, T2, ··, T5. The first query set, T1, contains 500 single term queries, the second query set, T2, contains 500 two term queries, and so on. The observed FD and response time values are obtained by taking the average of the FD and response time values obtained by each query in the query sets.

Except the query set T1, query terms are randomly selected from the vocabulary of the test database. Therefore, the result sets of individual query terms contain relevant records. For the query set T1, to obtain empty result sets we randomly generated 500 zero hit query terms.

4. COMPRESSING SPARSE BIT SLICES OF MFSF: C-MFSF

Compressing the record signatures of sequential signature files is inspected in [FAL85]. In this study, to obtain a lower false drop probability, record signatures are produced using large F and small S values. The resulting sparse record signatures are compressed. In [FAL88] Faloutsos and Chan propose the Compressed Bit Slices (CBS) method and extensions of it. A CBS is a BSSF with $S = 1$ and very large F value. The sparse bit slices of CBS are stored in a compressed form. In C-MFSF the value of S can be greater than or equal to 1 and its value is determined with the optimization algorithm given in [KOC95].

Reducing on-bit density while providing sufficient on-bits in query signatures is possible by increasing F (the number of hashing locations). However, increasing F also increases the space overhead if the bit slices are stored without compression. In this study we propose the Compressed Multi-Fragmented Signature File (C-MFSF) method that stores the bit slices of MFSF in a compressed form [KOC96a]. The space overhead of C-MFSF with a larger F value is less than the space overhead of MFSF with a smaller F value.

In the following presentation we assume that record numbers are represented with positive integers and they are stored in ascending order. The positions of the on-bits in bit slices of MFSF can be considered as record numbers and they are also kept in ascending order. Therefore, we use "*record number*" without limiting its use in the posting list of the inverted file method.

Generally, the differences (*gaps* or *run lengths*) between the record numbers are smaller than the record numbers. Since they may be represented with fewer number of bits, instead of the record numbers the gaps are compressed [GOL66]. For example, the ascending sequence

of record numbers "1, 7, 15, 23, 27" is represented as "1, 6, 8, 8, 4."

The performances of the compression methods are affected by the distribution of the gaps. The distributions of gaps in the bit slices of MFSF generated with BLISS-1 for *op* = 0.011 and *op* = 0.042 are plotted in Figure 2. The y axis, "% of Covered Gaps," represents the percent of the gaps that have a gap length less than or equal to the maximum gap value plotted in the x axis. For example, 95.1% and 76.1% of the gaps have a length of 96 or less for *op* = 0.042 and *op* = 0.011, respectively.

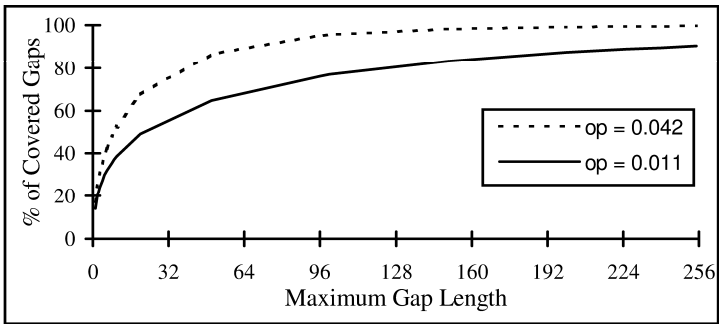


Figure 2. Distribution of the gaps in the bit slices of MFSF for on-bit densities 0.011 and 0.042.

To perform efficient bitwise AND operation between two bit slices we propose a new coding method, *fixed code* (FC), that uses fixed number of bits (*k*) for each codeword. However, in this approach representing a long gap may require more than one codeword. The value of the parameter *k* is determined according to the average gap length as follows.

$$k = \left\lceil \log_2 \frac{N}{N \cdot op} \right\rceil = \left\lceil \log_2 \frac{1}{op} \right\rceil \qquad (1)$$

where *N* is the number of records (number of bits in a uncompressed bit slice) in the database, *op* is the on-bit density of the bit slice, and $N \cdot op$ is the average number of on-bits in the slice.

In FC, a codeword with *k* bits can represent 2^k different codes. Among these codes, 0 (all bits are "0") is used to represent $2^k - 1$ consecutive "0"s either after the last "1" or from the start of the bit string. A code value v $(1 \le v \le 2^k - 1)$ represents $v - 1$ consecutive "0" followed by a "1" either after the last "1" or from the start of the bit string. Note that a FC with *k* = 1 corresponds to the bit string representation for the ascending record numbers.

