Pergamon

Information Processing & Management, Vol. 33, No. 3, pp. 353–376, 1997
© 1997 Elsevier Science Ltd
All rights reserved. Printed in Great Britain
0306-4573/97 $17+0.00

PII: S0306-4573(96)00073-8

VERTICAL FRAMING OF SUPERIMPOSED SIGNATURE FILES USING PARTIAL EVALUATION OF QUERIES

SEYIT KOCBERBERA[1] and FAZLI CAN[2]*

[1] Department of Computer Engineering and Information Science, Bilkent University, Bilkent, 06533 Ankara, Turkey, and [2] Department of Systems Analysis, Miami University, Oxford, OH 45056, U.S.A.

(Received 20 October 1995; accepted 3 December 1996)

Abstract—A new signature file method, Multi-Frame Signature File (MFSF), is introduced by extending the bit-sliced signature file method. In MFSF, a signature file is divided into variable sized vertical frames with different on-bit densities to optimize the response time using a partial query evaluation methodology. In query evaluation the on-bits of the lower on-bit density frames are used first. As the number of query terms increases, the number of query signature on-bits in the lower on-bit density frames increases and the query stopping condition is reached in fewer evaluation steps. Therefore, in MFSF, the query evaluation time decreases for increasing numbers of query terms. Under the sequentiality assumption of disk blocks, in a PC environment with 30 ms average disk seek time, MFSF provides a projected worst-case response time of 3.54 seconds for a database size of one million records in a uniform distribution multi-term query environment with 1–5 terms per query. Due to partial evaluation, this desired response time is guaranteed for queries with several terms. The comparison of MFSF with the inverted file approach shows that MFSF provides promising research opportunities. © 1997 Elsevier Science Ltd

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 multi-media environments (Aktug & Can, 1997; Salton, 1989).

A signature file reflects the contents of database records in terms of bit strings. Signature files provide a space efficient fast search (index) structure by eliminating a great majority 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 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.

Several signature generation and signature file methods have been proposed to obtain a desirable response time and space overhead. A survey of the signature file methods can be found in Aktug & Can (1997) and Faloutsos (1992). The use of various forms of bitmaps as a basic tool for improving the search algorithms in medium sized information retrieval systems is described in Bookstein & Klein (1990). In this study, we consider only vertically partitioned superimposed signatures and conjunctive queries. In superimposed signature files, the length of the record signature (F) and term signatures are the same and $F>>S$. In this environment a record is defined as 'relevant' if it contains all query terms.

For a database of N records, the signature file can be viewed as an N by F bit matrix. Sequential Signature Files (SSF) require retrieval and processing of all $N{\cdot}F$ bits in the signature file. However, off-bits of a query signature have no effect on the result of the query processing,

* To whom all correspondence should be addressed.

since only the on-bits of the query signature are compared with the corresponding record signature bits. Therefore, the result of the signature file processing can be obtained by processing only the record signature bits corresponding to the on-bits of a 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) (Roberts, 1979) and generalized frame-sliced signature files (GFSSF) (Lin & Faloutsos, 1992). Vertical partitioning a signature file improves performance by reducing the amount of data to be read and processed. The partially evaluated bit-sliced signature file (P-BSSF) method improves performance of the BSSF method by using a subset of the query signature on-bits in a multi-term query environment (Kocberber & Can, 1995a; Kocberber & Can, 1995b).

In the formal development of the P-BSSF method, the number of bit-slices to be processed is determined by a stopping condition which minimizes the response time and it is independent of the number of query terms, i.e., the same number of bit-slices are processed for all queries. Therefore, the average number of bit-slices processed per query term decreases for an increasing number of query terms. However, in practice, to ensure all query terms contribute to the query evaluation, at least one bit-slice is processed for each query term. Therefore, (in practice) the lower bound of the number of bit-slices processed per query term is one. In the P-BSSF method for the queries with many terms, if the bit-slices are stored contiguously on the disk, only one disk access will be sufficient per query term contrary to the two disk accesses of the Inverted File (IF) method without in memory search structures (Zobel *et al.*, 1992).

In this paper a new signature file method, Multi-Frame Signature File (MFSF), is proposed. MFSF improves the performance by dividing the signature matrix into variable sized vertical frames (with different on-bit densities) which provides a desirable response time in a multi-term query environment. For query evaluation with MFSF, the stopping condition defined for P-BSSF (Kocberber & Can, 1995b) is adapted which provides decreasing response time for increasing number of query terms in MFSF. In multi-media environments, search conditions on text and images are expressed in a single query (Zezula *et al.*, 1991) which cause an increase in the number of query terms. Therefore, the access method of such an environment should provide acceptable response times for a high number of query terms. At the same time, a general purpose access method should also provide acceptable response times for queries containing a few query terms. Our study shows that the new method introduced in this paper provides desirable performance across the spectrum.

We compared the performance of MFSF with other vertical (signature) partitioning methods. The analysis shows that, with no space overhead, MFSF provides up to 17% and 85% query processing time improvement with respect to the P-BSSF and GFSSF organizations, respectively. With a database of 152,850 library MARC records we tested the method and compared its theoretically expected and practical behavior under various conditions and showed its scaleability for very large databases. Under the sequentiality assumption of disk blocks, MFSF provides a projected worst-case response time of 3.54 seconds for a database size of one million records in a uniform distribution multi-term query environment with 1–5 terms per query. The comparison of MFSF with the inverted file approach shows that MFSF provides promising research opportunities.

The organization of the paper is as follows. Section 2 gives a description of existing vertically partitioned signature file methods. Section 3 compares the IF method and P-BSSF in terms of the number of disk accesses. Section 4 describes the proposed MFSF. Section 5 provides a model for query processing operations and gives estimated performance of MFSF obtained with simulation runs and the results of comparisons with other vertical partitioning methods. Section 6 presents the results of the experiments with real data. Section 7 provides a theoretical comparison of the IF method and MFSF and gives future research topics. Finally, Section 8 provides the conclusions.

2. VERTICALLY PARTITIONED SIGNATURE FILES

The query evaluation with signature files is conducted in two phases. To process a query with signature files, first a query signature is produced using query terms. Then, this query signature

Record Terms	**Term Signature**	
computer	0 1 0 0 0 1 0 0 1 0	
information	0 0 0 0 1 0 0 1 0 1	
Record Signature	0 1 0 0 1 1 0 1 1 1	
Query	**Query Signature**	**Result**
access	0 1 0 0 0 1 0 0 0 1	False Drop
information	0 0 0 0 1 0 0 1 0 1	True Match
retrieval	**1** 0 0 0 1 0 **1** 0 0 0	No Match
	($F = 10, S = 3$)	

Fig. 1. Signature generation and query processing with superimposed signatures.

is compared with the record signatures. 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. Therefore, 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 (the record does not match the query). Thereby in the first phase, the signature file processing phase, most of the irrelevant records are eliminated.

Due to hashing and superimposition operations used in obtaining signatures, the signature of an irrelevant record 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 (Christodoulakis & Faloutsos, 1984; Roberts, 1979). In the second phase of the query processing, the false drop resolution phase, these possible false drop records are resolved by accessing the actual records (Aktug & Can, 1997; Faloutsos, 1992; Kocberber & Can, 1995b; Lin & Faloutsos, 1992; Roberts, 1979; Sacks-Davis *et al.*, 1985).

To illustrate signature extraction and query processing with superimposed signatures an example is provided in Fig. 1. Query signature on-bits shown in bold font have a '0' bit at the corresponding record signature positions. Since the 1st and 7th bits of the record signature are '0' while the signature of the query 'retrieval' has '1' at these positions, the record is irrelevant to this query. The record signature matches the signatures of the queries 'access' and 'information'. The on-bit positions set by the query term 'access' (2nd, 6th, and 10th) are also set by the record terms 'computer' and 'information' (2nd, 5th, 6th, 8th, 9th, and 10th). Therefore, although the record does not contain the term 'access', the record seems to qualify the query (i.e., it is a false drop).

The superimposed signature file approach represents each record with a fixed size bit string which facilitates parallel processing of search requests (Couvreur *et al.*, 1994; Grandi *et al.*, 1992; Lee, 1986; Pogue & Willett, 1987). Vertical partitioning of a signature file provides conducting the signature file processing phase of the query evaluation by retrieving a small fraction of the signature file (Aktug & Can, 1997; Faloutsos, 1992; Kocberber & Can, 1995b; Kocberber, 1996a; Lin & Faloutsos, 1992; Roberts, 1979). Parallel processing of vertically partitioned signature files is also studied in the literature (Grandi *et al.*, 1992).

Brief descriptions of the available vertical partitioning methods BSSF (Roberts, 1979), Extended Bit-Sliced Signature File (B'SSF) (Lin & Faloutsos, 1988), Generalized Frame-Sliced Signature File (GFSSF) (Lin & Faloutsos, 1992) and P-BSSF (Kocberber & Can, 1995b) are given below. Later in Section 4 we also provide a graphical representation of these methods and MFSF. The meanings of the important symbols and full names of the frequently used method acronyms of the paper are provided in Table 1.

2.1. BSSF

In BSSF each term sets S bits of a bit string that is F bits long. The value of S is determined such that the optimality condition is satisfied (Christodoulakis & Faloutsos, 1984; Roberts, 1979). BSSF requires retrieval of $W(Q)_t \cdot N$ bits instead of $F \cdot N$ bits where $W(Q)_t$ is the expected number of on-bits in the query signature (query weight) of a t term query. Usually, $W(Q)_t \ll F$; hence the amount of retrieved and processed data is reduced. Therefore, the response time of BSSF is less than the response time of SSF except for very small N values (Roberts, 1979).

Query processing with BSSF is demonstrated in Fig. 2. The database contains five records and six unique terms. The term signatures, the records with the record signatures, and sequential storage of these record signatures are shown at the top of Fig. 2. The bits in the horizontal boxes are stored sequentially from the left to the right.

BSSF storage of the signature file is shown in the middle of Fig. 2. The bits in the vertical boxes are stored sequentially from the top to the bottom. A record pointer table (RPT) is needed to store the addresses of the records. For SSF the record pointers can be stored with the record signatures.

