COMPRESSED MULTI-FRAMED SIGNATURE FILES: AN INDEX STRUCTURE FOR FAST INFORMATION RETRIEVAL

Seyit Koçberber
Department of Computer Engineering and Information Science, Bilkent University, Bilkent, Ankara 06533, Turkey, seyit@bilkent.edu.tr

Fazlı Can
Department of System Analysis, Miami University, Oxford, OH 45056, U.S.A., canf@muohio.edu

Keywords: Signature Files, Inverted Files, Compression

ABSTRACT

A new indexing method, called Compressed Multi-Framed Signature File (C-MFSF), that uses a partial query evaluation strategy with compressed signature bit slices is presented. In C-MFSF, a signature file is divided into variable sized compressed vertical frames with different on-bit densities to optimize the response time. Experiments with a real database of 152,850 records show that a response time less than 150 milliseconds is possible. For multi-term queries C-MFSF obtains the query results with fewer disk accesses than the inverted files. The method requires no indexing vocabulary. These attributes have important implications; for example, web search engines process multi-term queries in very large databases with sizeable vocabularies.

1. INTRODUCTION

Signature file approach is a well-known indexing technique for information access. In signature files, the content of a record (an instance of any kind of data will be referred to as a *record*) is encoded in a bit string called *record signature*. In (superimposed) signatures each term (an attribute of a record, without loss of generality, will be referred to as a *term*) is hashed into a bit string of size F by setting S bits to "1" (*on-bit*) where F >> S. The result is called a *term signature*. Record signatures are obtained by superimposing (i.e., bit wise ORing) the record term signatures [1, 2, 3]. In this paper we consider superimposed signatures and conjunctive queries. Query signatures are obtained by superimposing the query term signatures.

In this study, we propose the Compressed Multi-Framed Signature File (C-MFSF) method that stores the sparse bit slices of MFSF [8] with large F values in a compressed form. C-MFSF can be used in the implementation of various types of Information Retrieval (IR) systems such as text and multimedia systems, on-line library catalogs, set accesses in object-oriented databases, on-line help systems, etc. [4, 12].

SAC '99, San Antonio, Texas

The method obtains the multi-term query results with fewer disk accesses than the inverted file approach. The contribution of this study is that for very large databases, queries containing more than two terms can be evaluated by one disk access per query term without storing and searching a vocabulary. This has important implications; for example, web search engines process multi-term queries in very large databases with enormous vocabularies.

2. MULTI-FRAMED SIGNATURE FILE (MFSF)

The query evaluation with signature files is conducted in two phases. In the *first phase*, the query signature is compared with the record signatures. The records whose signatures contain at least one "0" bit (off-bit) in the corresponding positions of on-bits of the query signature do not contain all query terms. If a record contains all of the query terms (such records will be referred to as *matching* records), its signature will have on-bits in the corresponding bit positions of all on-bits of the query signature. Due to hashing and superimposition operations used in obtaining signatures, the signature of some non-matching records may coincide with the query signature. These records are called *false drops*. In the *second phase* of the query processing, false drop records (if any) are eliminated by accessing the actual records.

For a database of N records, the signature file can be viewed as an N by F bit matrix. Signature file processing can be done by considering only the columns (bit slices) corresponding to the on-bits of the query signature [9, 10].

In BSSF (bit-sliced signature files), the time required to complete the first phase of the query evaluation increases as the number of on-bits of the query signature, i.e., query weight, increases [10]. MFSF solves this problem by employing a partial evaluation strategy and considering the submission probabilities of queries with different number of terms in multi-term query environments [6, 8]. Our query evaluation technique employs a stopping condition that tries to complete the first phase of the query evaluation without using all on-bits of the query signature, i.e., by *partial evaluation* [7]. This approach stops bit-slice processing and switches to the false drop elimination when the expected cost of false drop elimination is less than that of the bit slice processing.