FC can be explained with gaps as follows. A gap is represented with *k* bits if the gap is less than or equal $2^k - 1$. Otherwise, a codeword of length *k* with all "0" is used and $2^k - 1$ is subtracted from the gap value. The remaining part is coded with FC. Thus, a gap may be represented with more than one codeword. Some sample gap values coded in FC with *k* = 4 and *k* = 8 are given in Table IV (the codewords are divided with spaces).

Table IV. Example FC Codes with k = 4 and k = 8

Gap	k = 4	k = 8
1	0001	00000001
15	1111	00001111
16	0000 0001	00010000
47*	0000 0000 0000 0010	00101111
255*	(16 "0000") 1111	11111111
257*	(17 "0000") 0010	00000000 00000010

* $47 = 3 \cdot 15 + 2,\ 255 = 16 \cdot 15 + 15,\ 257 = 17 \cdot 15 + 2$

In the best case, each gap value (on-bit) is represented with one codeword (k bits) and the signature file contains $F \cdot N \cdot op \cdot k$ bits. In the worst case, the first $N \cdot (1 - op)$ bits of all bit slices are "0" while the remaining $N \cdot op$ bits are on-bits. For each bit slice, leading "0"s are represented with $\left\lfloor \frac{N \cdot (1-op)}{2^k - 1} \right\rfloor$ codewords and remaining on-bits are represented with $N \cdot op$ codewords. Therefore, in the worst case the compressed signature file contains

$$F \cdot k \cdot \left(\left\lfloor \frac{N \cdot (1-op)}{2^k - 1} \right\rfloor + N \cdot op \right) \tag{2}$$

bits. On the average there are $F \cdot N \cdot op$ on-bits in the signature file; hence, for the worst case each on-bit is represented by

$$WC_k = k + \frac{k \cdot \left\lfloor \frac{N \cdot (1-op)}{2^k - 1} \right\rfloor}{N \cdot op} \tag{3}$$

number of bits.

We compare the performance of γ (gamma) [ELI75], δ (delta) [ELI75], Golomb [GOL66], and FC on the bit slices of MFSF produced for BLISS-1. The number of bits required to represent each on-bit for various op values for γ, δ, Golomb, and FC ("Obs "denotes the observed, "Best" denotes the best case, and "Worst" denotes the worst case behavior of FC) are given Table V. (For the Golomb code, an appropriate b value is computed for each bit slice as proposed in [WIT94]. The definition of b is also provided in the same reference.) Similarly, for FC, a different k value is determined using Equation (1) for each slice. FC outperforms γ and δ codes and uses approximately one bit more than the Golomb code for small op values. Note that the observed number of bits required per on-bit by FC is approximately equal to the average of its best and worst cases.

For low op values (such as the ones used in Table V), the number of bits required to represent an on-bit in FC is very close to the number of bits in a byte. Using a fixed size codeword that fits a byte provides efficient processing of compressed bit slices since one byte is used to represent a character and the computers contain operations to manipulate them

efficiently.

Table V. Average Number of Bits Required to Represent an On-Bit for γ, δ, Golomb and FC for Various op Values for BLISS-1

op	γ	δ	Golomb	FC (Obs)	FC (Best)	FC (Worst)
0.011	8.84	8.34	6.96	8.15	5.86	10.06
0.014	8.42	8.02	6.70	7.79	5.63	9.63
0.028	7.33	7.17	6.00	6.85	4.98	8.47
0.042	6.63	6.62	5.54	6.26	4.55	7.72
0.069	5.76	5.94	4.94	5.51	4.00	6.78

If the space overhead is the most important criteria for the performance, the Golomb code may be used to compress the bit slices of MFSF. However, if obtaining a better response time is the primary objective, FC may be preferred since it requires less CPU operations to decode a codeword while providing a satisfactory compression. In the following analysis and experiments with real data we compressed bit slices of MFSF using FC with $k = 8$.

5. PERFORMANCE ESTIMATION WITH SIMULATION RUNS

We use the *response time* as the performance measure [LIN92]. It involves the time required to process the signature file, resolve all false drop records (if any), and find the first relevant record to the query. Generally, IR systems display the first screen of the relevant records to a query. Remaining records are retrieved in groups upon user requests. Therefore, the definition of response time coincides with real applications. According to this definition, the response time after processing i bit slices, $RT(i)$, is estimated as follows.

$$RT(i) = \sum_{s=1}^{i} T_{slice-s} + FD_i \cdot T_{resolve} \quad where\ 0 < i \le W(Q)_t \qquad (4)$$

where $T_{slice\text{-}s}$ is the time required to process the sth bit slice used in the query evaluation, FD_i is the expected number of false drops after processing i bit slices, $T_{resolve}$ is the time required to resolve a false drop, t is the number of query terms, and $W(Q)_t$ is the number of on-bits in the query signature. To provide the contribution of each query term to the query evaluation we forced to use at least one on-bit from each term.

