Generalized Vertical Partitioning of 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, fc74sanf@miamiu.acs.muohio.edu

BU-CEIS-9513

September 18, 1995

Abstract : A new signature file method, called Generalized Multi-Fragmented Signature File (GMFSF), is presented. The new method provides a unified framework for other vertical partitioning signature files. The performance of GMFSF is measured with a prototype information retrieval system using a database of 152,850 MARC records. The experimental results agree with the theoretical analysis. The response time of GMFSF decreases with an increasing number of query terms and is independent of the number of hits to the query. These features make the method competitive with inverted files, and the experiments further show that GMFSF performs better than inverted files for the non-zero hit conjunctive queries with more than three terms.

1. INTRODUCTION

Recent developments in the data storage technology, e.g., optical disks, enable the storage of formatted and unformatted data, such as text, voice and image in the same database. The growing size of the databases necessitates the development of efficient file structures and search techniques for such multimedia environments [1,9].

Signature files provide a space efficient fast search structure by eliminating most of the irrelevant records by comparing the record signatures and the query signature without retrieving the actual records. In this paper, an instance of any kind of data will be referred to as a *record*. An attribute of a record, without loss of generality, will be referred to as a *term*. In signature approach, record terms are encoded in a bit string called a *record signature*. During the generation of signatures each term is hashed into a bit string of size F by setting m bits to 1 (*on-bit*) where $F > m$. The result is called a *term signature*. Record signatures are obtained either by concatenating or superimposing the signatures of the record terms.

In superimposed signature files, the length of the record signature (F) and term signatures are the same and $F >> m$. For a database of N records, the signature file can be

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

viewed as an N by F bit matrix. Similar to records, query signatures are obtained by superimposing the query term signatures. In this study, 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 [1, 3].

In this study, a generalized multi-fragmented signature file (GMFSF) method is proposed for vertical partitioning of signatures. In vertical partitioning, N by F database signature bit matrix is stored column wise. The size of each column depends on the partitioning scheme. In GMFSF, a signature file consists of sub-signature files with different signature widths (column sizes). Depending on its size each sub-signature file is stored in either the bit-sliced (column size = 1) or frame-sliced (column size > 1) format. The other vertical partitioning approaches, namely bit-sliced (BSSF) [7], generalized frame-sliced (GFSSF) [6], and multi-fragment (MFSF) [5] signature file methods are special cases GMFSF.

A prototype information retrieval (IR) system is implemented based on the proposed method GMFSF. The performance of GMFSF is measured analytically and experimentally. Also, a comparison with the inverted files in terms of response time, space overhead, and inversion time is provided. The experiments show that GMFSF obtains better response times than the inverted file for queries with more than three terms and its space overhead is 13% less than that of the inverted files.

The organization of the paper is as follows. Section 2 gives a description of the signature files and existing vertically partitioned signature file methods. Section 3 describes the proposed signature file generation method. Section 4 gives the optimization algorithm that searches the GMFSF configuration with the best response time. Section 5 presents the results of the experiments with real data. Section 6 covers the results of the comparison with the inverted file method. Finally, Section 7 provides the conclusions.

2. VERTICALLY PARTITIONED SIGNATURE FILES

The query evaluation using signature files is conducted in two phases. In the *first phase* the signature file is used, and record signatures are compared with the query signature. The records whose signatures contain on-bits in the corresponding positions of all on-bits of the query signature are selected as the result of the signature file processing phase. Due to hashing and superimposition operations used in obtaining signatures, the result of the first phase may contain *false drops*: The record signature satisfies the query

although the actual record does not. Therefore, in the *second phase* of query evaluation, the actual records are accessed to resolve possible false drops [1, 3].

Off-bits of a query signature have no effect on the result, since only the on-bits of the query signature are compared with the corresponding record signature bits. Vertical partitioning of a signature file provides the elimination of the slices corresponding to the off-bits of the query signature in query evaluation. Retrieving only the slices corresponding to the on-bits of the query signature reduces the amount of the data to be read; hence, this improves performance. Usually a small subset of the vertical slices of the signature file is needed for the query evaluation.

Short descriptions of the vertical partitioning methods BSSF, GFSSF, and MFSF are given below. In this study one signature is produced for each record. In the text *D* indicates the average (expected) number of terms per record. The meanings of the important symbols used in this paper are provided in Table I.

Table I. Definition of Important Symbols (related equation no.)

Symbol	**Meaning**
$cffd_{(i,t)}$	cumulative false drop probability for a t term query if i on-frames are processed (eqs. 2, A.4)
f	number of fragments
k_i	number of frames in ith fragment
m_i	number of bits to be set by each term in a frame of ith fragment
n_i	number of frames selected in each fragment to set bits
rfd_t	false drop probability for a t term query if all on-frames are processed (eq. 1)
s_i	size of each frame in ith fragment (in bits)
$ffd_{(i,t)}$	frame false drop probability for a t term query if only one on-frame of ith fragment is processed (eq. 9)
t	number of query terms
t_{max}	maximum number of terms in a query
B_{size}	size of a disk block (in bytes)
D	average number of terms in a record
F	size of a signature (in bits)
F_i	size of ith fragment of F (in bits)
$FW(Q_i)_t$	number of on-frames in ith fragment for a t term query (eq. A.1)
N	number of records in database
P	maximum number of records in first result screen
P_t	probability of submission of a t term query
P_{size}	size of a record pointer (in bytes)
PB	number of record pointers in record pointer buffer
RB	average number of disk block accesses required to retrieve a record
RT_t	response time for a t term query (eq. 3)
SP	sequentiality probability of logically consecutive disk blocks
T_{bitop}	time required to perform bit operations on a disk block
T_{read}	time required to read a disk block
T_{recchk}	time required to determine whether a record is found relevant by first phase
T_{scan}	time required to scan a record to test it with query
T_{seek}	time required to position read head of disk
TR	expected response time (eq. 10)
$W(Q_i)_t$	number of on-bits in an on-frame of ith fragment for a t term query

2.1 Bit-Sliced Signature Files : BSSF(D, F, m)

In BSSF each term sets m distinct bits in a bit string of length F and the file is stored in bit-sliced format [7]. In the file, the size of a slice is N bits.