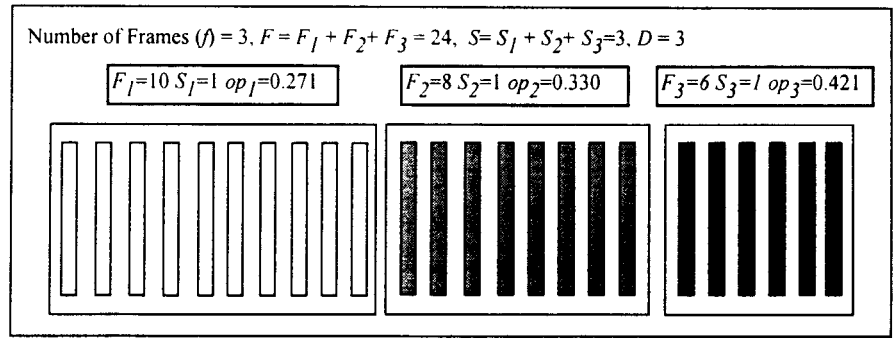
In MFSF a signature file is conceptually divided into f sub-signature files. The bits of a signature file are distributed among the sub-signature files, frames, such that $F = F_1 + F_2$

... + F_f ($f \leq F$). Each term sets S_r bits in the rth frame such that $S = S_1 + S_2 \ldots + S_f$ ($1 \leq S_r \leq F_r$, $1 \leq r \leq f$). Each sub-signature file is a BSSF with its own F (signature size) and S (number of bits set by each term) parameters.

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

The use of a very large F_1 value would eliminate the need for the second and the following frames. However, this would increase the file size to unrealistic amounts even after compression. In our case F_1 is kept "relatively large." For queries with small number of terms the first frame will eliminate insufficient number of false drops. The additional frames are provided for further false drop elimination and they are mainly for one and two term queries.

Reducing on-bit density while providing sufficient on-bits in query signatures is possible by increasing the signature size F. However, increasing F also increases the space overhead if the bit slices are stored without compression. The Compressed Multi-Framed Signature File (C-MFSF) method stores the bit slices of MFSF in a compressed form. Because of space limitation the details of compression are skipped ([6, 12]).



t	Number of On-Bits in Each Query Frame:		
1	1	1	1
2	2	2	2
3	3	3	2*

t: number of query terms, D: number of distinct terms in a record

Different gray levels indicate different on-bit densities (op values) of the frames, $op_i = 1 - (1 - S_i / F_i)^D$

(*) More than one term may set the same bit position

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

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

* In this formula S_i/F_i indicates the probability that a (random) record signature bit is set by a record term, $(1-S_i/F_i)$ indicates the probability that a bit is not set to 1 by a record term. Therefore, $(1-S_i/F_i)^D$ is the probability that a bit is not set to 1 by any of D record terms, then $(1-(1-S_i/F_i)^D)$ indicates the probability that a signature bit is set to 1 by the record terms.

3. TEST APPLICATION ENVIRONMENT

To estimate the performance of C-MFSF a simulation and test environment is designed. The values of the parameters used (see Table I) in the simulation runs were determined experimentally in a PC environment. By this way we can validate our simulation using real data experiments. A validated index model can be used to obtain the optimum index structure (in our case C-MFSF) by employing new system parameters.

We used MARC (MAchine Readable Cataloging) records of the Bilkent University library collection as the test database. The database, BLISS-1, contains (N) 152,850 records and defined by (V) 166,216 unique terms. The MARC database size is 93.24 MB.

To measure the performance of C-MFSF we considered three different query cases: Low Weight (LW), Uniform Distribution (UD), and High Weight (HW) queries. (The weight of a signature means the number of 1s in the

signature; therefore, a LW query contains least number of 1s among all query types.) The values of P_t $(1 \le t \le 5)$ where P_t denotes the probability of submitting a t term query, for these query cases are given in Table II.