The number of evaluation steps, i, and the expected number of false drops after processing i bit slices, FD_i, are determined as in [KOC96a]. The expected number of false drops are determined by considering the individual number of distinct terms in each record of BLISS-1. Also, C-MFSF is optimized with the heuristic search algorithm given in [KOC96a].

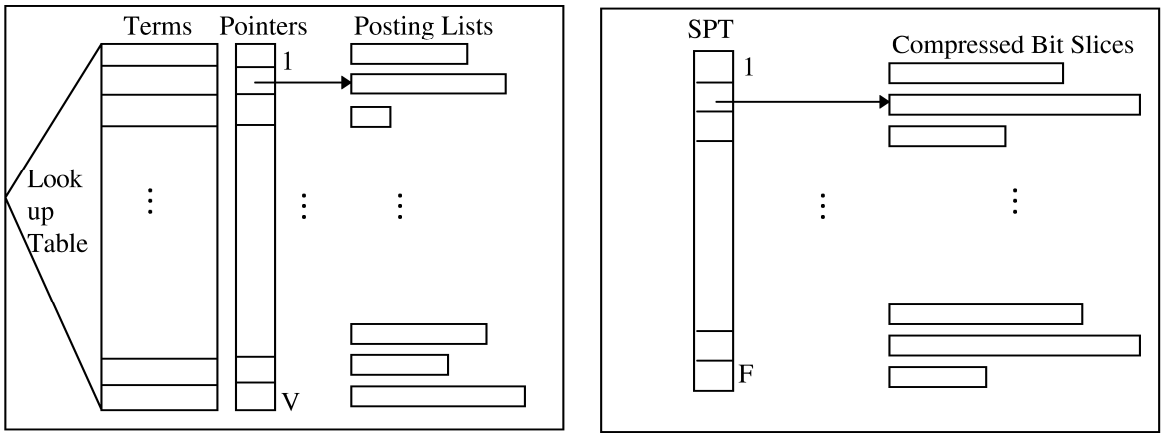
In MFSF each fragment may have a different *op* value and hence the number of on-bits in the bit slices of MFSF and the length of the compressed bit slices vary. To obtain the addresses of the compressed bit slices a Slice Pointer Table (SPT) with F locations is used. SPT is stored

in memory and to retrieve a bit slice, first the address of the bit slice is obtained by accessing SPT. To illustrate the difference between C-MFSF and the inverted file method the storage structures of these methods are shown in Figure 3.

The observed number of bits required to store an on-bit with FC is approximately equal to the average of the best and the worst cases (see Table V), i.e., each on-bit is represented with $(k + WC_k)/2$ bits. Therefore, we estimate the number of disk block accesses to retrieve a bit slice of C-MFSF, *sl*, as follows.

$$sl_i = \left\lceil \left(N \cdot op_i \cdot \left(\frac{k_i + WC_{k_i}}{2} \right) \right) \Big/ \left(8 \cdot B_{size} \right) \right\rceil \quad for\ 1 \le i \le f \qquad (5)$$

bits where op_i is the expected on bit density in ith fragment, k_i is the codeword length used in this fragment, and WC_{k_i} is the number of bits required to store an on-bit of the ith fragment in the worst case (see Equation 3).



a. Inverted File method. b. C-MFSF method.

V: Number of unique terms in the vocabulary, F: Number of hashing positions (signature size), Usually F << V

Figure 3. Storage structures of C-MFSF and the inverted file methods.

The time required to position the read head of disk to the desired block, seek time, depends on the size of the processed file. Since the compressed signature files are relatively small (approximately 15% of the record file) we used different seek times for the signature file ($T_{nearseek}$) and the record file ($T_{farseek}$). We estimate the time required to process a compressed bit slice of ith partition as follows.

$$T_{slice-i} = Read(T_{nearseek}, sl_i) + T_{byteop} \cdot \left\lceil \left(N \cdot op_i \cdot \frac{k_i + WC_{k_i}}{2} \right) \Big/ 8 \right\rceil \qquad (6)$$

where T_{byteop} is the time required to process a byte and sl_i is the average number of disk blocks required to store a slice of the ith partition.

Read(T_{seek}, *b)* incorporates the sequentiality probability, *SP*, to the estimation of the time

required to read b logically consecutive disk blocks. SP is the probability of reading the next logically consecutive disk block without a seek operation.

$$Read(T_{seek}, b) = (1 + (b-1)\cdot(1-SP))\cdot T_{seek} + b\cdot T_{read} \qquad (7)$$

where T_{seek} and T_{read} are average times required to position the disk head to the block to be accessed and to transfer a disk block to memory, respectively. The first disk block of each request always requires a seek operation.