Evaluation of the query 'access' is illustrated at the bottom of Fig. 2. To evaluate the query three bit slices (2nd, 6th, and 10th), shown with dark gray background color in BSSF, are read. The result of the signature file processing is also a bit string of length five where an on-bit indicates that the corresponding record is found relevant to the query. Only the first and second bits of the result bit string are on-bits. Therefore, the first and second record pointers are obtained by accessing RPT and then the actual (corresponding) records are read and compared with the query. Since the first record does not contain the query term 'access', it is a false drop.

Table 1. Meanings of important symbols (defining equation no.) and full names of frequently used method acronyms

Symbol	Meaning
b_s	on-bit density of sth bit slice used in query evaluation (Eq. 6)
f	number of frames in a MFSF
fd_i	false drop probability after processing i bit slices (Eqs. 6, 9)
k	number of frames in a GFSSF
m	number of bits to be set by each term in a GFSSF frame
n	number of frames selected to set bits in GFSSF
op_r	average on-bit density in rth frame (Eq. 5)
t	number of query terms
t_{max}	maximum number of terms that can be used in queries
w_t	expected total number of on-bits in all frames of a t term query signature (Eq. 4)
B_{size}	size of a disk block (bytes)
D	average number of distinct terms in a record
$E(t)$	expected number of query terms
F	size of a signature (bits)
F_r	size of rth frame of F (bits) in MFSF
FD_i	expected number of false drops after processing i bit slices (Eq. 2)
IP	improvement percentage (Eq. 19)
N	number of records in database
P_t	probability of submission of a t term query
P_{size}	size of a record pointer (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. 14)
S	number of bits set by each term
S_r	number of bits set by each term in rth frame in MFSF
SP	sequentiality probability of logically consecutive disk blocks
T_{bitop}	time required to perform bit operations between two memory words (ms)
T_{read}	time required to read a disk block
$T_{resolve}$	time required to resolve a false drop
T_{scan}	time required to scan a record to test it with query
T_{seek}	time required to position read head of disk
T_{slice}	time required to process one bit-slice
TR	expected response time (Eq. 15)
$V(t)$	variance of t
$W(Q)_{t,r}$	expected number of on-bits in rth frame for a t term query (Eq. 4)
W_{size}	size of a memory word (in bytes)
Acronym	Meaning
GFSSF	Generalized Frame-Sliced Signature File
MFSF	Multi-Frame Signature File
P-BSSF	Partially Evaluated Bit-Sliced Signature File

2.2. *B'SSF: the enhanced version of BSSF*

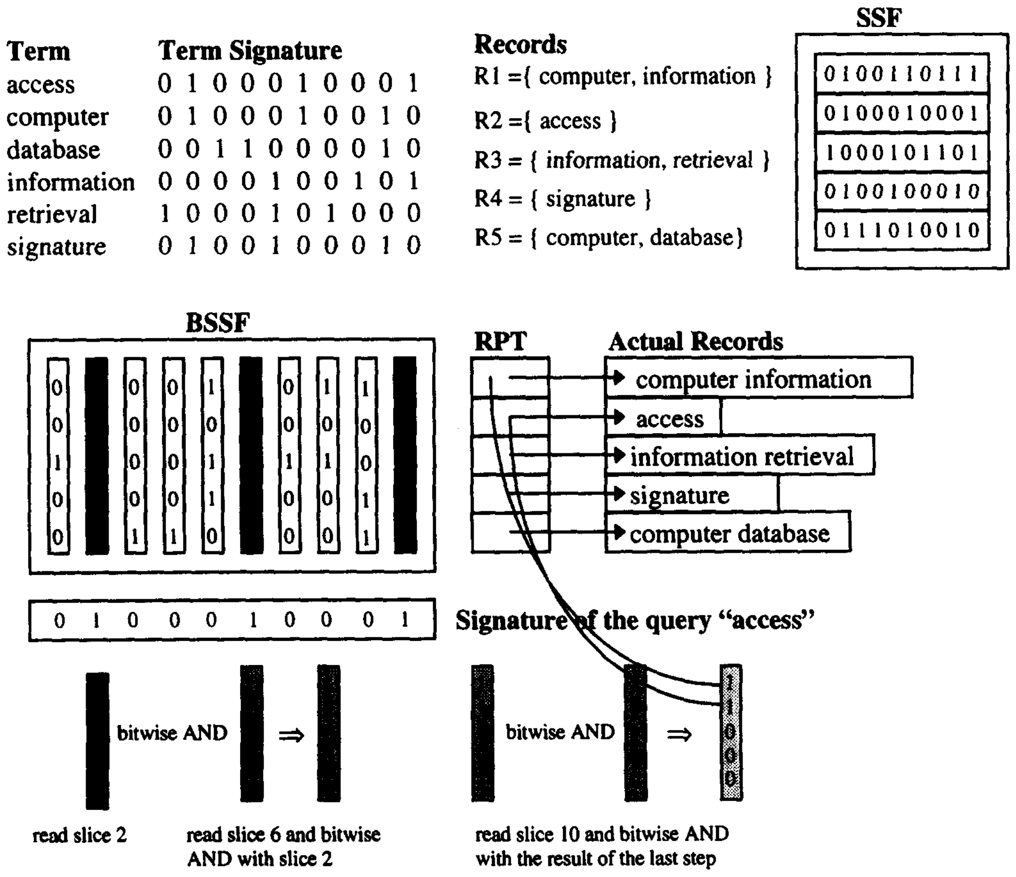
For BSSF, the optimality condition requires a larger S value for a larger signature size (F) (Roberts, 1979). Increasing S also increases the query weight and the number of retrieved bit slices. Consequently, except for small F values, increasing F also increases the response time in the BSSF method.

In the B'SSF method, the optimality condition is relaxed and the response time is minimized for single term queries instead of minimizing the false drop probability (Lin & Faloutsos, 1988). An optimized B'SSF configuration for a minimum response time may have a smaller S value than a BSSF requires. The value of S decreases for increasing F value. Therefore, the response time of B'SSF decreases for increasing F value. The formula to find the optimum S value can be found in Lin & Faloutsos (1988).

2.3. *GFSSF*

Current auxiliary storage seek time is much larger than the read time per disk block. GFSSF provides improvement over B'SSF (Lin & Faloutsos, 1988) by minimizing the number of seek operations (Lin & Faloutsos, 1992). GFSSF optimizes the signature file parameters to minimize the response time for a given number of query terms.

In GFSSF, a signature is divided into k frames, each of size s bits ($s = F/k$). Each term first randomly selects n ($1 \leq n \leq k$) frames, then randomly sets m ($1 \leq m \leq s$) bits (not necessarily distinct) in each of the selected frames (Lin & Faloutsos, 1992). In this method, the size of a



Term	Term Signature
access	0 1 0 0 0 1 0 0 0 1
computer	0 1 0 0 0 1 0 0 1 0
database	0 0 1 1 0 0 0 0 1 0
information	0 0 0 0 1 0 0 1 0 1
retrieval	1 0 0 0 1 0 1 0 0 0
signature	0 1 0 0 1 0 0 0 1 0

Records
R1 ={ computer, information }
R2 ={ access }
R3 = { information, retrieval }
R4 = { signature }
R5 = { computer, database}

$(N = 5, F = 10, S = 3)$

Fig. 2. SSF and BSSF organizations and BSSF query processing example.

frame is $s{\cdot}N$ bits and each frame is stored separately as a SSF. The methods SSF, BSSF, and B'SSF are special cases of GFSSF (Lin & Faloutsos, 1992) and GFSSF outperforms all of its special cases.

2.4. *P-BSSF*

In the B'SSF method, the response time is minimized for single term queries. In a multi-term query environment, which is the case in real environments, the optimized configuration of a B'SSF unnecessarily requires processing of additional bit slices for the queries with more than one term (Kocberber & Can, 1995b).

P-BSSF solves this problem by using a partial query evaluation technique. The response time is minimized in a multi-term query environment by employing the partial evaluation strategy and by considering the submission probabilities of the queries with different numbers of terms (Kocberber & Can, 1995b). The technique employs a stopping condition that tries to complete the signature file processing phase of 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 an acceptable level that will also provide the lowest response time within the framework of P-BSSF (Kocberber & Can, 1995b).

In P-BSSF the signature file query evaluation stops when the time required to evaluate a bit slice becomes equal to (or greater than) the time required to resolve the false drops which will be eliminated by processing this bit slice. This stopping condition minimizes the response time of P-BSSF and is expressed as follows (Kocberber & Can, 1995b).

$$T_{\text{slice}}=FD_i{\cdot}(1-op){\cdot}T_{\text{resolve}} \tag{1}$$

$$FD_i=N{\cdot}op^i \tag{2}$$

$$op=1-(1-S/F)^D \tag{3}$$

where T_{slice} is the time required to process one bit-slice, i is the number of the processed bit-slices, FD_i is the number of expected false drops after processing i bit-slices, T_{resolve} is the time required to resolve a false drop, op (on-bit density) is the ratio of the number of on-bits to the total number of bits in the signature file, N is the number of records in the database, and D is the average number of distinct terms in a record.

3. IF VERSUS P-BSSF

P-BSSF performs better than other vertical partitioning methods such as BSSF and GFSSF (Kocberber & Can, 1995b). Therefore, in the following discussion, we compare the IF method and the P-BSSF method in terms of number of disk accesses (seek requests) since it is expected that the response time will be proportional to the number of seeks. Our aim is to show that P-BSSF is not inferior to the IF method and this provides the motivation of this study since our new MFSF method outperforms P-BSSF.

In the IF method, each term is associated with a list of identifiers (or pointers) of the records containing this term. At least one disk access is required per query term to read the posting list of the term (we ignore chained long posting lists). Also, to obtain the locations of the posting lists, a term 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 (second disk access is to retrieve the posting list and we assume that the pointers to the actual records are stored in main memory) (Zobel *et al.*, 1992).