Table I. System Parameter Values of the Application Environment

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

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

Query Case	P1	P2	P3	P4	P5
Low Weight (LW)	0.30	0.25	0.20	0.15	0.10
UniformDistribution (UD)	0.20	0.20	0.20	0.20	0.20
High Weight (HW)	0.10	0.15	0.20	0.25	0.30

For each query case, we generated a query set containing 500 queries by considering the occurrence probabilities of the number of query terms. For example, the HW query set contains 50 (0.10·500) one term queries. In our experiments we also consider the execution time of queries with a specific number of terms and used five additional query sets: T1, ..., T5. The first query set, T1, contains 500 single term queries, the second query set, T2, contains 500 two term queries, and so on.

$$RT(i) = \sum_{s=1}^{i} T_{slice-s} + FD_i \cdot T_{resolve} \quad (1)$$

where $0 < i \le W(Q)_t$

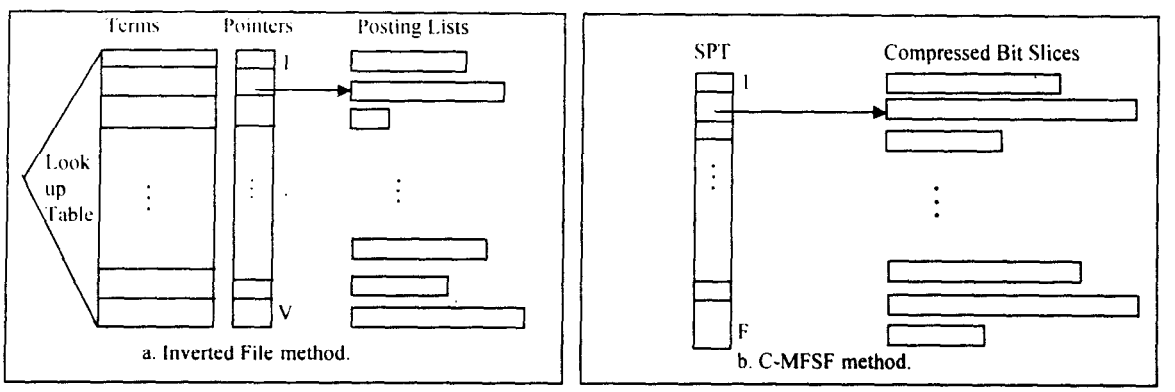
where $T_{slice\text{-}s}$ is the time required to process the sth bit slice (which involves decompression) used in the query evaluation, FD_i is the expected number of false drops after processing i bit slices, $T_{resolve}$ is the time required to resolve a false drop, t is the number of query terms, and $W(Q)_t$ is the number of on-bits in the query signature. Our response time definition ignores the time needed to access the matching records as in other studies (for explanation see [8, 9]).

The number of evaluation steps, i, and the expected number of false drops after processing i bit slices, FD_i, are determined as in [8]. To provide the contribution of each query term to the query evaluation we use at least one on-bit from each term. The C-MFSF structure is optimized with the heuristic search algorithm given in [8].

In C-MFSF each frame may have a different *op* value and hence the number of on-bits in the bit slices of C-MFSF and the length of the compressed bit slices vary. To obtain the addresses of the compressed bit slices a Slice Pointer Table (SPT) with F entries is used. SPT is kept in memory and to retrieve a bit slice, first the address of the bit slice is obtained from SPT. To illustrate the difference between C-MFSF and the inverted file method the storage structures of these methods are shown in Figure 2. Compression can also be used in posting lists of inverted files [12, 13].

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

$$T_{slice-i} = Read(T_{nearseek}, sl_i) + T_{byteop} \cdot \lceil \text{compressed bit slice size in bits} \rceil \quad (2)$$



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

Figure 2. Storage structures of C-MFSF and the inverted file methods.