Our model is versatile, i.e., it can be used in all operating system environments and is applicable to both dedicated and multi-user IR systems. This is due to the sequentiality probability (SP) concept incorporated into its development.

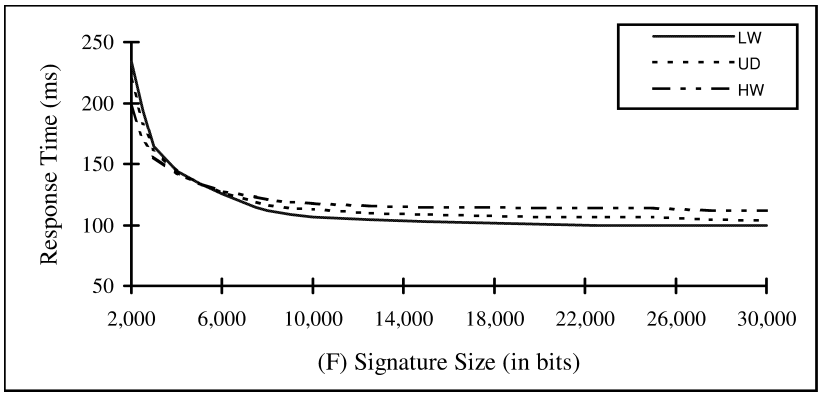
The false drop resolution time for one record, $T_{resolve}$, is computed as follows.

$$T_{resolve} = (1 - \frac{PB}{N})\cdot Read(T_{farseek}, \left\lceil \frac{PB\cdot P_{size}}{B_{size}} \right\rceil) + Read(T_{farseek}, RB) + T_{scan} \qquad (8)$$

where T_{scan} is the time required to compare a record with the query and RB is the average number of disk blocks that must be accessed to read a record. In the above equation obtaining the record pointer can be explained as follows. PB record pointers, each occupying P_{size} bytes, are read into a buffer of $PB \cdot P_{size}$ bytes long at the database initialization stage. Since this is a one time cost, it is excluded from the cost calculations. The probability of finding a requested record pointer in the buffer is approximately equal to PB / N. For the databases with fixed length records or when all record pointers are stored in main memory, PB must be equal to N, i.e., the cost of finding the record pointers is zero.

5.1 Effect of Signature Size on Response Time

We plot the expected response time values of C-MFSF for increasing F values in Figure 4. Increasing F values provides lower on-bit densities and the stopping condition is reached in fewer slice evaluations. Therefore, the optimization algorithm of C-MFSF selects smaller S values for increasing signature size. This also decreases the response time. However, there is a lower bound for the value of S that is one. If a sufficiently large F value is used, S will become equal to one and single term queries can be evaluated with only one seek operation. This idea is inspected by Faloutsos and Chan in [FAL88]. In their model, since storing SPT will require enormous amount of memory they use smaller F values and propose additional data structures to reduce the false drop probability.



(SP = 1.0, N = 152,850)

Figure 4. Expected response time versus very large F values for C-MFSF for LW, UD, HW.

5.2 Effect of Record Size on Response Time

The number of expected false drops depends on the number of bit slices used in the query evaluation and the on-bit densities of these bit slices. Large records increase the on-bit densities of the fragments and require processing more bit slices to reach the stopping condition. Therefore, the value of S increases to provide sufficient on-bits in the query signatures. An increased S value in a resulting configuration implies higher response time. To avoid this problem, i.e., to reach the stopping condition by processing the same number of bit slices, F should be increased to compensate the effect of large records.

To simulate the effect of large records we conceptually added a constant number of terms to all records of our test database. For example, we obtain D_{avg} = 50.7 by conceptually adding 25 terms to all records. For increasing D_{avg} values we search the F value that requires S = 3 which gives the best results in the experiments with the test database (for efficiency, F values are increased in steps of 50). The minimum F values with the expected FD and TR values are given in Table VI.