2.2 Generalized Frame-Sliced Signature Files: GFSSF(D, F, m, k, n)

In GFSSF a signature is divided into k equal sized s bits wide frames ($s = F/k$). Each term first randomly selects n frames, then randomly sets m bits (not necessarily distinct) in each of the selected frames [6]. In this method, the size of a frame is $s \cdot N$ bits and each frame is stored separately as a sequential signature file. For $k = F$, the frames become bit-slices of BSSF. A bit-sliced signature file BSSF(D,F,m) is a special case of GFSSF with the parameters (D,F,1,F,m) [6].

GFSSF provides improvement over BSSF [6] by minimizing the number of the seek operations. However, it requires the optimization of the signature file for a specific t value that is undesirable.

2.3 Multi-Fragment Signature Files: MFSF(D, f, { $(F_i, m_i) \mid 1 \le i \le f$ })

In MFSF, a signature file is regarded as the combination of f sub-signature files, *fragments*, such that $F = F_1 + F_2 \cdots + F_f$ ($f \le F$). Each term sets m_i bits in the ith fragment such that $m = m_1 + m_2 \cdots + m_f$ ($m_i < F_i$, $1 \le i \le f$) [5]. Each sub-signature file is a BSSF with the relaxation of *optimality condition*. According to the optimality condition half of the bits in the record signature need to be on-bits [2, 7].

For BSSF, since the query weight (number of on-bits in query signature) will increase as the number of the terms used to construct the query increases, the time required to complete the first phase of the query evaluation will also increase. MFSF solves this problem by using a partial query evaluation technique. The technique employs a stopping condition that tries to complete the first phase of query evaluation without using all on-bits of the query signature. The aim of the stopping condition is to reduce the number of expected false drops to an acceptable level that will also provide the lowest response time within the framework of MFSF.

In MFSF, sub-signature files, fragments, have an increasing on-bit density value. The query evaluation starts by using on-bits from the lowest on-bit density fragment and continues with the higher on-bit density fragments of the query signature until the stopping condition is satisfied.

The stopping condition may leave unused on-bits in the query signature. Therefore, the number of on-bits in the query signature may be reduced by decreasing m_i values of the fragments. This produces record signatures with on-bit densities lower than 0.5, i.e., the optimal value. The optimal value minimizes the number of false drops if all on-bits

of the query signatures are used [2, 7]. In the case of MFSF reducing the on-bit density of the record signatures satisfies the stopping condition with fewer numbers of bit-slice processing and shortens the query processing time.

Optimizing the number of the on-bits in a query signature for a specific t value (number of query terms) will produce non-optimum response times for other t values. Therefore, MSFS approach tries to minimize the expected query response time for a given distribution of t values, not for a specific t value. By this way, it minimizes the average query response time. The performance improvement provided by MSFS depends on the distribution of t values and up to an 80% improvement over BSSF is possible by employing similar partial query evaluation techniques for both methods [5].

A bit-sliced signature file with the parameters (D,F,m) is a special case of MFSF with the parameters (D,1,{(F,m)}) without the relaxation of the optimality condition [5].

3. GENERALIZED MULTI-FRAGMENTED SIGNATURE FILES (GMFSF)

The GMFSF approach combines the concepts introduced in GFSSF and MFSF. A GMFSF consists of f non-equal sized fragments as in MFSF such that $F = F_1 + F_2 \cdots + F_f$ ($f \le F$). Each fragment is treated as a separate GFSSF and contains k_i ($1 \le k_i \le F_i$) equal sized frames. Each term sets bits in all fragments. To set bits in ith fragment, first n_i frames are selected then in each selected frame m_i ($1 \le m_i \le F_i/k_i$) bits are set. A signature generation scheme in GMFSF is defined as GMFSF(D,f,{(F_i,m_i,k_i,n_i) | $1 \le i \le f$ }). The size of the frames in the fragments are

$$s_i \cdot N \ \ where \ \ s_i = F_i/k_i, \quad 1 \le i \le f, \ \ 1 \le f \le F.$$

The signature generation methods BSSF, GFSSF, and MFSF are special cases of GMFSF. The representations of the signature generation schemes of these methods in GMFSF are given in Table II.

Table II. Representations of BSSF, GFSSF, and MFSF in GMFSF

Signature Generation Method	Signature Generation Scheme	Equivalent Representation in GMFSF
BSSF	(D,F,m)	(D, 1, {(F, 1, F, m)})
GFSSF	(D, F, m, k, n)	(D, 1, {(F, m, k, n)})
MFSF	(D, f, { (F_i, m_i) | $1 \le i \le f$ })	(D, f, {(F_i, 1, F_i, m_i) | $1 \le i \le f$ })

3.1 False Drop Probability

Three different false drop probability types are used to describe the false drop calculation method for GMFSF. The first one is rfd_t (*record signature false drop probability*) for a t term query. For rfd_t all on-bits of the query signature and corresponding record signature bits are compared. The second one is $ffd_{(i,t)}$ (*frame false drop probability*) which is the false drop probability if only one frame from ith fragment of the query signature is used

in the query evaluation for a t term query. The third one is $cffd_{(i,t)}$ (*cumulative frame false drop probability*) which is the false drop probability after processing i frames for a t term query. It is computed by the product of the $ffd_{(i,t)}$ values of the frames used in the query evaluation.