4. SIMULATION MODEL

Like in other signature applications we use the *response time* as the performance measure [9]. It involves the time required to process the signature file and resolve all false drop records. The response time after processing i bit slices, $RT(i)$, is estimated as follows.

where T_{byteop} is the time required to process a byte and sl_i is the average number of disk blocks required to store a slice of the ith frame and the compressed bit slice size can be estimated using on-bit density information [6, 12].

Read(T_{seek}, b) incorporates the sequentiality probability, *SP*, to the estimation of the time required to read a bit slice involving *b* disk blocks. *SP* is the probability of reading a disk block without a seek operation.

$$\text{Read}(T_{seek}, b) = (1 + (b-1)\cdot(1-SP))\cdot T_{seek} + b\cdot T_{read} \qquad (3)$$

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

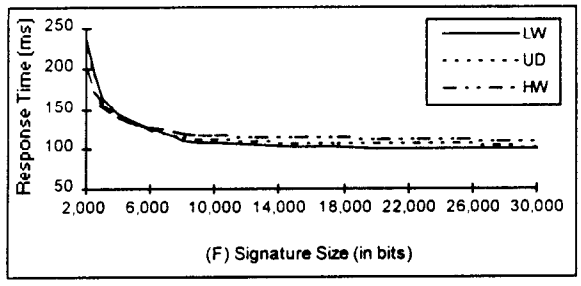
The false drop resolution time for one record, $T_{resolve}$, is computed as follows.

$$T_{resolve} = (1 - \tfrac{PB}{N})\cdot \text{Read}(T_{farseek}, \left\lceil \tfrac{PB\cdot P_{size}}{B_{size}} \right\rceil) + \text{Read}(T_{farseek}, RB) + T_{scan} \qquad (4)$$

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

5. SIMULATION EXPERIMENTS

We plot the expected response time values of C-MFSF for increasing *F* values in Figure 3.



(SP = 1.0, N = 152,850)

Figure 3. Expected response time versus very large *F* values for C-MFSF for LW, UD, HW.

Increasing *F* values provides lower on-bit densities and the stopping condition is reached in fewer slice evaluations. Therefore, the optimization algorithm of C-MFSF selects smaller *S* values for increasing signature size. This also decreases the response time. After a certain F value the increase in F has no effect on the response time.

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

To simulate the effect of large records we gradually increased the D_{avg} (average number of distinct terms in a record) values in a new set of simulation experiments. For increasing D_{avg} values we search the *F* value that requires *S* = 3 which gives the best results in the experiments with the test database BLISS-1 (for efficiency, *F* values are increased in steps of 50). The minimum *F* values with the expected FD and RT (expected total response time in multi-term query environments, in millisec) values are given in Table III.

Table III. Minimum F Values that Provide S = 3 for Increasing D_{avg} Values and Compression Performance