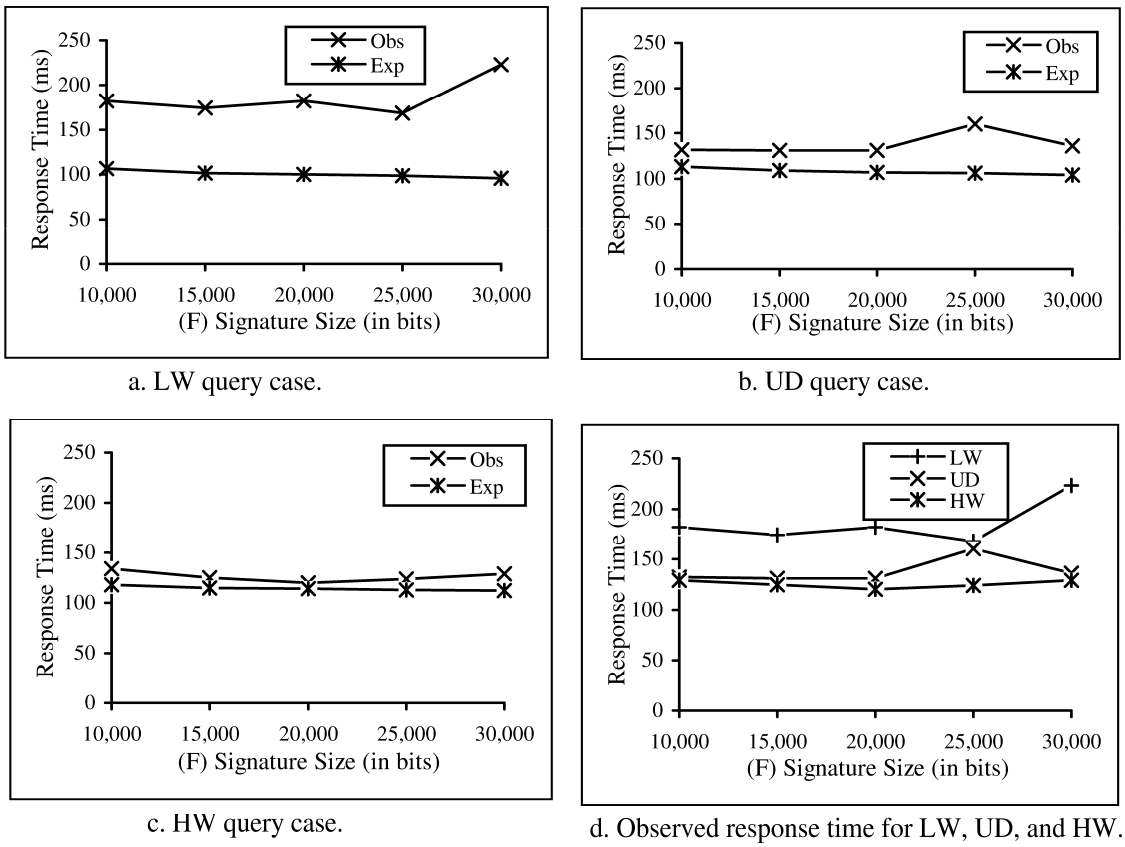
Table VI. Minimum F Values that Provides S = 3 for Increasing D_{avg} Values

Davg	Minimum F	Expected FD	Expected TR
25.7	6150	0.272	125
50.7	10750	0.271	126
100.7	20650	0.269	126
150.7	30700	0.269	126
200.7	40600	0.272	126

The experiments show that similar performance levels can be obtained by selecting an appropriate F value for larger D_{avg} values. Large F values compensate the increased number on bits due to higher number of terms in the records.

6. EXPERIMENTS WITH REAL DATA

The analysis given in the previous section shows that a response time less than 150 milliseconds is possible if large F values are used. We tested the optimized C-MFSF configurations with BLISS-1. The expected (denoted by Exp) and the observed (denoted by Obs) response time values are plotted in Figure 5 (for easy comparison the observed response time values for LW, UD, and HW repeated in Figure 5.d). The expected (denoted by Exp) and the observed (denoted by Obs) average false drop values of these experiments for LW, UD, and HW are given in Table VII.



a. LW query case.

b. UD query case.

c. HW query case.

d. Observed response time for LW, UD, and HW.

Figure 5. Expected and observed response time of C-MFSF versus F for LW, UD and HW (SP = 1).

Most of the inspected C-MFSF configurations require setting three bits for each term. Consequently, the number of on-bits in the signature files are approximately the same for all configurations. Therefore, gap sizes, and hence the size of the compressed signature file, increase for increasing signature size. This causes a small increase in the observed response time for increasing F values.

Table VII. Expected and Observed Average False Drop Values of C-MFSF for LW, UD, and HW

F	LW		UD		HW	
	Exp	**Obs**	**Exp**	**Obs**	**Exp**	**Obs**
10,000	0.064	1.168	0.048	0.280	0.031	0.334
15,000	0.020	1.092	0.014	0.240	0.009	0.196
20,000	0.008	1.126	0.006	0.254	0.004	0.162
25,000	0.088	1.022	0.075	0.650	0.046	0.162
30,000	0.059	1.644	0.050	0.284	0.035	0.258

The observed false drop values and the response time values are greater than the expected values. The difference between the observed and the expected values decreases for increasing query weight. To find the cause of this deviation we evaluate the query sets containing specific number of query terms (T1, T2, T3. T4, and T5) with C-MFSF optimized according to LW, UD, and HW query cases. We measure the average response time and false drop values for each query case. We give the observed response time and false drop values for the LW query case in Table VIII. Similar results are obtained for the UD and HW query cases.