The bits set in each fragment is determined independent of other fragments. Therefore, the record false drop probability of the whole signature for a t term query is

$$rfd_t = \prod_{i=1}^{f} rfd_t \text{ of fragment } i \qquad (1)$$

Since each fragment of GMFSF is a GFSSF, rfd_t of the ith fragment can be computed by the formulas given in [6] for GFSSF(F_i,m_i,k_i,n_i). However, these formulas are impractical for the partial query evaluation method described later in Section 3.3. For the partial query evaluation we have to know the expected false drop probability after using i frames from a fragment. Therefore, we will compute the false drop probability as follows.

$$\text{cffd}_{(i,t)} = \text{ffd}^{d}_{(h+1,t)} \cdot \prod_{r=1}^{h} \text{ffd}^{FW(Q_r)_t}_{(r,t)} \qquad (2)$$

$$\text{where } \; i = d + \sum_{r=1}^{h} FW(Q_r)_t, \; h < f, \text{ and } \; 0 \le d \le FW(Q_{h+1})_t$$

The derivation of equation 2 is given in the appendix. Equation 2 reduces to the false drop probability computation formulas of MFSF for $k_i = F_i$ $(1 \le i \le f)$.

3.2 Response Time

We define the *response time* as the time needed to find and display the first P (if exists) relevant records, where P is the maximum number of records shown in the first result screen.

The (query) response time, RT, given below contains the signature file processing time (Tphase1_t) and the time required to resolve false drops and access the first P relevant records (Tphase2_t).

$$RT_t = Tphase1_t + Tphase2_t \qquad (3)$$

where

$$Tphase1_t = \sum_{i=1}^{steps} (Read(size_i) + size_i \cdot T_{bitop}) \qquad (4)$$

Tphase1_t contains reading and processing of *steps* frames from the signature file. The number of the evaluation steps depends on the number of the query terms and whether the partial evaluation method described in Section 3.3 will be used. The frame used in the ith evaluation step occupies $size_i$ disk blocks and these disk blocks are assumed to be

logically consecutive. This means that the byte position of the jth disk block of the frame is $pos + (j-1) \cdot B_{size}$ where pos is the position of the first byte of the frame in the signature file and B_{size} is the size of a disk block in bytes. T_{bitop} is the time required to perform the bit operations on a disk block.

The false drop resolution time for one record, CT, is computed as follows.

$$CT = (1 - PB/N) \cdot Read(\left\lceil \frac{PB \cdot P_{size}}{B_{size}} \right\rceil) + Read(RB) + T_{scan} \qquad (5)$$

The above equation can be explained as follows. To check a record, the record pointer is obtained, the record is read, and the record is scanned to test whether it matches the query. 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 $PB = N$, i.e., all record pointers are stored in main memory, and the cost of finding the record pointers is zero. T_{scan} is the time required to compare a record with the query.

For the second phase of query evaluation, it is assumed that P relevant documents will be retrieved after resolving all false drops. Accordingly the time required to complete the second phase of a t term query is computed as follows.

$$\mathrm{Tphase2_t} = (\mathrm{N} \cdot \mathrm{cffd}_{(\mathrm{steps,t})} + \mathrm{ret}) \cdot \mathrm{CT} + \mathrm{N} \cdot \mathrm{T_{recchk}} \qquad (6)$$

$$\text{where} \quad \mathrm{ret} = \begin{cases} \mathrm{q_{rel}} & \text{if} \quad \mathrm{q_{rel}} < \mathrm{P} \\ \mathrm{P} & \text{otherwise} \end{cases}, \quad \text{and} \quad q_{rel} << N$$

T_{recchk} is the time required to inspect whether a record is identified as relevant by the first phase of the query evaluation. The total number of accessed records contains the expected number of false drops after processing *steps* frames from the signature file and *ret* relevant records. q_{rel} is the actual number of the relevant records and is assumed to be a small fraction of N. For large q_{rel} values, almost all of the first P records found in the first phase will also be actually relevant to the query. Therefore, the false drop part of the equation can be ignored.

Read(b) incorporates the sequentiality probability, SP, to the estimation of the time required to read b logically consecutive disk blocks and is computed as follows.

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

For example, if sequentiality probability is 0.2, to read 10 disk blocks, 8.2 disk seek operations will be required. The first disk block of each request will always require a seek operation. The remaining 9 disk blocks will require 7.2 disk seek operations.

3.3 Partial Evaluation of Queries

Obtaining the lowest possible rfd_t value requires retrieval and processing of $FW(Q)_t$ frames (see equation A.3 in Appendix). On the other hand, reducing the false drop probability below a certain value for a given N value is practically unnecessary. For example, consider a database with the parameters $N = 10^6$, $FW(Q_1)_t \geq 21$, and $ffd_{(1,t)}$ (slice false drop probability of the first fragment) = 0.5. After processing the first 21 frames with at least one on-bit (*on-frame*) from the query signature, the expected number of false drops will be approximately zero.

$$10^6 \cdot 0.5^{21} = 0.477 \cong 0$$

We can stop the first phase of the query evaluation at this step even though there are unprocessed on-frames in the query signature.

A stopping condition is derived in [5] based on this observation. The stopping condition states that the first phase of the query evaluation must end after processing *steps* bit-slices if the false drops to be eliminated by processing the next bit-slice can be eliminated in a less time by accessing the actual records. The stopping condition given in [5] is generalized for GMFSF as follows.

$$RFD_{steps+1} \cdot CT < Read(size_{steps+1}) + size_{steps+1} \cdot T_{bitop} \qquad (8)$$

The undefined variables of expression 8 are defined as follows.