In P-BSSF no lookup table is needed (note that term signatures are directly generated from actual terms). To obtain the pointers of the records that satisfy the search query, the bit slices corresponding to the on-bits of the query signature must be accessed and bitwise ANDded. We assume one disk accesses will be required to retrieve a bit-slice (we ignore chained bit slices as

$D = 25.7, N = 10^5, t = 1$

F	S	op	FD	# of seeks
1000	6	0.143	0.855	6.855
2000	4	0.050	0.625	4.625
3000	4	0.034	0.134	4.134
10000	3	0.008	0.051	3.051

	Number of Query Terms (t)					
F	1	2	3	4	5	6
1000	6.9	6.9	6.9	6.9	6.9	6.9
2000	4.6	4.6	4.6	4.6	5.0*	6.0
3000	4.1	4.1	4.1	4.1	5.0	6.0
10000	3.0	3.0	3.0	4.0	5.0	6.0

$D = 25.7, N = 10^6, t = 1$

F	S	op	FD	# of seeks
1000	8	0.187	1.464	9.464
2000	5	0.062	0.916	5.916
3000	5	0.042	0.131	5.131
10000	3	0.008	0.512	3.512

	Number of Query Terms (t)					
F	1	2	3	4	5	6
1000	9.5	9.5	9.5	9.5	9.5	9.5
2000	5.9	5.9	5.9	5.9	5.9	6.1
3000	5.1	5.1	5.1	5.1	5.1	6.0
10000	3.5	3.5	3.5	4.0	5.0	6.0

Fig. 3. Estimated number of disk accesses for P-BSSF for $N = 10^5$ and $N = 10^6$ (D value is taken from the experiments of Section 5). *The fractional parts become zero due to rounding. Actual values are greater than zero.

in the posting lists of the IF method). Since the pointers to the actual records are stored in main memory, only one disk access will be required to read a false drop record.

The number of expected false drops decreases as the number of processed bit-slices increases (see equation (2)). However, the stopping condition of P-BSSF puts an upper bound to the number of processed bit-slices such that after this point additional bit-slice processing just increases the response time (Kocberber & Can, 1995b). To show the relation between the signature file parameters F, S, D, N, the number of processed bit-slices (i), and FD we provide estimated number of disk accesses for $N = 10^5$ and $N = 10^6$ for P-BSSF in Fig. 3.

To simplify the signature file optimization procedure we optimized the signature file parameters for single term queries ($t = 1$) and used the same configuration for multi-term queries with the stopping condition. The stopping condition minimizes the total number of disk accesses, which is equal to ($S + FD$). (Note that P-BSSF obtains better response times by considering the submission probabilities of queries with different numbers of query terms.)

At the upper parts of Fig. 3 (for $t = 1$), the optimum S values and the number of disk accesses decreases for increasing F values. Since each term sets fewer number of bits for increasing F, the *op* decreases and signature file processing requires fewer disk accesses.

The same signature file configuration is used for multi-term queries. S disk accesses are required for $t \leq S$ and the expected FD values are the same. To make sure that each query term contributes to the query evaluation, at least one bit-slice is processed for each query term. Therefore, for $t > S$ even the number of expected false drops is less than one, we assume t bit slices are used in the query evaluation. For this reason, the expected FD values decreases for increasing t values.

For $F = 1000$ and $N = 10^5$, P-BSSF requires fewer disk accesses than IF for the queries containing more than three terms. For the same *op* value, the number of expected false drops increases for increasing N. Therefore, the number of processed bit slices, hence the number of disk accesses, increases for increasing N.

For $F = 2000$, $N \leq 10^6$, and $t > 2$, P-BSSF requires fewer disk accesses than IF. If F is increased to 10,000, for $t = 2$ P-BSSF requires only 3.5 disk accesses which is less than four disk accesses of IF. Note that higher F values will require fewer disk accesses while the space overhead of P-BSSF increases.

4. PROPOSED SIGNATURE FILE METHOD

The probability of a particular bit of a bit slice being on-bit is the *op*. Low *op* provides rapid reduction in the expected number of false drops. Thus, the stopping condition defined for P-

Table 2. Properties of vertical signature file partitioning methods

Properties/signature file methods	BSSF	B'SSF	GFSSF	P-BSSF	MFSF
On-bit density (*op*) <0.5 is allowed	No	Yes	Yes	Yes	Yes
Optimized in multi-term query env.	No	No	No	Yes	Yes
Partial evaluation strategy defined	No	No	No	Yes	Yes
Obtaining the optimum configuration	Exact	Exact	Heuristic	Exact	Heuristic

BSSF is reached by processing fewer number of bit slices (Kocberber & Can, 1995b).

For a given D value, *op* can be reduced by either increasing F or decreasing S (see equation (3)). For P-BSSF, the value of S is selected to obtain the minimum response time in a multi-term query environment. Therefore, decreasing S will produce insufficient on-bits in the query signature of low-weight queries and the number of false drops will increase for these queries. This will also increase the response time.

The performance of P-BSSF can be improved if the *op* can be reduced while providing enough on-bits in the query signature of low weight queries. We propose a new signature generation and query evaluation method, MFSF, which improves the performance of P-BSSF without increasing the space overhead (F value). MFSF decreases the response time in multi-term query environments by dividing the signature file into variable sized sub-signature files, *frames* of bit slices. Each frame is a separate BSSF with its own F and S parameters and the optimality condition is relaxed. In MFSF each frame may have a different on-bit density. A summary of the vertical partitioning methods is given in Table 2.

4.1. MFSF

A MFSF is a combination of f sub-signature files, *frames*, such that $F=F_1+F_2...+F_f\,(f\leq F)$. Since the bit slices of a BSSF are stored separately, dividing the signature file into sub-signature files can be accomplished conceptually without changing the physical storage structure of the BSSF method. Each term sets S_r bits in the rth frame such that $S=S_1+S_2...+S_f\,(0<S_r<F_r, 1\leq r\leq f)$.

Since each frame is a BSSF, we use the same formulas as were used for BSSF and compute the expected query weights of the frames ($W(Q)_{(r,t)}$) and the expected total query weight (w_t) for a t term query as follows.

$$W(Q)_{(r,t)}=F_r\cdot(1-(1-{}^{S_r}/_{F_r})^t)\ \mathit{for}\ 1\leq r\leq f \tag{4}$$

$$w_t=\sum_{r=1}^{f} W(Q)_{(r,t)}$$

The *op* values of the frames are

$$op_r=1-(1-{}^{S_r}/_{F_r})^D \text{ for } 1\leq r\leq f. \tag{5}$$

Graphical representations of SSF, BSSF, B'SSF, P-BSSF, GFSSF, and MFSF are illustrated in Fig. 4. A horizontal box represents the sequential storage of the bits in the box. First are stored the bits of the first box, then the bits of the second box and so on. A vertical box represents the sequential storage of the bits in the box from the top to the bottom. The *op* values of the bit strings are represented with the gray level of the box. A darker area has higher *op* than the lighter one. Note that the highest *op* is 0.5 (case a and case b).

4.2. False drop computation for MFSF

We define fd_i as the false drop probability if i bit-slices ($i\leq w_t$) are used in the signature file processing phase of a query evaluation. fd_i is computed by multiplying the *op* of the bit slices used for the query evaluation as follows.

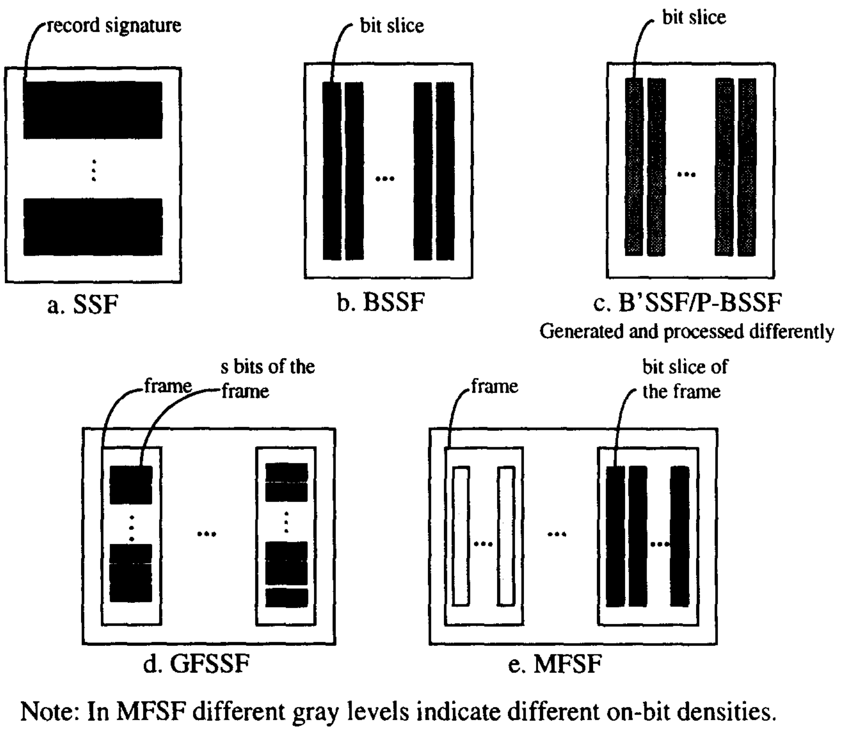


Fig. 4. Graphical representation of SSF and vertical partitioning methods.

$$fd_i = \prod_{s=1}^{i} b_s \tag{6}$$

where $b_s = op_r$ if the sth slice used for query evaluation is selected from the rth frame.

If the number of bit slices used for query evaluation, i, is less than total query weight ($i < w_t$), which is usual in the partial evaluation approach, the selection order of the bit slices used for the query evaluation may change the false drop probability. Therefore, the frames of MFSF are ordered in non-decreasing op value such that

$$op_r \leq op_{r+1} \text{ for } 1 \leq r < f \tag{7}$$

holds for all frames.

In the query evaluation the on-bits of the lower op frames are used first. This rule is specified as

$$b_s \leq b_{s+1} \text{ for } 1 \leq s < w_t \tag{8}$$

and ensures that the stopping condition is reached in fewest evaluation steps. As the number of query terms increases, the expected number of query signature on-bits in the lower op frames increases. Therefore, the stopping condition will be reached in fewer evaluation steps and the query evaluation time will decrease for increasing number of query terms (later we provide a numerical example for this).