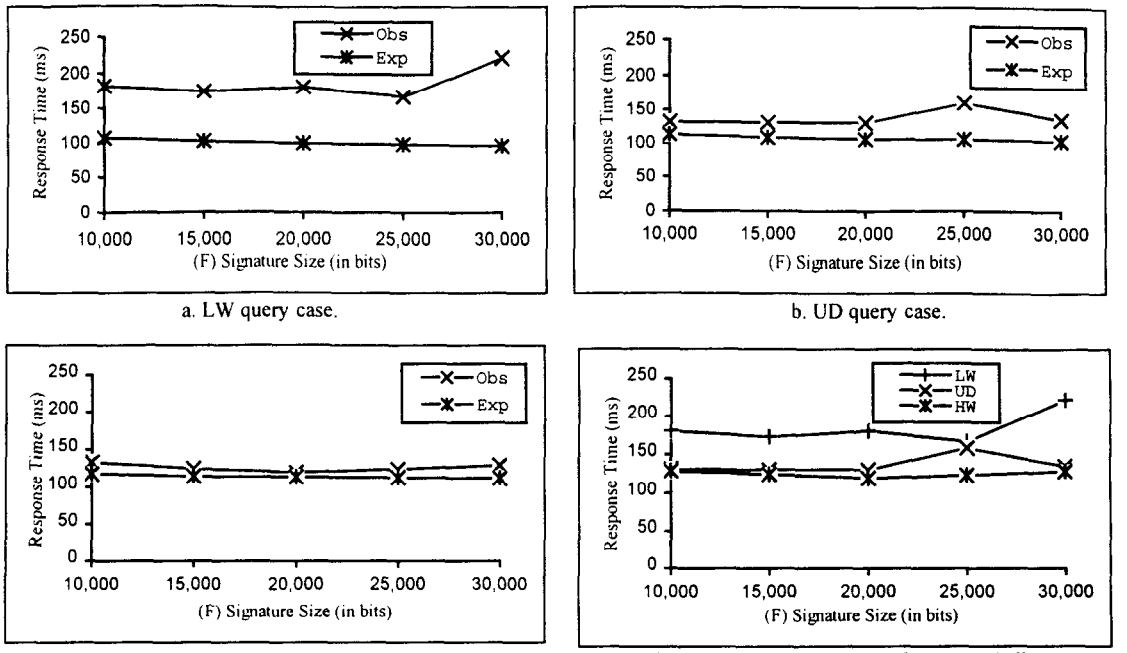
Davg	Minimum F	Expected FD	Expected RT
25.7	6150	0.272	125
50.7	10750	0.271	126
100.7	20650	0.269	126
150.7	30700	0.269	126
200.7	40600	0.272	126

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

6. REAL DATA EXPERIMENTS

The simulation experiments (Figure 3) show that a response time less than 150 milliseconds is possible if large *F* values are used. We tested the optimized C-MFSF configurations with BLISS-1 and validated the results of the simulation model. The expected (denoted by Exp) and the observed (denoted by Obs) response time values are plotted in Figure 4 (for easy comparison the observed response time values for LW, UD, and HW repeated in Figure 4.d). In the experiments most of the processed bit slices and MARC records (used for false drop elimination) fit into a disk block and therefore SP= 1.0.

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



a. LW query case.

b. UD query case.

c. HW query case.

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

Figure 4. Expected and observed response time of C-MFSF versus F for LW, UD and HW (SP = 1)

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

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

The table shows that the queries with more than two terms (t > 2) generate almost no false drops and the query evaluation is completed by accessing only the signature file without any actual record accesses for false drop resolution. Furthermore observed and expected response times are closer to each other. Therefore, we conclude that the difference between the expected and the observed values are especially due to single term queries. Single term queries have only three on-bits in their query signature and if one of them shares the same bit slice with a high frequency term, more false drops are produced than the expected number. The number of disk accesses is almost the same as the number of query terms for queries with more than two terms.

7. COMPARISON OF C-MFSF AND INVERTED FILE

The number of disk accesses for index performance evaluation is a commonly accepted measure [11, pp. 14 - 15]. In the following discussion, for the C-MFSF and inverted file (IF) methods we assume that disk addresses of the records are kept in main memory. In the IF method we assume that one disk access is required per query term to read the posting list of the term. (We ignore chained posting lists and the method used for posting list representation.) In IF, to obtain the locations of the posting lists, a term lookup table is needed. 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. Therefore, in IF, a t term query will require $2 \cdot t$ disk accesses.

In C-MFSF no lookup table is needed (terms are directly used in signature generation). For F = 30,000, simulation experiments show that reaching the stopping condition requires processing only three bit slices even for very large databases ($N \geq 10^6$). For single term queries C-MFSF requires three disk accesses plus false drop resolution. Therefore, IF outperforms C-MFSF for single term queries. However, note that single term queries are less common in today's databases [5] since they produce excessive number of hits. Both methods have similar performance for queries with two terms. IF will require one more disk access but C-MFSF may produce false drops for t = 2. However, the average number of false drops requires less than one disk access (see Table IV). Therefore, the expected performance of C-MFSF is better than IF for t = 2.