$$\mathrm{RFD}_{\mathrm{steps+1}} = \mathrm{N} \cdot \mathrm{cffd}_{(\mathrm{steps,t})} - \mathrm{N} \cdot \mathrm{cffd}_{(\mathrm{steps+1,t})} = \mathrm{N} \cdot \mathrm{cffd}_{(\mathrm{steps,t})} \cdot (1 - \mathrm{ffd}_{(\mathrm{next,t})})$$

$$\mathrm{next} = \begin{cases} \mathrm{h}+1 & \text{if } \mathrm{d} < \mathrm{W}(\mathrm{Q}_{\mathrm{h+1}})_{\mathrm{t}} \\ \mathrm{h}+2 & \text{otherwise (if } \mathrm{h}+2 > \mathrm{f} \text{ query evaluation is completed)} \end{cases}$$

h is determined as

$$steps = d + \sum_{r=1}^{h} FW(Q_r)_t$$

with the constraints $h < f$, $0 \leq d \leq FW(Q_{h+1})_t$,

$$\mathrm{ffd}_{(\mathrm{r,t})} \leq \mathrm{ffd}_{(\mathrm{r+1,t})} \text{ for } 1 \leq \mathrm{r} < \mathrm{f} \qquad (9)$$

The constraints given in expression 9 guarantees that the stopping condition is reached in a minimum number of evaluation steps by forcing the usage of the frames with lower *ffd* values first. $RFD_{steps+1}$ is the number of false drop records that can be eliminated upon the processing of one more frame after processing *steps* frames. *steps* frames are obtained by using all of the on-frames of the query signature starting from the first fragment and including the hth fragment. In addition to this, d frames are used in the query evaluation from the h+1st fragment. If there is an unprocessed on-frame in the h+1st fragment, the next frame of query evaluation will be selected from it, otherwise the frame will be

selected from the h+2nd fragment. If there are only h+1 fragments, the query evaluation stops before reaching the stopping condition.

A proof is given in [5] which guarantees that once the stopping condition is satisfied, it will be valid in subsequent steps for unchanging frame sizes. In GMFSF the frame sizes may be different in different fragments. To guarantee the validity of the stopping condition for all subsequent evaluation steps after it is satisfied, instead of $size_{steps+1}$ in expression 8, the minimum frame size in the signature file must be used.

The on-frames used in query evaluation are selected from the query terms using a round robin approach (the first on-frame comes from the first query term, the second on-frame comes from the second query term, and so on). For the configurations where the number of the processed on-frames is less than t_{max}, which may exist for small N and large t_{max} values, the stopping condition must be modified to process at least t_{max} on-frames. This ensures that each query term contributes to the query evaluation.

4. SEARCHING THE OPTIMUM CONFIGURATION

In most of the real IR systems, multi-term queries are the norm. Therefore, in the optimization of the signature file parameters the submission of queries with varying number of terms must be considered. Given the occurrence frequencies of the queries with different number of terms used to construct the query, the expected response time can be computed as follows [5].

$$TR = \sum_{t=1}^{t_{max}} P_t \cdot RT_t \qquad (10)$$

where RT_t, given in equation 3, is the time required to evaluate a t term query, P_t is the probability of submission of a t term query, and t_{max} is the maximum number of terms that can be used in a query.

Minimizing the cost function, TR, with the stopping condition given in expression 8 requires determination of the values of the parameters f, F_i, k_i, n_i, and m_i ($1 \le i \le f$). The heuristic search algorithm outlined in Figure 1 is used to search the optimum configuration and to determine the TR value for this case.

The algorithm assumes $q_{rel} = 0$. The stopping condition is independent of the actual number of hits to the query. Therefore, the optimized response time for the zero hit queries will also be the optimized response time for non-zero hit queries provided that $q_{rel} << N$. For large q_{rel} values, the query evaluation may be stopped before reaching the stopping condition given in expression 8. During query evaluation, the number of the candidate relevant records and the expected number of false drops at this step may be used to predict the q_{rel} value. The query evaluation algorithm may test the stopping

condition dynamically by watching the reduction in the expected false drops. However, the experimental results show that counting the relevant records in the result may take as much time as evaluating a frame.

In the algorithm joining of two fragments to form one fragment is initiated when *decrease* m_i or *decrease* n_i is selected and the corresponding parameter value in the selected fragment is one. *Join Fragments*, *increase* F_i, and *decrease* F_i operations require random selection of another fragment. After adjusting the F_i value of a fragment, k_i value is also adjusted such that mod(F_i, k_i)= 0. To prevent trapping in a local minima, a sufficient number of initial configurations must be tried. The results given in this paper are obtained with 80 initial configurations.

Algorithm SearchConfiguration
f ← Select randomly the number of fragments (1 ≤ f ≤ F).
Set F_i values randomly 1 ≤ i ≤ f) where $F = F_1 + F_2 + \cdots + F_f$.
Set k_i values randomly such that mod(F_i, k_i)= 0, (1 ≤ i ≤ f).
Set m_i values to 1 (1 ≤ i ≤ f).
Set n_i values to 1 (1 ≤ i ≤ f).
Mark all fragments as *not-tried*.
minimum cost ← *infinity*.
while there are *not-tried* fragments
    i ← Select randomly a *not-tried* fragment (1 ≤ i ≤ f).
    Select randomly an applicable operation among the operations *split*,
      increase m_i, *decrease* m_i, *increase* F_i, *decrease* F_i, *increase* n_i, *decrease* n_i.
    if an applicable operation exists
        Apply the operation and obtain candidate configuration.
        if total cost, *TR*, of the candidate configuration is less than *minimum cost*
            accept it as the new configuration, *minimum cost* ← *TR*.
            Mark all fragments as *not-tried*.
        else
            Mark fragment i as *tried*.
    else
        Mark fragment i as *tried*.

Figure 1. Algorithm to search optimal fragmentation scheme.

The expected response time depends on the distribution of the number of query terms. The examination of the performance improvement of MFSF over BSSF [5] reveals that the results for uniform distribution of number of query terms are also a good example for other term distributions. Therefore, we will assume uniform distribution for the number of query terms for analytical and experimental tests.

The expected response times of zero-hit queries for the *SP* values 1, 0.5, and 0.0 are plotted in Figure 2. *SP* = 0.0 is the typical situation in multi-user environments. Even if the files are fully sequential (if possible), disk read and write requests originating from

other users will move the read head and the next request probably will require a disk seek operation. However, memory capacity is very high in these environments; therefore, buffering of the data in main memory will reduce the response time. The values of the environment variables used by the optimization algorithm are given in Table III.