If we consider only one frame, say frame r, and all query signature on-bits of this frame are used in the query evaluation, the false drop probability of this frame becomes $op_r^{W(Q)_{(r,t)}}$ (note that $W(Q)_{(r,t)}$ is the expected query weight of rth frame for a t term query). If d on-bits are used from a frame, say the $h+1$st frame, the inequalities (7) and (8) ensure that all of the query signature on-bits of the lower numbered frames (the first h frames) were already used in the query evaluation. Therefore, the number of bit slices used in the query evaluation, i, is computed by adding the query weights of these lower numbered frames and d.

$$i = d + \sum_{r=1}^{h} W(Q)_{(r,t)} \text{ where } h < f, 0 \leq d \leq W(Q)_{(h+1,t)}$$

Similarly, the false drop probability can be computed by multiplying the false drop probabilities of the first h frame and op_{h+1}^{d} since only d on-bits are used from $h+1$st frame.

$$fd_i = op_{h+1}^{d} \cdot \prod_{r=1}^{h} op_r^{W(Q)_{(r,t)}} \tag{9}$$

If there is only one frame, i.e., $f=1$, then $h=0$, $d=i$, and $fd_i = op^i$. In this case, a MFSF converges to a P-BSSF. Consequently, B'SSF and P-BSSF are special cases of MFSF.

4.3. Stopping condition for MFSF

To derive the stopping condition for MFSF, first we obtain a general stopping condition for vertically partitioned signature files and then we will apply this formula to MFSF. We define RFD_{i+1}, the number of *reduced false drops*, as the number of false drops which will be eliminated by processing an additional bit slice after processing i bit slices. We derive the formula for RFD_{i+1} as follows.

$$\begin{aligned} RFD_{i+1} &= FD_i - FD_{i+1} \\ &= N \cdot op^i - N \cdot op^{i+1} \\ &= N \cdot op^i \cdot (1 - op) \end{aligned} \tag{10}$$

At the stopping step (i) time required to process one bit slice (T_{slice}) must be equal to (or greater than) the time required to resolve RFD_i false drops. Therefore, the stopping condition becomes

$$T_{\text{slice}} = RFD_{i+1} \cdot T_{\text{resolve}}. \tag{11}$$

This stopping condition is independent of the false drop computation method and is explained as follows: at the stopping step the false drops which will be eliminated by processing the next bit slice can be checked by accessing the actual records in equal or less time than eliminating these false drops by using the signature file.

To obtain the stopping condition for MFSF we derive the formula to compute RFD_{i+1}. The false drop probability after processing $i+1$ bit slices is $fd_{i+1} = fd_i \cdot b_{i+1}$ where b_{i+1} is the false drop probability of $i+1$st bit slice used in signature file processing. All query signature on-bits of the first h frames and d on-bits of $h+1$st frame are used to process i bit slices (see equation (9)). Therefore, if there is an unused on-bit in $h+1$st frame, i.e., if $d < W(Q)_{(h+1,t)}$, b_{i+1} will be equal to op_{h+1}. If all on-bits of $h+1$st frame are already used, i.e., $d = W(Q)_{(h+1,t)}$, the $i+1$st on-bit will be selected from $h+2$nd frame if $h+2$nd frame exists ($h+2 \leq f$). By considering this discussion the value of b_{i+1} is determined as follows.

$$b_{i+1} = \begin{cases} op_{h+1} \ \textit{if}\ d < W(Q)_{(h+1,t)} \\ op_{h+2} \ \text{otherwise}\ (\textit{if}\ h+2 > f \ \text{query evaluation is completed}) \end{cases} \tag{12}$$

where $h < f$, $0 \leq d \leq W(Q)_{(h+1,t)}$, and $i = d + \sum_{r=1}^{h} W(Q)_{(r,t)}$.

RFD_{i+1} for MFSF is computed as follows.

$$RFD_{i+1} = N \cdot fd_i - N \cdot fd_i \cdot b_{i+1}$$

$$RFD_{i+1} = N \cdot fd_i \cdot (1 - b_{i+1})$$

We obtain the following stopping condition for MFSF by substituting RFD_{i+1} in equation (11).

$$T_{\text{slice}} = N \cdot fd_i \cdot (1 - b_{i+1}) \cdot T_{\text{resolve}} \tag{13}$$

To prove the stopping condition given in equation (13) is valid in subsequent steps we have to consider the following theorem.

Theorem. The number of false drops eliminated in successive evaluation steps, *RFD* (the number of Reduced False Drops), decreases.

Proof. RFD_{i+1} is the number of false drops that can be eliminated by processing one more bit slice after processing i bit slices for $1 \leq i < w_t$, where w_t is the expected total query weight for a t term query.

Now show that $RFD_{i+1} > RFD_{i+2}$

$N \cdot fd_i \cdot (1 - b_{i+1}) > N \cdot fd_{i+1} \cdot (1 - b_{i+2})$

$N \cdot fd_i \cdot (1 - b_{i+1}) > N \cdot fd_i \cdot b_{i+1} \cdot (1 - b_{i+2})$

$(1 - b_{i+1}) > b_{i+1} \cdot (1 - b_{i+2})$

Since $b_{i+1} \leq b_{i+2}$ for $1 \leq i < w_t - 1$ (see inequality 8)
$1 - b_{i+1} \geq 1 - b_{i+2} > (1 - b_{i+2}) \cdot b_{i+1}$ holds and RFD is decreasing.

Since the cost of processing a bit slice is the same in all frames, the above proof guarantees that once the stopping condition given in equation (13) is satisfied, it will be valid in subsequent steps.

4.4. *Searching the optimum configuration*

We define the response time as the time needed to find the first relevant record to the query, if any, as defined in Kocberber & Can (1995b); Lin & Faloutsos (1988)and Lin & Faloutsos (1992). In query processing index structures are used to eliminate the irrelevant records. However, usually, the actual records relevant to the query are further processed before presenting them to the user. For example, a report generator has to access actual records after processing the search index to print them. Similarly, information retrieval applications display the first screen of the relevant records and access the remaining relevant records upon user requests. Also, in the client-server computing technology, like the cursor processing in commercial relational database systems, the server sends relevant records of a query in batches upon the requests of the client. Therefore, our response time definition, which requires processing of the search index and finding the first relevant record, coincides with real applications. We assume FD_i record accesses will be performed to find the first relevant record to the query. In this way all false drop records must be accessed and eliminated before all true matches. This is a worst-case assumption since it assumes that false drops are accessed first and we used this approach in our response time calculations.

To find the first relevant record, the signature file processing phase must be completed which requires retrieval and processing of i bit slices (i is determined by using the stopping condition given in equation (13). The response time for a t term query with i slice processing and FD_i actual record accesses is computed as follows.

$$RT_t = i \cdot T_{\text{slice}} + FD_i \cdot T_{\text{resolve}} \qquad (14)$$

Before passing let us consider the following question: "How would equation (14) change if the criterion were not to find the first relevant record, but all relevant records?" For a query with NR relevant records, to find all such records (i.e., to determine the relevant record pointers) $FD_i + NR$ record accesses are needed and equation (14) becomes $RT_t = i \cdot T_{slice} + (FD_i + NR) \cdot T_{resolve}$.

Note that, NR is independent of signature file parameters, i.e., NR record accesses are always needed independent of the number of processed bit-slices. In other words, the product $NR \cdot T_{resolve}$ is always need to be added to the minimized response time in equation (14); therefore, minimizing equation (14) also minimizes the response time even the criterion is finding all relevant records.

Since MFSF optimizes the response time in a multi-term query environment, we consider the submission probabilities of queries with different numbers of query terms as follows in determining the (expected) response time, TR.

$$TR = \sum_{t=1}^{t_{\max}} P_t \cdot RT_t \qquad (15)$$

where RT_t, given in equation (14), 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.

```
Algorithm SearchConfiguration
$f \leftarrow$ Select randomly the number of frames ($1 \le f \le F$).
Set $F_r$ values randomly ($1 \le r \le f$) where $F = F_1 + F_2 + \cdots + F_f$.
Set $S_r$ values to 1 ($1 \le r \le f$).
Mark all frames and all operations in the frames as not-tried.
minimum_response_time $\leftarrow$ infinity.
while there are not-tried frames
    { $r \leftarrow$ Select randomly a not-tried frame ($1 \le r \le f$).
     Select randomly a not-tried operation among the operations split, increase $S_r$, decrease $S_r$,
         increase $F_r$, decrease $F_r$ for frame $r$
    if a not-tried operation exist
        { if the selected operation is applicable
            { Apply the operation and obtain candidate configuration.
              if response time, TR, of the candidate configuration is less than minimum_response_time
                 { Accept the candidate as the new configuration, minimum_response_time $\leftarrow$ TR.
                   Mark all frames and all operations in the frames as not-tried.
                 }
                 else
                   Mark the selected operation in frame $r$ as tried.
            }
            else
              Mark the selected operation in frame $r$ as tried.
        }
        else
          Mark frame $r$ as tried.
    }
```

Fig. 5. Algorithm to search optimal framing scheme.

The values of the parameters N and D involved in the response time computation depend on the database instance. Therefore, minimizing the response time, *TR*, with the stopping condition given in expression (13) requires determination of parameters f, F_r and S_r ($1 \le r \le f$) for a given F value. The heuristic search algorithm outlined in Fig. 5 is used to search the optimum configuration and to determine the *TR* value for this case.

The algorithm starts with a randomly determined initial multi-framed scheme. A candidate configuration is obtained by changing the value of a randomly chosen parameter. Since the algorithm minimizes the response time for a given F value, *Join Frames, Increase* F_r, and *Decrease* F_r operations of the algorithm require random selection of another frame, p, and adjusting the F_p value of this frame accordingly. In the algorithm, joining of two frames to form one frame is initiated when *decrease* S_r is selected and the S_r value in the selected frame is one. After obtaining the candidate configuration, the consistency of the parameters is ensured such as $1 \le S_r \le F_r$ holds for $1 \le r \le f$. 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 20 initial trials.