Table VIII. Observed Response Time and False Drop Values for T1, T2, T3, T4, and T5 Evaluated with the C-MFSF Optimized for LW Query Case

	T1		T2		T3		T4		T5	
F	**FD**	**RT**	**FD**	**RT**	**FD**	**RT**	**FD**	**RT**	**FD**	**RT**
10,000	2.340	293	0.428	133	0.010	85	0.000	103	0.000	125
15,000	2.232	281	0.492	133	0.010	85	0.000	104	0.000	125
20,000	2.332	301	0.338	121	0.012	87	0.000	106	0.000	127
25,000	2.480	301	0.306	120	0.004	85	0.000	107	0.000	128
30,000	3.716	414	0.290	124	0.004	90	0.000	112	0.000	136

The queries with more than two terms ($t > 2$) obtain almost no false drops and the query evaluation is completed by accessing only the signature file without any actual record accesses for false drop resolution. Therefore, the number of disk accesses is almost the same as the number of query terms for queries with more than two terms. For $t > 2$, one seek operation is required for each query term. (We deliberately processed at least t bit slices -one bit from each term- even all false drops are eliminated in fewer bit slice evaluation. This guarantees the contribution of each term to the query evaluation process.) Most of the processed bit slices fit into a disk block. The experiments show that the probability of reading a second disk block to process a bit slice is less than or equal to 0.01. Therefore, the results obtained with $SP = 1.0$ can be generalized for other SP values.

The gap sizes and hence the C-MFSF file size increase for higher F values since approximately the same number of on-bits are generated (each term sets three bit positions to

"1" according to the file optimization algorithm [KOC95]) for the inspected F values. Therefore, for $t > 2$ the response time slightly increases for increasing F. For $t \leq 2$, the increase in the number of false drops hides the response time increase due to higher F values.

For $t = 2$, three bit slices are processed and on the average 0.37 false drops is obtained for each query. Therefore, except accessing the RPT (Record Pointer Table), 3.37 seek operations (1.69 seeks per query term) are required to evaluate a two term query.

The difference between the expected and the observed false drops and the response time values are incurred due to single term queries. Single term queries have only three on-bits in their query signature and if one of them shares the same bit slice with a high frequency term, more false drops are produced than the expected number. To obtain better performance with low weight queries, the document frequency of the terms must be considered in the signature file optimization.

7. PROJECTION FOR VERY LARGE DATABASES

For our test database BLISS-1 we performed a series of experiments to test the change in the observed response time for increasing database sizes (N value). The results of the experiments are plotted in Figure 6. (Since the lines are too close to each other we exclude T2 and T4. However, the same trend is observed for these query cases too.) The test cases for the experiments were obtained by considering only the first N records of the original database. The signature file parameters f, F_r, and S_r ($1 \leq r \leq f$) were optimized for each run by considering the tested N value for $SP = 1$ and $F = 15{,}000$.

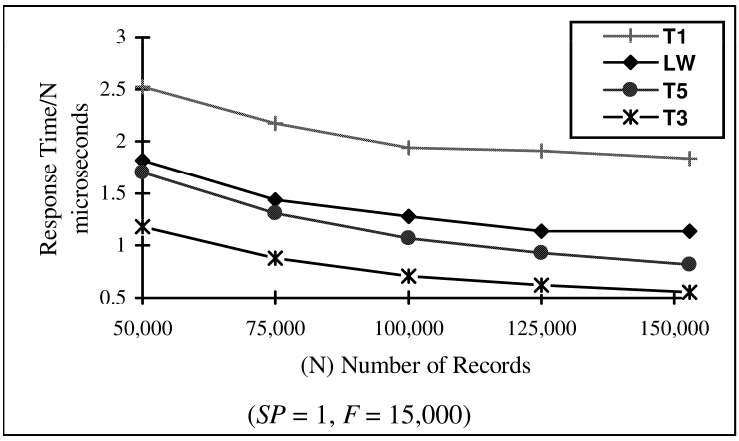


Figure 6. Response time per record versus N for LW, UD and HW.

Our simulation experiments show that approximately the same number of bit slices will be processed for $N = 10^6$ and $N = 152{,}850$ for F values between 10,000 and 30,000.

Consequently, the number of seek operations will be the same for increasing N and the number of seek requests per record will decrease for increasing N when $SP = 1$. (For the bit slices, $SP = 1$ can easily be satisfied since a few number of disk blocks are required per bit slice.) Therefore, in the first phase of a query evaluation the time spend for each record of the database decreases for increasing N value (see Figure 6).