Figure 2 shows that for $SP = 1.0$, the response time is a linear function of N. For $SP < 1.0$, the sharp rises in the response time occur with a period of $8 \cdot B_{size}$. At these points the last disk blocks of each frame become full and increasing N by one requires an additional disk block for each frame. The increase in response time is incurred by the increase in the evaluation steps to reach the stopping condition. This is due to the increase in N and the amount of data to be read for each frame.

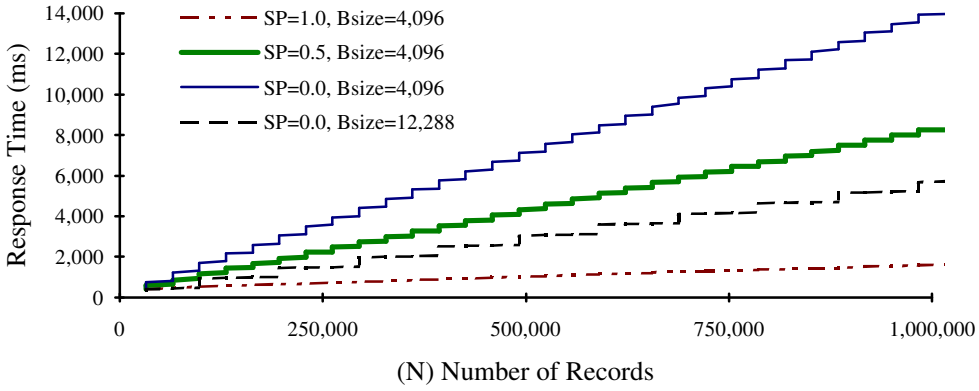

Figure 2. Expected response time for GMFSF with respect to varying N values (F = 960).

The slope of the response time line increases for decreasing SP value. To see the effect of disk block size on the response time, response time values for SP = 0.0 and B_{size} = 12,288 are plotted in Figure 2. As a result, we can say that to alleviate the performance decrease for SP values less than one, the disk block size must be increased. If the disk block size is large enough to hold a frame, the performance is independent of the SP value.

The response time for $N = 10^6$ for various F values are plotted in Figure 3. The signature sizes of 512, 1024, and 2048 corresponds to 10.5%, 21%, and 42% space overhead, respectively. The decrease in the response time becomes insignificant (especially for $SP \geq 0.5$) for the space overhead greater than 20%. Therefore, the simulation runs and the experiments with real data use F = 960 that corresponds to a 20% space overhead.

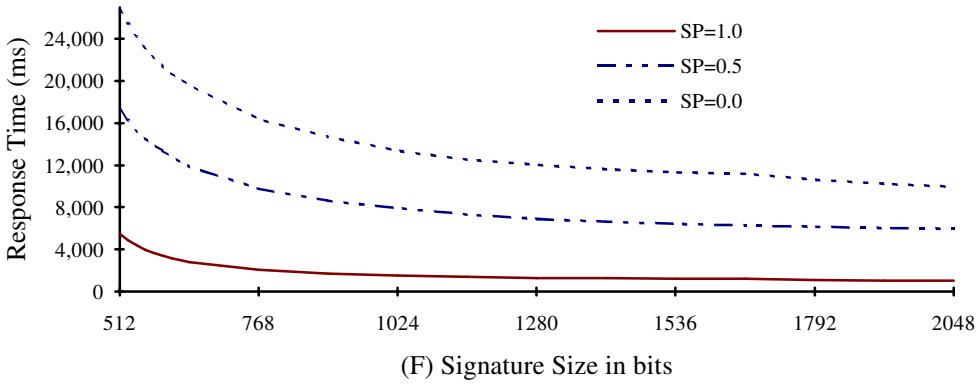

Figure 3. Expected response time for GMFSF versus F ($N = 10^6$).

5. EXPERIMENTAL EVALUATION OF GMFSF

To test the estimated performance of GMFSF with real data, a 33 MHz, 486 DX personal computer with a hard disk of 201 MB running DOS operating system is used. Since DOS provides a single-user non-preemptive programming environment, the processing time can be measured by using the real clock of the system. Also, the file allocation process can be controlled to obtain different *SP* values. We expect that a multi-user system can offer a computing power and I/O speed equivalent to our experimental environment if no better. So the results of the experiments can be achieved in multi-user environments without a performance degradation.

The values of the parameters T_{bitop}, T_{read}, T_{scan}, T_{recchk}, and T_{seek} are determined experimentally. For the disk drive used in the experiments, the value of T_{seek} increases as the size of the file on which the seek operations are performed increases. An average T_{seek} value is obtained by taking the average of T_{seek} values for different file sizes. Parameter values used in the simulations and experiments are given in Table III.

Table III. Parameter Values for the Simulation Runs and Experiments

t_{max}	= 10	maximum number of terms in a query
B_{size}	= 4096	size of a disk block (in bytes)
D	= 27	average number of terms in a record
F	= 960	size of a signature (in bits)
P	= 1	maximum number of records in first result screen
P_{size}	= 4	size of a record pointer (in bytes)
PB	= 1024	number of record pointers in record pointer buffer
RB	= 1	average number of disk block accesses to retrieve a record
T_{bitop}	= 0.616	time required to perform bit operations on a disk block (ms)
T_{read}	= 3.42	time required to read a disk block (ms)
T_{recchk}	= 0.00012	time required to test a result bit of first phase (ms)
T_{scan}	= 4.5	average time required to match a record with query (ms)
T_{seek}	= 80	time required to position read head of disk (ms)

The test data used in the experiments are imported from BLISS (Bilkent University Library Information Services System) which uses MARC records to store bibliographic information of the collection of the university library. MARC records contain variable sized information about the bibliographic materials and are widely used by library automation systems. The test data contains 152,850 records and the size of the data file is 91 MB. Average record length is 613 bytes and on the average each record contains 25.63 terms. MARC records are aligned according to disk block boundaries such that reading of each record 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 10%.