The convergence time of the algorithm depends on the number of initial frames randomly selected at the beginning of the algorithm. To speed up the convergence time we limit the maximum number of frames in the initial configuration to 20, which gives similar results with a higher number of initial frames. The average convergence time of the algorithm for one randomly selected initial configuration measured by elapsed time on a 33 MHz 486 DX personal computer is 2.34 seconds. Since we tried 20 randomly selected initial configurations, the average elapsed time required to obtain the optimized configuration for a given F value is 46.8 seconds.

4.5. Example MFSF configuration

To illustrate the computation of *TR* values of P-BSSF and MFSF, we provide a numerical example in Fig. 6. The configurations are obtained by using the values of our experimental parameters (later provided in Section 5). The optimized configuration and the stopping step, *i*, for P-BSSF are obtained as proposed in Kocberber & Can (1995b).

Except $t=1$, the response time of P-BSSF remains unchanged for an increasing number of query terms. In MFSF, for an increasing number of query terms, since the stopping condition is reached in fewer evaluation steps the response time decreases.

5. PERFORMANCE ESTIMATION AND EVALUATION

Various performance measures were used in the literature. Some of them are the number of seek operations (Kent *et al.*, 1990), the signature reduction ratio (Lee & Leng, 1989), the computation reduction ratio (Lee *et al.*, 1995), and the response time (Kocberber, 1996a; Lin & Faloutsos, 1992; Roberts, 1979; Salton, 1989). Some of these measures are not applicable to all indexing methods. For example, the signature reduction ratio is meaningless for the IF method. Consequently, there may be difficulties in the performance comparisons between the methods if a common performance measure is not used. Since the primary goal of all information retrieval methods is to obtain a desirable response time, we used the response time as a performance measure. In this way we can compare the performance of MFSF with any other indexing method and estimate its performance in real life.

To estimate the response time of MFSF, we modeled the operations involved in evaluating a query with signature files. The values of the database parameters are obtained by inspecting 152,850 MARC records from the Bilkent University Library collection (BLISS-1). The database contains 166,216 unique terms. We used MARC records since they are widely used to store and

$N = 10^6$, $SP = 1$, $F = 1200$, $T_{slice} = 153$ ms, $T_{resolve} = 76$ ms, $t_{max} = 5$, $P_t = 0.2$ for $1 \le t \le 5$

MFSF Configuration

$f = 4$, $F_1 = 451, S_1 = 1, op_1 = 0.055$
$F_2 = 254, S_2 = 1, op_2 = 0.096$
$F_3 = 137, S_3 = 1, op_3 = 0.172$
$F_4 = 358, S_4 = 4, op_4 = 0.251$

P-BSSF Configuration

$S = 6, op = 0.121, i = 7$

Response Time Calculation for MFSF

t	i*	fd_i	FD_i	RT_i(ms)
1	7	$0.055 \cdot 0.096 \cdot 0.172 \cdot 0.251^4 = 3.60 \cdot 10^{-6}$	3.60	$7 \cdot 153 + 3.60 \cdot 76 = 1344.6$
2	6	$0.055^2 \cdot 0.096^2 \cdot 0.172^2 = 0.825 \cdot 10^{-6}$	0.83	$6 \cdot 153 + 0.825 \cdot 76 = 980.7$
3	5	$0.055^3 \cdot 0.096^2 = 1.53 \cdot 10^{-6}$	1.53	$5 \cdot 153 + 1.53 \cdot 76 = 881.3$
4	5	$0.055^4 \cdot 0.096 = 0.878 \cdot 10^{-6}$	0.88	$5 \cdot 153 + 0.878 \cdot 76 = 831.7$
5	5	$0.055^5 = 0.50 \cdot 10^{-6}$	0.50	$5 \cdot 153 + 0.50 \cdot 76 = 803$

$TR\ for\ MFSF = 0.2 \cdot 1344.6 + 0.2 \cdot 980.7 + 0.2 \cdot 881.3 + 0.2 \cdot 831.7 + 0.2 \cdot 803 = 968.3\ ms$

Response Time Calculation for P-BSSF

t	i*	fd_i	FD_i	RT_i(ms)
1	6	$0.121^6 = 3.14 \cdot 10^{-6}$	3.14	$6 \cdot 153 + 3.14 \cdot 76 = 1156.6$
2	7	$0.121^7 = 0.38 \cdot 10^{-6}$	0.38	$7 \cdot 153 + 0.38 \cdot 76 = 1099.9$
3	7	$0.121^7 = 0.38 \cdot 10^{-6}$	0.38	$7 \cdot 153 + 0.38 \cdot 76 = 1099.9$
4	7	$0.121^7 = 0.38 \cdot 10^{-6}$	0.38	$7 \cdot 153 + 0.38 \cdot 76 = 1099.9$
5	7	$0.121^7 = 0.38 \cdot 10^{-6}$	0.38	$7 \cdot 153 + 0.38 \cdot 76 = 1099.9$

$TR\ for\ P\text{-}BSSF = 0.2 \cdot 1156.6 + 0.2 \cdot 1099.9 + 0.2 \cdot 1099.9 + 0.2 \cdot 1099.9 + 0.2 \cdot 1099.9 = 1111.2\ ms$

Fig. 6. Example response time calculations for MFSF and P-BSSF. *i stands for the number of slices used to reach the stopping condition.

Table 3. Parameter values for the simulation runs and experiments

Variable	Meaning
$t_{max}=5$	maximum number of terms in a query
$B_{size}=8192$	size of a disk block (bytes)
$D=25.7$	average number of distinct terms in a record
$N=152{,}850$	number of records
$P_{size}=4$	size of a record pointer (bytes)
$PB=2048$	number of record pointers in the record pointer buffer
$RB=1$	average number of disk block accesses to retrieve a record
$T_{bitop}=0.00098$	time required to perform bit operations between two memory words (ms)
$T_{read}=5.77$	time required to read a disk block (ms)
$T_{scan}=4.5$	average time required to match a record with query (ms)
$T_{seek}=30$	average time required to position the read head of disk (ms)
$W_{size}=4$	size of a memory word (bytes)

distribute the bibliographic information about various types of materials such as books, films, slides, videotapes, etc.

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. Also, controlling the sequentiality probability (defined in the next section) is easy in the DOS environment. A current multi-user system can offer computing powe: and I/O speed equivalent to our experimental environment if not better. So the results of the experiments can be achieved in multi-user environments without a performance degradation. The values of the variables are determined experimentally and they are given in Table 3.

5.1. *Modeling the query processing operations*

The basic operation to be modeled is reading a specified amount of data from the auxiliary storage. Data are written and read in blocks and the physical layout of the data on the auxiliary storage affects the reading time. Therefore, we incorporate the sequentiality probability, *SP*, into the estimation of the time required to read *b* logically consecutive disk blocks (Kocberber & Can, 1995b; Lin & Faloutsos, 1992). *SP* is the probability of reading the next logically consecutive disk block without a seek operation. We estimate the time required to read *d* logically consecutive disk blocks as follows (Kocberber & Can, 1995b).

$$\text{Read}(d)=(1+(d-1)\cdot(1-SP))\cdot T_{seek}+d\cdot T_{read} \tag{16}$$

where T_{seek} and T_{read} are average times required to position the read head of the disk to the desired block (i.e., it includes the rotational latency time) and to transfer a disk block to memory, respectively. The first disk block of each request will always require a seek operation.

To process a bit slice, the bit slice must be read and ANDed with the result of the processed bit slices. By assuming two bit slices will be stored in main memory, the time required to process a bit slice, T_{slice}, is computed as follows.

$$T_{slice}=\text{Read}\left(\left\lceil \frac{N}{8\cdot B_{size}} \right\rceil\right)+T_{bitop}\cdot\left\lceil \frac{N}{8\cdot W_{size}} \right\rceil \tag{17}$$

where B_{size} is the size of a disk block and W_{size} is the size of a memory word, both in bytes. T_{bitop} is the time required to perform a bitwise AND operation between two memory words and store the result in one of the words.

Usually, data records are variable lengths and a lookup table is used to find the record pointer of the actual record (see Fig. 2). Since MARC records are variable lengths, we needed a lookup table and we modeled one obtaining a record pointer as follows. At the database initialization stage *PB* record pointers, each occupying P_{size} bytes, are read into a buffer of $PB\cdot P_{size}$ bytes. Since this is a one time cost, it is excluded from the response time calculations. The probability of finding a requested record pointer in the buffer is approximately equal to *PB/N*. For databases with fixed length records or when all record pointers are stored in the main memory, PB must be equal to N, i.e., the cost of finding the record pointers is zero.

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

$$T_{\text{resolve}}=(1-PB/N)\cdot\text{Read}\left(\left\lceil\frac{PB\cdot P_{\text{size}}}{B_{\text{size}}}\right\rceil\right)+\text{Read}(RB)+T_{\text{scan}} \quad (18)$$

where the first component of T_{resolve} is the time needed to read the necessary record pointers, RB is the average number of disk blocks that must be accessed to read a record, and T_{scan} is the time required to compare a record with the query.

5.2. Performance measurement and comparison with simulation runs

5.2.1. Performance measurement. We used the *improvement percentage* (IP) value in the comparison of the performance of MFSF with GFSSF and P-BSSF. Note that BSSF and B'SSF are special cases of both P-BSSF and GFSSF. Therefore, we exclude BSSF–MFSF and B'SSF–MFSF cases in the comparisons. The improvement percentage provided by MFSF with respect to GFSSF (GFSSF–MFSF) is defined in terms of the response times of the methods involved as

$$\text{IP}_{\text{GFSSF - MFSF}}=100\cdot(TR_{\text{GFSSF}}-TR_{\text{MFSF}})/TR_{\text{GFSSF}}, \quad (19)$$

and a similar definition is used for the P-BSSF–MFSF case.

The same T_{resolve} value is used in response time calculations of GFSSF, P-BSSF, and MFSF. The GFSSF method uses a different storage structure. Therefore, T_{slice} for GFSSF is computed by considering the frames of GFSSF. The false drop probability computation method proposed in Lin & Faloutsos (1992) requires extensive computations to optimize a configuration. Therefore, we computed the false drop probability of GFSSF by using the approximation proposed in Kocberber & Can (1995c). The false drop probability computation method proposed in Kocberber & Can (1995c) converges to the false drop computation method of B'SSF for a frame size of one bit. B'SSF is a special case of both GFSSF and MFSF and in most of the inspected cases, GFSSF converges to B'SSF producing a frame width of one bit. Therefore, this approximation works well for GFSSF.