We can project the result of this experiment to predict the observed response time for larger databases by assuming *TR/N* ratio will not be greater than the TR/N figure observed for $N = 152{,}850$. Note that this is a pessimistic assumption since the *TR/N* ratio (response time/record) decreases for increasing N. The projected response time values are given in Table IX.

Table IX. Projected Response Time Values for $N = 10^6$

Query Set	TR for N = 152,850 (ms)	TR/N (microseconds)	TR for $N = 10^6$ (ms)
LW	174	1.138	1138
T1	281	1.838	1838
T2	133	0.870	870
T3	85	0.556	556
T4	104	0.680	680
T5	125	0.818	818

For increasing N values the size of a disk block can be increased such that most of the compressed bit slices still fits into a disk block. In that case, retrieving a bit slice will require only one seek operation for all SP values. Therefore, the response time will be the same for all SP values and the results obtained for C-MFSF with $SP = 1$ can be generalized for other SP values.

8. THEORETICAL COMPARISON OF C-MFSF AND THE INVERTED FILES

In this section, we provide a brief theoretical comparison of IF and C-MFSF in terms of space and the number of disk accesses required to respond a query. In the following discussion, for both methods we assume that RPT (record pointer table) is stored in main memory.

In the IF method at least one disk access is required per query term to read the posting list of the term. (We ignore chained long posting lists, i.e., we assume that $SP = 1$ which is easy to satisfy using the bucket concept or compression for posting lists or both.) Also, to obtain the locations of the posting lists, a lookup table must be maintained and it should be searched for query processing. If we assume only one disk access will be required to obtain the location of the posting list of a query term, each query term will require two disk accesses [ZOB95]. Therefore, in IF, a t term query will require $2 \cdot t$ disk accesses.

In C-MFSF no lookup table is needed. For $F = 30{,}000$, reaching the stopping condition

requires processing only three bit slices even for very large databases ($N \geq 10^6$). For the single term queries C-MFSF requires three disk accesses plus false drop resolution. Therefore, even without any false drops IF outperforms C-MFSF for single term queries. Both methods have similar performance for queries with two terms. IF will require one more disk access but C-MFSF may produce false drops for $t = 2$. However, the average number of false drops require less than one disk access (see Table VIII). Therefore, the expected performance of C-MFSF is better than IF for $t = 2$.

For $t > 2$, since the contribution of each query term to the query evaluation is a must, C-MFSF process t bit slices for a t term query. Experiments with BLISS-1 show that almost no false drop is obtained for queries with more than two terms (see Table VIII). Therefore, we can assume that for $F = 30{,}000$, C-MFSF will require only t disk accesses for queries with t > 2, i.e., one disk access for each query term contrary to two disk accesses of IF.

Since each term sets more than one bit in C-MFSF, the number of on-bits in a C-MFSF will be greater than the number of on-bits in the posting lists of an IF constructed for the same database instance. The number of bits required to store each on-bit of a bit string in a compressed form decreases as the number of on-bits in the bit string increases (see Table V). Since, on the average, a posting list of an IF is more sparse than a bit slice of a C-MFSF, an on-bit of C-MFSF requires less space than an on-bit of IF. Additionally, IF requires storing a lookup table containing an entry for each term of the vocabulary. Therefore, the space overhead of IF also depends on the number of terms in the vocabulary. Usually, records contain unique terms such as names, id numbers, or dates. Consequently, the number of terms in the vocabulary increases as the number of records in the database increases and in turn this increases the space overhead of IF.

For multi-term queries IF may process terms according to their document frequency (from least frequent to most frequent) and may switch to false drop resolution after processing a certain number of terms [ZOB92]. However, this approach implies at least t number of disk accesses just to obtain the document frequency information.

The performance of IF can be increased if the lookup table can be stored in main memory [ZOB92]. In this case, still one disk access for each query term is required to read the posting list of the query term. However, this can be avoided by switching to false drop resolution as sketched above. If such a large memory is available, we can store the compressed form of a C-MFSF fragment (or a part of it) in main memory. For example, a fragment of MFSF for BLISS-1 with op = 0.011 (S and F values of the fragment are 1 and 2400, respectively) will require 3.31 MBytes (average-no.-of-bits/on-bit. $D_{avg} \cdot N = 7.07 \cdot 25.7 \cdot 152850\ bits$) of

memory (see Tables II and V). In C-MFSF the value of *op* (on-bit density) can be adjusted to fit the fragment to the available memory [KOC96a]. Since the bit slices with many on-bits are rarely used in query evaluation (MFSF [KOC95] and consequently C-MFSF use the bit slices with fewer number of on-bits first), to reduce the memory requirement we can store only short bit slices in memory.