Response time is measured by taking the average for 500 randomly generated zero-hit queries. The number of terms in the queries is determined according to the uniform probability distribution, since it provides typical experimental results [5]. The maximum number of the query terms is 10. Therefore, the test query set contains 50 occurrences of each value of the number of query terms.

Three experiments are performed to compare the expected performance of GMFSF with the performance obtained using real data. In the experiments only the conventional DOS memory is used and buffering is disabled. The experiments and the results obtained are presented below. In this section expected (simulation) response times are obtained by equation 10.

5.1 Experiment I: Finding the Optimum D Value

The MARC records used as the test data are in different sizes and the number of the terms occurring in the records varies. The average number of the terms is 25.63, where 57% of the records have 26 or less terms, i.e., D = 26 covers 57% of the records. This means 43% of the record signatures may contain more on-bits than the optimized number of on-bits assumed by the simulation. The extra on-bits in actual record signatures may cause more false drops than the estimated false drops and may decrease the performance.

To see the effect of using a *D* value different from the average number of terms per record, the response time is measured for four different *D* values. The results are given in Table IV. The expected and observed false drop values are for single term queries. An increasing *D* value decreases the difference between the observed response time and the estimated (simulation) response time. The query evaluation algorithm expects higher on-bit densities for increased *D* value since *F* is fixed. Consequently, more frames are used in the query evaluation and the response time increases. We take D = 27, covering 61%

of the records, since D = 27 obtains the best observed response time value in the test environment.

Table IV. Response Time and False Drop Values for Various D Values

D	Percent of Covered Records	Response Time (ms)		False Drops	
		Expected	Observed	Expected	Observed
25	53%	584	885	2.44	4.27
26	57%	597	754	1.47	3.19
27	61%	604	728	2.13	2.52
28	65%	616	817	1.47	3.44

5.2 Experiment II: Testing the Expected Response Time

The expected and the observed response time values for five different N values are plotted in Figure 4. Except for the smallest and largest database sizes, the expected and observed response time values are very close. The deviation is due to the fact that T_{seek} is proportional to N. For example, T_{seek} value for N = 32,768 is less than the average T_{seek} value. Therefore, the observed response time is less than the expected response time. On the other hand, T_{seek} value for N = 152,850 is larger than the average T_{seek} value and the observed response time is greater than the expected response time. Experimental results validate equation 10. They also show that a response time of 1.6 seconds can be achieved for 10^6 records using a disk drive with 80 ms seek time (refer to Figure 2).

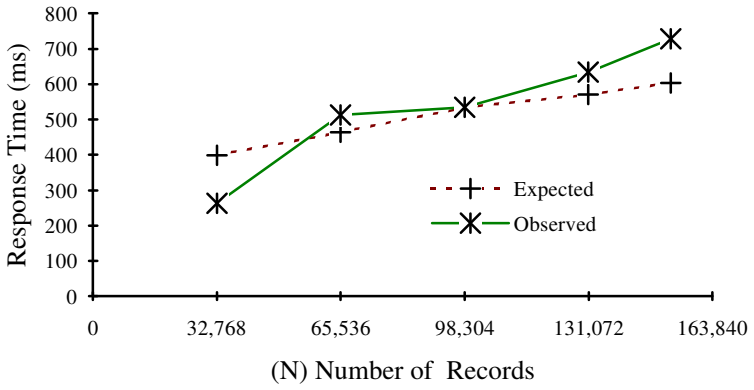

Figure 4. Expected and observed response time.

Our further experiments show that the optimized file parameters obtained for a given N value remains effective for large database increases. For example, we used the optimized signature file parameters for N= 98,304 for the complete database. This increased the observed response time of the latter case by only 2% with respect to its optimized case. We have similar observation in the other experiments; actually the largest deviation from the optimized case is reported above. Therefore, we can say that

for most environments the need for GMFSF reorganization/reoptimization will be infrequent. This is due to the fact that stopping condition for the query evaluation process is determined dynamically according to the current N value.

5.3 Experiment III: Testing the Effect of Sequentiality Probability

An *SP* value of 1.0 is obtained by fully optimizing the disk with Norton Disk Doctor (NDD). NDD reorganizes the files on the disk such that each file occupies physically consecutive disk locations. The *SP* values lower than 1.0 are obtained by randomly fragmenting the free space on the disk, and copying the file to the disk. The value of *SP* for the fragmented file is determined experimentally. For this purpose, first the reading times of fully sequential and fragmented versions of the same file from the beginning to the end are recorded. The difference between fragmented and sequential reading times is divided by average T_{seek} value of the same file to find the number of the seek operations generated by the fragmentation of the file. The ratio of the number of the seek operations to the number of the blocks required to store the file gives the value of *SP*. The *SP* values less than 0.6 necessitates a larger disk capacity than the capacity of the disk used for the experiments.

Table V. Response Times for Different SP values

Sequentiality Probability	Response Time (ms) Experimental	Response Time (ms) Simulation
1.000	535	534
0.870	679	644
0.735	794	755
0.660	800	820

The tests for the sequentiality probability are performed for N = 93,304 that gives the closest expected and observed response times in the previous experiment. The results of the experiment are given in Table V. The expected and observed response time values are very close to each other, and the response time increases for decreasing *SP* values.

6. COMPARING PERFORMANCE OF GMFSF AND INVERTED FILES

To compare GMFSF and inverted files, an inverted file generation and search program are implemented in the C language with the C library of Code Base that offers indexing and sorting functions.

The same record pointer file parameters are used for both methods. The inverted file system contains two files. The first file is used to store the unique terms, total number of the postings, a part of the posting list, and if it exists a pointer to the secondary posting

record for each term. The portion of the posting list stored with the term is called the *prime posting list*. The size of the prime posting list is selected such that the posting list of 90% of the terms can be obtained without additional seek and read requests. Increasing the size of the prime posting list also increases the space overhead. The second file is used to store the secondary posting records. The size of the secondary posting records is selected such that the posting lists of the 6% of the terms can be read with only two seek operations.