The optimization method of GFSSF is defined for a given number of query terms (Lin & Faloutsos, 1992). Since there may be queries with different numbers of query terms in a multi-term query environment, we obtained TR value for GFSSF as follows. First, we obtained the optimized configuration of GFSSF $t=1$ as proposed in Lin & Faloutsos (1992). Then, we computed TR value of this configuration by considering the probability distribution of the number of query terms (P_t values) in the inspected multi-term query environment. We repeated the same computations for $t=2$, $t=3$, $t=4$, and $t=5$ and we obtained five different TR values. We selected the minimum TR value among these five as the TR value of the inspected case. In other words, in our comparisons our treatment of GFSSF is more than fair. In most of the inspected cases, the configuration optimized by taking $t=1$ gives minimum response time in a multi-term query environment.

To simulate a multi-term query environment, P_t values are determined by assuming a bounded normal distribution from left and right. The changes in P_t values are modeled by changing variance, $V(t)$, and the expected number of query terms, $E(t)$, values ($1\leq t\leq 5$). P_t values for the inspected $V(t)$ and $E(t)$ values are given in Table 4. The difference among P_t values, hence the effect of changing $E(t)$ values, decreases for $V(t)$ values greater than or equal to 10. Consequently, P_t values are approximately equal for these distributions ($V(t)\geq 10$) and they are modeled by the uniform distribution (UD) where all P_t values are equal to 0.2 and invariant of the change in $E(t)$. Therefore, we consider only $V(t)=1$ and $V(t)=5$. The case $V(t)=0$ implies an

Table 4. P_t Values for $V(t)=1$ and $V(t)=5$

P_t	$V(t)=1$					$V(t)=5$				
	$E(t)=1$	$E(t)=2$	$E(t)=3$	$E(t)=4$	$E(t)=5$	$E(t)=1$	$E(t)=2$	$E(t)=3$	$E(t)=4$	$E(t)=5$
P_1	0.553	0.258	0.061	0.006	0.000	0.311	0.231	0.161	0.105	0.064
P_2	0.351	0.412	0.246	0.066	0.009	0.284	0.257	0.218	0.173	0.129
P_3	0.088	0.259	0.388	0.260	0.089	0.211	0.233	0.241	0.233	0.211
P_4	0.008	0.065	0.244	0.410	0.350	0.129	0.173	0.218	0.257	0.284
P_5	0.000	0.006	0.061	0.258	0.552	0.065	0.106	0.162	0.232	0.312

environment with queries only with t number of terms, i.e., $P_t = 1$. Since it is unrealistic we omit this case.

The response times, and consequently the IP values, of the inspected methods are affected by the values of the parameters N, F, SP, $t_{\max}$ and $P_t(1 \leq t \leq 5)$. Due to the space limitation, comparing the performance of the methods in all possible domains of the parameters is impractical. We measure the performance of the methods by allowing change in one parameter and keeping others unchanged. The values of unchanged variables are selected such that, if possible, the performance improvement near the selected value is quite stable.

5.2.2. Effect of number of query terms, signature size and placement of disk blocks. IP values for varying $V(t)$ and $E(t)$ values are plotted in Fig. 7. In general the effect of the framing on the performance of MFSF increases as the possibility of queries with various number of query terms increases, i.e., more $P_t(1 \leq t \leq t_{\max})$ cases assume non-zero probability values. For example, for $V(t)=1$ and $E(t)=1$ ($P_1=0.553$ and $P_2=0.351$ and other P_t values are negligible) the IP value for GFSSF–MFSF case is 35.37%. In the UD case all P_t values are equal to 0.2 and the IP value for the GFSSF–MFSF case increases to 70%. Since the UD cases exhibit an average performance, we will use only the UD case in the following comparisons.

Improvement percentage values for varying signature sizes are plotted in Fig. 8. Large databases ($N \geq 10^6$), with signature sizes less than 800 bits, corresponding to a space overhead less than 20%, produce many false drops and the response time is relatively high. Therefore, for practical purposes, we consider F values greater than 800 for such large databases. For $F > 800$, the IP value varies between 11% and 12.7% for the P-BSSF–MFSF case and between 65% and 70% for the GFSSF–MFSF case. This shows that, except for small F values ($F < 800$), the performance improvement is quite invariant to changing F values.

To inspect the effect of SP on the IP values for changing F values we included the extreme cases for SP in Fig. 8. Except for small F values, the same relative performances were obtained for all SP values.

We want to revisit the effect of the number of query terms on performance one more time. As shown in Fig. 7, the effect of multi-framing a signature file increases if the possibility of queries

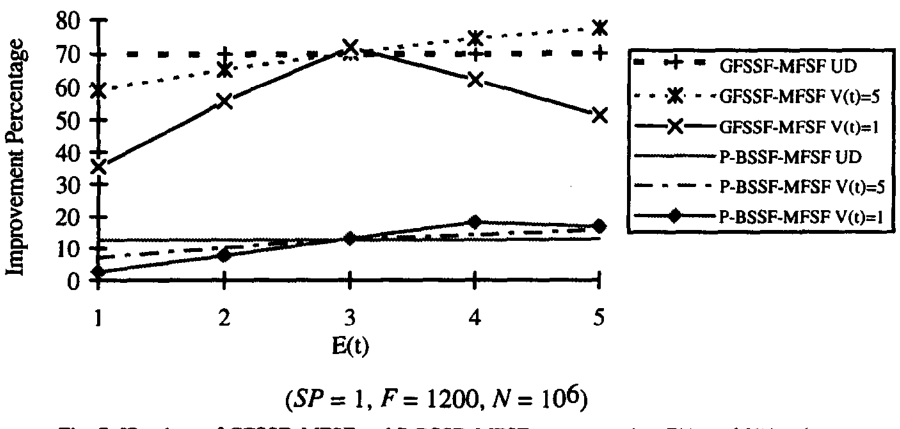
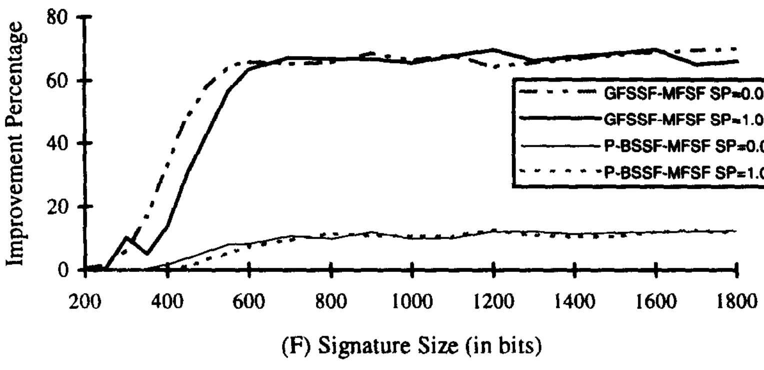


Fig. 7. IP values of GFSSF–MFSF and P-BSSF–MFSF versus varying $E(t)$- and $V(t)$-values.



($N = 10^6$, UD Query Case)

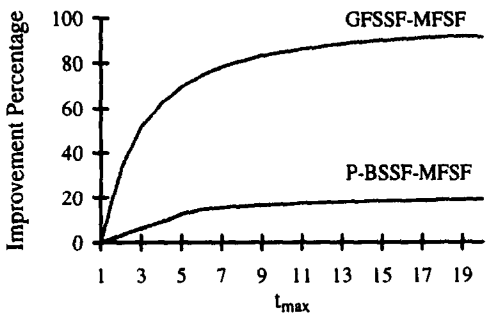
Fig. 8. IP values of GFSSF–MFSF and P-BSSF–MFSF versus varying F values.

with a different number of query terms increases. The maximum number of query terms is limited by t_{max} in our optimization model. We plot the IP values for increasing t_{max} values in Fig. 9. In this figure high t_{max} values are included to provide a theoretical comparison and may be hard to observe in practical settings. However, such cases are becoming more realistic due to multi-media applications (Zezula *et al.*, 1991). For $t_{max} = 1$, i.e., when there are only single term queries, all methods obtain the same response time and IP values are zero. For increasing t_{max} values, the number of queries with a different number of query terms increases. This increases the performance of MFSF over P-BSSF and GFSSF. Note that t_{max} value used in other comparisons ($t_{max} = 5$) is below the saturation point ($t_{max} = 10$) where IP values of P-BSSF–MFSF and GFSSF–MFSF cases are 16.9% and 84.78%, respectively.

5.2.3. Effect of database size. The performance improvement values for changing N values are plotted in Fig. 10. For N values near 2000, IP values of P-BSSF–MFSF reach 10% and vary between 11% and 12.7% for increasing N values. IP values of GFSSF–MFSF rise to 65% for $N = 30{,}000$ and vary between 65% and 70% for increasing N values. Therefore, accept for very small N values ($N < 2000$), MFSF performs better than GFSSF.

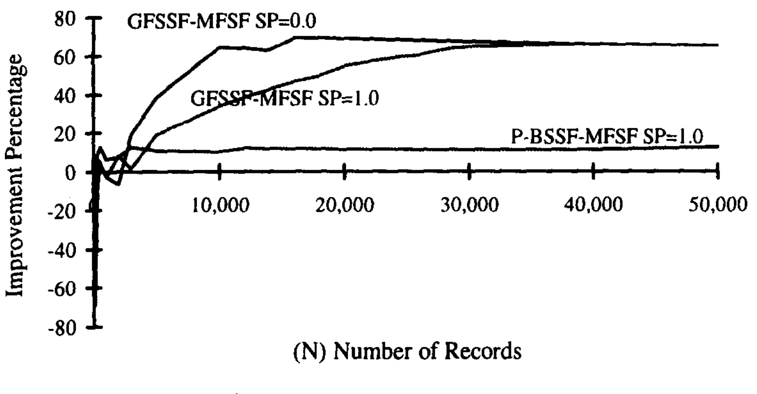
In the P-BSSF–MFSF case, for $N \leq 65{,}536$, a bit slice fits in a disk block, the same IP values are obtained for changing SP values. For larger N values negligible variations in IP values are observed for changing SP values. In the GFSSF–MFSF case, smaller SP values cause IP values to increase more rapidly for increasing N values. The reason of such a performance decrease for GFSSF is that the effect of reducing seek operations decreases for lower SP values. As a result, except for very small database sizes ($N < 30{,}000$), the performance improvement of MFSF over P-BSSF and GFSSF is invariant to the changes in N and SP.