For t > 2, since the contribution of each query term to the query evaluation is provided, C-MFSF processes t bit slices for a t term query. Experiments with BLISS-1 show that almost no false drop is obtained for queries with more than two terms (see Table IV). Therefore, we can assume that for F = 30,000, C-MFSF will require only t disk accesses for

queries with t > 2, i.e., one disk access for each query term contrary to two disk access per query term requirement of IF.

For multi-term queries IF may process terms according to their document frequency (from least frequent to most frequent) and may switch to false drop resolution after processing a certain number of terms [13]. However, this approach implies at least t number of disk accesses just to obtain the document frequency information of the query terms.

The performance of IF can be improved if the lookup table and document frequency information are kept in main memory [13]. In this case, still one disk access for each query term is required to read the posting list of the query term. However, this can be avoided by switching to false drop resolution as suggested above. If such a large memory is available, we can store the compressed form of a C-MFSF frame (or a part of it) in main memory. For example, a frame of C-MFSF for BLISS-1 with op = 0.011 (S and F values of the frame are 1 and 2400, respectively) requires 3.82 MBytes with "no compression." Furthermore, in C-MFSF the value of *op* (on-bit density) can be adjusted to fit the frame to the available memory [6]. Since the bit slices with many on-bits (i.e., the frames other than the first frame) are rarely used in query evaluation; therefore, we can keep the compressed bit slices of the first frame in memory. It should be stated that the time needed for decompression of one bit slice is much shorter than the time needed for one disk I/O.

Since we store one frame in memory, for single term queries one of the bit slices will be in memory. Two disk accesses will be needed to retrieve the bit slices of the other frames (usually only the second frame) to complete the first phase of the query processing. Similarly, for the queries with two terms since two bit slices will be in memory only one disk access will be needed to complete the first phase of the query processing. For the queries containing more than two terms, one bit slice for each query term will be available in memory and therefore no disk accesses will be required.

8. CONCLUSION

A new indexing method, called Compressed Multi-Framed Signature File (C-MFSF), that uses a partial query evaluation strategy with compressed signature bit slices is presented. In C-MFSF, a signature file is divided into variable sized compressed vertical frames with different on-bit densities to optimize the response time. A query processing simulation model is introduced. The experiments with a real database of 152,850 records show that a response time less than 150 milliseconds is possible and the method is readily adaptable to large databases. For multi-term queries C-MFSF obtains the query results with fewer disk accesses than the inverted file approach. The performance of C-MFSF depends on the on-bit density of the signature file and it decreases the on-bit density by increasing signature size (F) with a limited space overhead. For the databases with large records, we show that the same performance can be obtained by increasing the signature size. Since larger records occupy more disk space, the relative space overhead of C-MFSF will be approximately the same.

The contribution of this study is that for very large databases, queries containing more than two terms can be evaluated by accessing and processing one bit slice per query term without storing and searching a vocabulary. This has important implications; for example, web search engines process multi-term queries in very large databases with enormous vocabularies.

REFERENCES

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

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

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

[4] Ishikawa, Y., Kitagawa, H, and Ohbo, N. 1993. Evaluation of signature files as set access facilities in OODBs. In Prooceedings of the ACM SIGMOD'93 Conference (Washington, D.C., USA). 247-256.

[5] Jansen, B. J., et al. 1998. A study of user queries on the Web. *ACM SIGIR Forum*. 32, 1 (Spring). 5-17.

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

[7] Kocberber, S., Can, F. 1996. Partial evaluation of queries for bit-sliced signature files. *Information Processing Letters* 60. 305-311.

[8] Kocberber, S., Can, F. 1997. Vertical framing of superimposed signature files using partial evaluation of queries. *Information Processing & Management*. 33, 3, 353-376.

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

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

[11] Salzberg, B. 1988. *File Structures: An Analytical Approach*. Prentice Hall, N.J.

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

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