6.1 Experiment I: Comparing Signature File and Inverted Index Creation Times

For increasing database sizes, signature file creation times and inverted index creation times are given in Figure 5. The programs written for signature and inverted file creation are open to improvement. For example, the inversion may be performed more efficiently by using the FAST-INV algorithm that provides maximum utilization of available main memory [4]. The space overhead given in Table VI for inverted file starts with 45% and decreases as database size increases since the terms introduced by earlier records are also used by new records. The unique numbers of the bibliographic material such as ISBN, and ISSN are not included in the index. A recent study shows that the space overhead for inverted files can be reduced to less than 10% [10]. However, the decompression operation needed during query processing will increase the response time. The space overhead for the signature file is 20% for all record numbers, i.e., is always less than that of the inverted file.

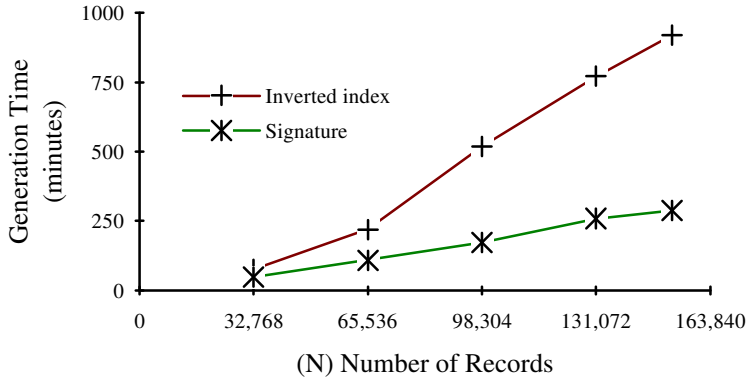

Figure 5. Inverted index and signature file generation times.

Another factor affecting the space overhead for inverted files is the maximum number of the characters in a term which is set to 10 and results truncation of 8% of all term occurrences. For maximum term length of 15, the space overhead for N= 152,850 is 36%.

Table VI. Inverted File Statistics

Number of Records	Number of Unique Terms	Total Number of Terms	Space Overhead	Inversion Time (minutes)
32,768	73,679	737,934	45%	76
65,536	111,170	1,568,021	40%	220
98,304	135,890	2,446,565	36%	518
131,072	154,126	3,334,523	34%	771
152,850	166,216	3,916,856	33%	920

The inversion time increases more rapidly for larger N values. The reason for this is the sorting of the terms extracted from the records. Since our aim is to compare the methods in the same environments, we limit the available memory to the conventional DOS memory. Both methods can perform better with higher memory capacities. For the test environment (N= 152,850) signature file generation requires 33% of the inversion time. A buffer area of 512 bytes is used for each bit of the record signature. After processing 4096 records, the buffers are dumped to the disk. Consequently, increasing the buffer size or decreasing the signature size decreases the signature file creation time.

6.2 Experiment II: Comparing Response Times

The response time of GMFSF decreases with the increase in number of query terms (*t*) since more frames are used from low on-bit density fragments for increased *t* values. For inverted files, it increases with t. The response times of both methods are plotted in Figure 6 for *t* values between one and nine to find the possible cross-over point. Optimizing the signature file parameters for each number of query terms, *t*, value is unrealistic. Therefore, the signature file is optimized for the *t* values 1 to 3, and 4 to 9. The queries with more than 9 terms result in the same response time for GMFSF.

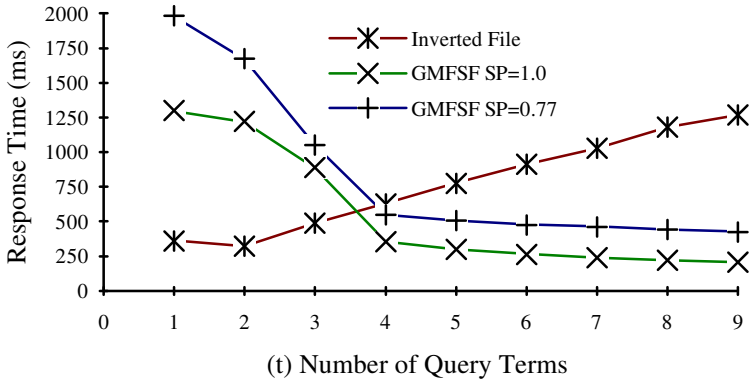

Figure 6. Response times for GMFSF and inverted file (N = 152,850, F = 960).

Another factor that affects the response time for inverted file is the number of the hits for the queries. Instead of controlling the hit numbers in the result, we control the hit numbers for each query term. One-hundred query terms are selected among the unique terms such that the subset obtained reflects the posting list statistics of all unique terms. For example, for the test database 52% of the unique terms have a posting list length of one. Therefore, we select sample query terms such that 52 terms have a posting list length of one. For the experiments the secondary posting records of the same term are stored sequentially which is the situation after creating the inverted file. Therefore, long posting lists can be read with only one additional seek operations. In real applications, additions to the posting lists cause non-sequential secondary posting list records.

For each t value, 500 queries are generated by randomly selecting the query terms from the sample query term set. Our aim is to measure the response time for non-zero hit queries. Therefore, we continue the query evaluation for inverted file even the result contains no relevant records to the evaluated part of the query. We limit the number of retrieved relevant records with one (P = 1) as in [6]. After *anding* more than two terms, almost all of the result sets of the test queries are empty.

The results show that GMFSF provides a shorter response time for non-zero hit queries with more than three terms. For SP < 1.0, the cross-over point requires higher number of query terms than SP = 1.0. However, increasing the disk block size, B_{size}, reduces the effect of lower SP values on the response time (refer to Figure 2). For zero-hit queries the response time of inverted file depends on the technique used to obtain the empty result set.