The simulation runs show that, excluding very small databases and signature sizes, MFSF always outperforms GFSSF and P-BSSF in all parameter domains. For small N values, the difference between the response times of the methods becomes negligible.



($SP = 1$, $F = 1200$, $N = 10^6$, UD Query Case)

Fig. 9. IP values of GFSSF–MFSF and P-BSSF–MFSF versus varying t_{max} values.



(F = 1200, UD Query Case).

Fig. 10. IP values of GFSSF–MFSF and P-BSSF–MFSF for varying N values.

There are two important findings of this analysis which verify our intuitive expectations: (i) the response time of MFSF decreases for an increasing number of query terms, (ii) the performance of MFSF increases for an increasing number of queries with a different number of terms (i.e., with more non-zero P_t values).

6. REAL DATA EXPERIMENTS

We tested the response time of MFSF by using the same test data used to obtain the database statistics provided in Table 3. The purpose of these experiments are to observe the actual behavior of the method, see the agreement between theory and practice, and make projections for large databases. The test database, BLISS-1, contains 152,850 MARC records and the size of the data file is 91 Mbyte. In BLISS-1 average record length is 613 bytes and on the average each record contains 25.7 distinct terms. The 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 10%.

6.1. Determining the query signature on-bits used in the query processing

If the query signature on-bits used in the query processing are selected randomly, for the queries with high numbers of query terms, the partial evaluation strategy may ignore some query terms by using none of their on-bits. To make sure that each query term contributes to the query evaluation, we selected the on-bits used in the query evaluation in a round-robin (RR) approach, i.e., the first on-bit is selected from the first query term, the second on-bit is selected from the second term, and so on. In MFSF, generally, each term sets only one bit in the lower numbered frames and the on-bits of a lower numbered frame are used first. Therefore, in MFSF, the RR method and random selection of the query signature on-bits for query evaluation produce similar results.

For small N and high t values, which is unlikely in real life, the stopping condition may require using less number of bit slices than the number of query terms. For such cases, to guarantee the contribution of each query term to the query evaluation, using additional bit slices may be required after the stopping condition is reached. However, since the size of bit slices will be small for small N values the increase in the response time will be small.

Although the observed and estimated average *op* values of the bit slices of MFSF agree, we observed higher *op* values than the estimated value at the bit positions where a high frequency term (a term occurring in many records) sets bits. When possible, to prevent using bit slices with

Table 5. 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

high *op* value in the query evaluation, we sorted the on-bits of a query term in non-decreasing *op* value. The RR bit selection method uses on-bits of each query term in this order. Sometimes, this may ensue using an on-bit from a higher numbered frame before using the on-bits of the same term in the lower numbered frames. Since this policy may prevent using the bit slices with high *op* values, the number of observed false drops and the response time decreases.

6.2. *Query generation and performance measurement*

To test the performance of MFSF, three different query cases are considered: Uniform Distribution (UD), Low Weight (LW), and High Weight (HW) queries. P_t ($1 \leq t \leq 5$) values for these distributions are given in Table 5.

We generated a query set containing 1000 zero hit queries randomly by considering the occurrence probabilities of the number of query terms for each query case. For example, since the occurrence probability of a one term query is 0.1 in the HW query case, the HW query set contains 100 ($0.1 \cdot 1000$) one term queries. The observed false drops and response time values are obtained by taking the average of the false drops and response times obtained by each query in the query sets. Since there is no relevant record to zero hit queries, all false drops must be eliminated to find the first relevant record. This coincides with the analytical response time definition used to test the performance of MFSF with simulation runs.

6.3. *Results for false drops and query processing time*

The expected (denoted by Exp) and the observed (denoted by Obs) average false drop values of MFSF for the query cases are given in Table 6. The expected and observed response times of the query cases are plotted in Fig. 11. In the experiments of this study the term signature generation algorithm used in Kocberber & Can (1995b) for P-BSSF is replaced with a more versatile one that avoids generating the same signature for different terms and is provided in Appendix C of Kocberber (1996a). Since the expected response times of the query cases are very close, we give only the HW case which provides the lowest response times.

The observed average false drop values, hence the observed response time values, are greater than the expected values. For increasing F values the expected and observed false drop values come closer. For $F \leq 800$, we obtain too many false drops. Consequently, the response time is very high and using a signature file of this size is impractical. This is consistent with our simulation results.

Table 6. Expected and observed average false drop values for the query cases LW, UD, and HW

	LW		UD		HW	
F	Exp	Obs	Exp	Obs	Exp	Obs
1000	0.60	5.19	0.64	2.73	0.47	2.81
1200	0.54	2.60	0.43	2.08	0.43	2.24
1400	0.39	1.66	0.37	1.66	0.41	1.29
1600	0.46	1.55	0.40	1.12	0.31	0.84
1800	0.39	1.28	0.33	0.81	0.24	0.47

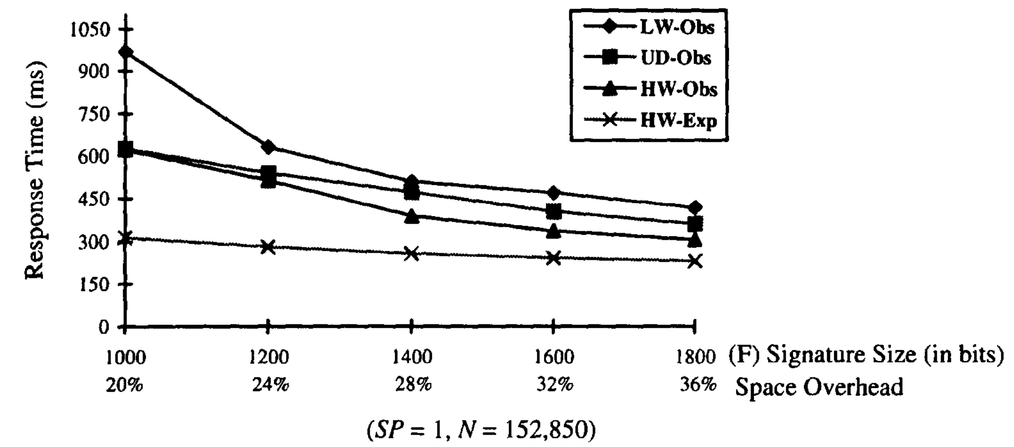


Fig. 11. Expected and observed response time versus *F*.

A small fraction of the records in the test database are very large; for example, there are 584 MARC records containing more than 75 terms which constitute 0.38% of the test database and the largest record contains 166 unique terms. These large records increase the observed response time by increasing the number of false drops. We tested the effect of these records on the response time by removing them from the test database. The signature file parameters for the reduced databases are optimized by using new average number of unique terms and taking $F=1200$. The results for the UD query case are given in Table 7. The percentage deviation from the expected response time is computed as

Deviation of TR $=100\cdot(TR_{\text{Observed}}-TR_{\text{Expected}})/TR_{\text{Expected}}$, and *Percentage Deviation of FD* is computed similarly.

The difference between the expected and observed response time values decreases dramatically as the maximum number of unique terms in the records (second column of Table 7) decreases. Since these large records constitute a small fraction of the database, they can be stored in a separate file and searched separately to provide a faster response time.

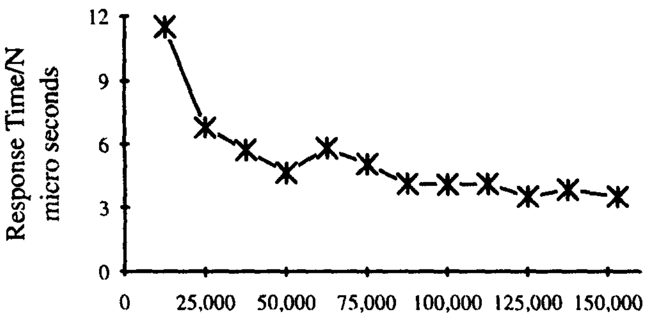
6.4. Scaleability of MFSF: projection for large databases

A desirable indexing method must be scaleable, i.e., adaptable for large databases. To test the change in the observed response time for increasing database sizes (*N* value), we performed a series of experiments in the UD query environment. The results of the experiments are plotted in Fig. 12. The test databases 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 $F=1200$.

To keep the number of false drops, *FD*, the same for increasing *N* values the number of bit slices used in the first phase of a query evaluation increases. This also increases the response time. However, each additional bit slice causes an exponential increase in the upper bound for the *N* value which produces at most the same number of false drops. For example, to obtain

Table 7. Results of limiting maximum number of terms in the records ($F=1200$, $SP=1$)

				Expected		Observed		Deviation (%)	
N	Max *D*	Avg. *D*	Standard Deviation of *D*	*FD*	*TR*(ms)	*FD*	*TR*(ms)	*FD*	*TR*
152,850	166	25.70	11.12	0.43	303	2.08	541	384	79
152,686	100	25.60	10.72	0.42	302	1.46	440	248	46
152,266	75	25.44	10.24	0.40	301	1.34	402	235	34
149,408	50	24.82	9.26	0.35	296	0.93	342	166	16
140,901	40	23.64	8.12	0.47	284	0.86	308	83	8



Number of Records (N)
($SP = 1$, $F = 1200$, UD).

Fig. 12. Response time per record versus N.

$FD=0.1$ with $op=0.01$ one bit slice is sufficient up to $N=10$ (this means 0.1 bit slice processing for each record), two bit slices are sufficient up to $N=100$ (this means 0.02 bit slice processing for each record), and so on (see equation (2)). Therefore, during query–document signature matching the time spend for each record of the database decreases for increasing N value.