Since we store one fragment in memory, for single term queries one of the bit slices will be in memory and only two seek requests will be needed to complete the first phase of the query processing. Similarly, for the queries with two terms since two bit slices will be in memory only one seek request will be needed to complete the first phase of the query processing. For the queries containing more than two terms, one bit slice for each query term will be available without any disk accesses and therefore no disk accesses will be required.

9. CONCLUSION

The Compressed Multi-Fragmented Signature File (C-MFSF) method that uses a partial query evaluation strategy with compressed bit slices is presented. Experiments with real data show that for queries with more than one term, C-MFSF obtains the query results with fewer disk accesses than the inverted file approach. For single term queries it requires more disk accesses than the inverted file method. However, for single term queries, the response time of C-MFSF is still satisfactory: for our database BLISS-1, which contains 152,850 records, it is 318 milliseconds, and for a database of one million records, the expected response time is less than 2 seconds.

The performance of C-MFSF depends on the on-bit density of the signature file and it decreases the on-bit density by increasing F with a limited space overhead. For the databases with large records, we analytically show that the same performance can be obtained by increasing the signature size. Since larger records occupy more disk space, the space overhead of C-MFSF will be approximately the same.

In real IR applications with heavy query load, there are many queries evaluated simultaneously. Retrieving the bit slices that can be used for more than one active query improves the overall performance of the system. This is an interesting problem that we want to study in our future research. We will also investigate buffering (or pre-fetching) and sharing the memory resident compressed bit slices for concurrent queries in multi-user environment. Two additional research topics are the adaptation of C-MFSF to parallel environments and considering the document frequencies of terms in the optimization of C-MFSF.

REFERENCES

[AKT93] Aktug, D., Can, F. 1993. Signature files: an integrated access method for formatted and unformatted databases. Submitted to *ACM Comp. Surveys* (under revision).

[CHR84] Christodoulakis, S., Faloutsos, C. 1984. Signature files: an access method for documents and its analytical performance evaluation. *ACM Transactions on Information Systems.* 3, 4 (Oct.). 267-288.

[ELI75] Elias, P. 1975. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*. 21, 2 (March). 194-203.

[FAL85] Faloutsos, C. 1985. Signature files: design and performance comparison of some signature extraction methods. In *Proceedings of the ACM SIGMOD Conference* (Austin, Tex., May). N.Y. 63-82.

[FAL88] Faloutsos, C., Chan, R. 1988. Fast text access methods for optical and large magnetic disks: design and performance comparisons. In *Proceedings of the 14th VLDB conference* (Long Beach, Calif., Aug.). 280-293.

[FAL92] Faloutsos, C. 1992. Signature files. In *Information Retrieval Data Structures and Algorithms*. Edited by W. B. Frakes and R. Baeza-Yates. Prentice Hall, Englewood Cliffs, N.J. 44-65.

[GOL66] Golomb, S. W. 1966. Run-length encodings. *IEEE Transactions on Information Theory*. 12, 3 (July). 399-401.

[KOC95] Kocberber, S., Can, F. 1995. Vertical fragmentation of superimposed signature files using partial evaluation of queries. Submitted to *Information Processing and Management.*

[KOC96a] Kocberber, S., 1996. Partial query evaluation for vertically partitioned signature files in very large unformatted databases. Ph.D. dissertation, Dept. of Computer Eng. and Information Science, Bilkent University, Ankara, Turkey (http://www.cs.bilkent.edu.tr/theses.html).

[KOC96b] Kocberber, S., Can, F. 1996. Partial evaluation of queries for bit-sliced signature files. *Information Processing Letters* (to appear).

[LIN92] Lin, Z., Faloutsos, C. 1992. Frame-sliced signature files. *IEEE Transactions on Knowledge and Data Engineering*. 4, (3). 281-289.

[ROB79] Roberts, C. S. 1979. Partial-match retrieval via the method of superimposed codes. In Proceedings of the IEEE. 67, 12 (Dec.). 1624-1642.

[WIT94] Witten, I. H. Moffat, A., and Bell, T. C. 1994. *Managing Gigabytes: Compression and Indexing Documents and Images.* Van Nostrand Reinhold, N.Y.

[ZOB92] Zobel, J., Moffat, A., and Sacks-Davis, R. 1992. An efficient indexing technique for full-text database systems. In *Proceedings of 18th VLDB Conference*. (Vancouver, British Columbia, Canada). 352-362.

[ZOB95] Zobel, J., Moffat, A., and Ramamohanarao, K. 1995. Inverted files versus signature files for text indexing. Tech. Rept. CITRI/TR-95-5, Dept. of Computer Science, The University of Melbourne.