Creation of bit map equivalents of posting lists may require shorter time compared to inverted file creation. The bit map file for the test database requires 3GB of disk memory. Compression of the bit maps reduces the space overhead, but the query evaluation time will increase to decompress the data [10]. Furthermore, during query processing we still have to access the pointer file to find the position of the bit map of the query terms. Only 4261 terms (3% of terms) of the test database have a posting list longer than 100, and only three of them are included in the 100 test query terms. Therefore, switching to bit map storage for long posting lists may reduce the space overhead, but its effect on the response time will be insignificant due to pointer file disk accesses.

7. CONCLUSION

A new signature file method, called generalized multi-fragmented signature file (GMFSF), is presented. The new method provides a unified framework for other vertical

signature file partitioning methods and employs a new partial query evaluation strategy that dynamically avoids the complete use of the on-bits of query signatures.

The performance of GMFSF is measured with a prototype information retrieval system using a database of 152,850 MARC records. The experimental results agree with the theoretical analysis. It is shown that the method provides desirable response times for multi-term queries. Using GMFSF an average response time of 1.6 seconds can be obtained for a database with 10^6 records using a disk drive with 80 ms seek time.

Unlike a recent work on vertical partitioning [6, p. 285], in GMFSF the response time decreases with an increasing number of query terms. This is due to our partial query evaluation strategy. Furthermore, the response time of GMFSF is independent of number of hits to the queries. These are desirable characteristics. A comparison with inverted files shows that GMFSF has less space overhead, and for queries with more than three terms it provides shorter reponse time. The experimental results also show that the need for GMFSF reorganization will be infrequent.

In our future research we will test GMFSF on larger databases. The inverted file approach and GMFSF will be compared using real queries submitted to BLISS. Another research topic will be the adaptation of GMFSF to parallel processing environments and its performance evaluation.

APPENDIX

Derivation of False Drop Estimation Formula for GMFSF

For a step-by-step query evaluation and false drop computation, we need to know the decrease in the false drop probability for the usage of each frame in the signature file processing. The expected number of on-frames in the rth fragment ($1 \le r \le f$) of a t term query signature can be computed by the formula used to compute signature weight.

$$FW(Q_r)_t = k_r \cdot (1 - (1 - n_r/k_r)^t) \tag{A.1}$$

At most $FW(Q_r)_t$ frames can be used in the query evaluation from the rth fragment. By using the on-frames of the lower numbered fragments first, i on-frames used for a query evaluation are distributed among the fragments such that

$$i = d + \sum_{r=1}^{h} FW(Q_r)_t \tag{A.2}$$

$$\text{where } h < f, \quad 0 \le d \le FW(Q_{h+1})_t$$

The total, number of on-frames in a query signature will be

$$FW(Q)_t = \sum_{r=1}^{f} FW(Q_r)_t \tag{A.3}$$

The expected false drop probability after processing *i* on-frames can be approximated by the multiplication of the false drop probabilities of the processed frames.

$$cffd_{(i,t)} = ffd^{d}_{(h+1,t)} \cdot \prod_{r=1}^{h} ffd^{FW(Q_r)_t}_{(r,t)} \qquad (A.4)$$

where *ffd(r,t)* is the probability of accidentally matching an on-frame in the rth fragment of the query signature with the corresponding record signature frame. *ffd(r,t)* is computed by the multiplication of two probabilities. The first one is the probability of being an on-frame of a record signature frame corresponding to a query signature on-frame. The second one is the probability of matching the bits in the query signature frame and the record signature frame.

$$ffd_{(r,t)} = (1-(1-n_r/k_r)^{D}) \cdot op_r^{W(Q_r)_t} \qquad (A.5)$$

$$op_r = 1-(1-m_r/s_r)^{E(D,k_r\cdot(1-(1-n_r/k_r)^{D}),n_r)} \qquad (A.6)$$

$$W(Q_r)_t = s_r \cdot (1-(1-m_r/s_r)^{E(t,FW(Q_r)_t,n_r)})$$

$$E(t,k,n) = \sum_{i=0}^{t} i \cdot \binom{t}{i} \cdot \left(\frac{n}{k}\right)^{i} \cdot \left(1-\frac{n}{k}\right)^{t-i}$$

op_r is the expected on-bit probability (on-bit density) in the record signature frame. $W(Q_r)_t$ is the expected number of on-bits in an on-frame of the rth fragment of the query signature. *E(t,k,n)* computes the expected number of terms which will select a frame if each of *t* terms select n frames among *k* frames by using binomial distribution.

REFERENCES

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

[2] S. Christodoulakis and C. Faloutsos, Design considerations for a message file server, *IEEE Trans. on Software Engineering* **10** (2) (1984) 201-210.

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

[4] E. Fox and W. C. Lee, FAST-INV: A fast algorithm for building large inverted files, Tech. Rept. Tr-91-10, VPI&SU, Department of Computer Science, Blacksburg, Va. 2461-0106, 1991.

[5] S. Kocberber and F. Can, Vertical Fragmentation of superimposed signature files using partial evaluation of queries, Tech. Rept., BU-CEIS-9501, Department of computer Engineering and Information Science, Bilkent University, 1995 (anonymous ftp to gopher.cs.bilkent.edu.tr pub/tech-reports/1995/BU-CEIS-9501.ps.z).

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

[7] C.S. Roberts, Partial-match retrieval via the method of superimposed codes, in: *Proceedings of the IEEE*, **67** (12) (1979) 1624-1642.

[8] R. Sacks-Davis, A. Kent and K. Ramamohanarao, Performance of multikey access method based on descriptors superimposed coding techniques, *Information Systems* **10** (4) (1987) 391-403.

[9] G. Salton, *Automatic Text Processing: The Transformation Analysis, and Retrieval of Information by Computer*, Addison-Wesley, Reading, MA. (1989)

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