To retrieve the bit slices of MFSF with minimum disk block accesses, the bit slices were aligned according to the disk block boundaries. In the experiments, a disk block is sufficient to store a bit slice up to $N=65{,}536$ $(8 \cdot 8192)$. For $65{,}537 \leq N \leq 131{,}072$, two disk blocks must be read to retrieve a bit slice. At the lower bound of this interval ($N=65{,}537$) the time spend to read the processed bit slices increases sharply due to reading additional disk blocks while N increases only by one. Consequently, the ratio of response time to the number of records (TR/N) increases for N values that cause to retrieve an additional disk block to read a bit slice. The rise in TR/N will increase for decreasing SP value since the number of seeks will also increase at these boundary N values. However, TR/N value (i.e., response time per record) will decrease for increasing N value since the number of bit slices processed for each record decreases exponentially for increasing N as illustrated by the discussion of the previous paragraph.

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 3.54 micro seconds for larger databases. Note that this value is the TR/N figure observed for the complete database, i.e., for $N=152{,}850$. By assuming $TR/N=3.54$ micro seconds, we projected the observed response time for $N=10^6$ as 3.54 seconds. Note that this is a pessimistic (worst-case) projection since the TR/N ratio (response time/record) decreases for increasing N.

7. DISCUSSION AND FUTURE WORK

IF methods and signature file methods are efficient search indices. There are theoretical (Zobel *et al.*, 1992) and experimental comparisons (Couvreur *et al.*, 1994; Kocberber & Can, 1996b) of these methods. However, the performance of IF and signature file methods in terms of efficiency depend on many parameters such as the database instance, the computer used in the experiments, disk space allocation methods, and the amount of available main memory. Due to the absence of well defined fair comparison environments the results of the comparisons become questionable. Another difficulty is that both methods have configuration parameters providing fine tuning of the performance of the methods. Signature files especially have many configuration parameters which provide adaptation of the method to various environments.

We confine the scope of this work to define the theoretical base of the MFSF method and show that it is the best vertically partitioning method among the BSSF, B'SSF, P-BSSF, and GFSSF methods. In the rest of this section, we provide a brief theoretical comparison of IF and MFSF in terms of number of disk accesses required to respond a query. Our aim is to show that MFSF opens new promising research directions rather than proving MFSF performs better than

IF. In the following discussion, we assume a posting list of IF and a bit slice of MFSF are represented with a bit string without implying the same storage structure will be used for both methods. Also, we assume that the pointers to the actual records are stored in main memory.

Our experiments show that for $F=1200$, $SP=1$, and $N=152{,}850$ depending on the number of query terms, only four or five disk accesses are needed to complete the signature file processing. Note that higher F values will require fewer disk accesses while the space overhead of MFSF increases. For BLISS-1 if only 3442 large records are processed with a different method, the observed number of false drops per query was reduced to 0.93 (see Table 7). Only five or six disk accesses are sufficient to respond a query independent of the number of relevant records to the query. Since IF requires two disk accesses for each query term, for $F=1200$, $SP=1$, and $N=152{,}850$, MFSF will require fewer disk accesses than IF for queries containing three or more terms.

As we stated before, usually, each term sets only one bit in the lower numbered frames of MFSF. Increasing the F value reduces *op* values of these frames. Lower *op* values provide reaching the stopping condition in fewer evaluation steps. For example, for $F=10{,}000$ and $N=152{,}850$, accessing only three bit slices will be sufficient to complete the signature file processing for the queries containing up to three terms. If the bit-slices of MFSF are compressed, increasing F will increase the space overhead slightly since the bit slices will be more sparse.

In our future research we will inspect increasing F value and storing the bit slices in a compressed form. If F value is increased until most of the compressed bit slices of a MFSF fit a disk block, no additional seek operation will be required for any SP value and MFSF can obtain the same performance in multi-user environments where SP can be considered as zero. Our initial experiments regarding this provide promising results (Kocberber & Can, 1996b).

To reduce the number of observed false drops to a negligible value, in our future research, we will incorporate the variation in the number of record terms in the signature file optimization process instead of using average D value, as used in Grandi *et al.* (1992); Kocberber & Can (1995b); Kocberber & Can (1996b); Lin & Faloutsos (1988); Lin & Faloutsos (1992); Roberts (1979) and Sacks-Davis *et al.* (1985). Using individual D values of the records in the signature file optimization process will decrease the gap between the expected and observed response time values by decreasing the observed response time while increasing the expected response time (due to more precise computation). To obtain better performance in the databases with non uniform record term frequencies, the database will be divided into sub-files by distributing the records among the sub-files according to the length of the records. Then, each sub-file will be searched by using the best method for the sub-file such as SSF, GFSSF, and MFSF. Another research topic will be the adaptation of MFSF to parallel processing environments and its performance evaluation.

8. CONCLUSION

A new signature file method, MFSF, is presented. The new method improves the performance in a multi-term query environment by dividing the signature file into variable sized vertical frames with different *op*. A partial evaluation strategy that dynamically avoids the complete use of the on-bits of query signatures is used which can be employed in other vertical partitioning methods. The analysis shows that MFSF provides significant (up to 85%) performance improvement over GFSSF in a multi-term query environment.

The performance of MFSF is also measured with a prototype information retrieval system with a database of 152,850 MARC records and using a disk drive with 30 ms seek time. By using the results obtained for the test database, we projected the worst-case response time for a database with 10^6 records as 3.54 seconds with a 24% space overhead in a uniform distribution multi-term query environment with 1–5 terms per query. This is a very promising result.

Unlike a recent work on vertical partitioning (Lin & Faloutsos, 1992p. 285), in MFSF the response time decreases with an increasing number of query terms. This is due to our framing and partial query evaluation strategy that reduces the expected number of false drops to an acceptable value in fewer bit slice evaluations for increasing numbers of query terms. Furthermore, the response time of MFSF is independent of the number of hits to the queries. Since the stopping step is determined dynamically according to the number of records in the database during query evaluation, the need for MFSF reorganization for growing database size will be infrequent. These are desirable characteristics.

We leave comparing MFSF with other retrieval methods such as IF (Zobel *et al.*, 1992), linear hashing with superimposed signatures (Zezula *et al.*, 1991), multi-level signature file methods (Sacks-Davis *et al.*, 1985), multi-organizational scheme (Kent *et al.*, 1990), and multi-level superimposed coding method (Lee *et al.*, 1995) as future work. We believe that, due to decreasing response time for an increasing number of query terms, MFSF will perform better than these methods for some application domains where queries with many terms are usual such as multi-media databases (Zezula *et al.*, 1991).

Acknowledgements—We are grateful to a referee for insightful comments that improved the presentation.

REFERENCES

Aktug, D., & Can, F. (1997). Signature files: An integrated access method for formatted and unformatted databases, submitted to *ACM Comp. Surveys* (under revision).

Bookstein, A., & Klein, S. T. (1990). Using bitmaps for medium sized information retrieval systems. *Information Processing & Management*, *26*(4), 525–533.

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

Couvreur, T. R., Bezel, R. N., Miller, S. F., Zeitler, D. N., Lee, D. L., Singhal, M., Shivaratri, N., & Wong, W. Y. P. (1994). An analysis of performance and cost factors in searching large text databases using parallel search systems. *Journal of American Society for Information Science*, *45*(7), 443–464.

Faloutsos, C. (1992). Signature files. In W.B. Frakes and R. Baeza-Yates (Eds) *Information retrieval data structures and algorithms*, (pp. 44-65). Englewood Cliffs, NJ: Prentice Hall.

Grandi, F., Tiberio, P., & Zezula, P. (1992). Frame-sliced partitioned parallel signature files. In *Proceedings of the 15th International ACM-SIGIR Conference*, Copenhagen, Denmark, pp. 286–297.

Kent, A., Sacks-Davis, R., & Ramamohanarao, K. (1990). A signature file scheme based on multiple organizations for indexing very large text databases. *Journal of the American Society for Information Science*, *41*(7), 508–534.

Kocberber, S., & Can, F. (1995a). Optimization of bit-sliced signature files in multi-term query environments, In *Proceedings of the 10th International Symposium on Computer and Information Sciences*, Ephesus, Turkey, pp. 161–168.

Kocberber, S., & Can, F. (1995b). Partial evaluation of queries for bit-sliced signature files. *Information Processing Letters* (to appear).

Kocberber, S., & Can, F. (1995c). Generalized vertical partitioning of signature files. Technical Report BU-CEIS-9513, Department of Computer Engineering and Information Science, Bilkent University, Ankara, Turkey (http:/ /www.cs.bilkent.edu.tr/tech-reports/1995/BU-CEIS-9513.ps.z).

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

Kocberber, S., & Can, F. (1996b). Fast information retrieval using compressed multi-fragmented signature files Technical Report BU-CEIS-9625, Department of Computer Engineering and Information Science, Bilkent University, Ankara, Turkey (http://www.cs.bilkent.edu.tr/tech-reports/1996/BU-CEIS-9625.ps.z).

Lee, D. L. (1986). A word parallel, bit-serial processor for superimposed coding. In *Proceedings of the 2nd International Conference on Data Engineering*, Los Angeles, California, pp. 352–359.

Lee, D. L., Kim, Y. M., & Patel, G. (1995). Efficient signature file methods for text retrieval. *IEEE Transactions on Knowledge and Data Engineering*, *7*(3), 423–435.

Lee, D. L., & Leng, C. W. (1989). Partitioned signature files: Design issues and performance evaluation. *ACM Transactions on Information Systems*, *7*(2), 158–180.

Lin, Z. &, Faloutsos, C. (1988). Frame-sliced signature files. Technical Report CS2146 and UMIACS-TR-88-88, Computer Science Department, University of Maryland.

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

Pogue, C. A., & Willett, P. (1987). Use of text signatures for document retrieval in a highly parallel environment. *Parallel Computing*, *10*, 259–268.

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

Sacks-Davis, R., Kent, A., & Ramamohanarao, K. (1985). Performance of multikey access method based on descriptors superimposed coding techniques. *Information Systems*, *10*(4), 391–403.

Salton, G. (1989). *Automatic text processing: The transformation analysis, and retrieval of information by computer.* Reading, MA: Addison-Wesley.

Zezula, P., Rabitti, F., & Tiberio, P. (1991). Dynamic partitioning of signature files. *ACM Transaction on Information Systems*, 9(4), 336–367.

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