Incremental Query Evaluation for Vertically Partitioned Signature Files in Very Large Databases

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

Department of Computer Engineering and Information Science
Faculty of Engineering, Bilkent University
Bilkent, 06533 Ankara, Turkey

BU-CEIS-9427

July 11, 1994

Abstract

Signature files provide efficient retrieval of formatted and unformatted data with a small space overhead. The main idea of all signature based schemes is to reflect the essence of the data objects into bit patterns. Vertical partitioning of signature files provides fast filtering of unqualifying objects by accessing only those bits of the object signatures that are set in the query signature. For very large databases even one slice of database signatures may occupy several disk blocks. An incremental query evaluation method, called INCBIT, is proposed which dynamically avoids the unnecessary blocks of bit slices which need to be accessed by utilizing the conjunctive nature of the queries. Three methods to enhance the performance of INCBIT are introduced. In the first method, bit slices are divided into vertical fragments in which the density of on-bits vary. For high weight queries, after reducing the false drop probability to a negligible level, the remaining on-bits of the query are not used due to the incremental nature of INCBIT. Based on this observation, first using the query bits in the fragments with low on-bit density increases the performance. The second method originates from the fact that clustering the signatures of the records containing discriminating terms obtains a few on-blocks at the end of query processing. This prevents the deterioration of the performance for the majority of the non-zero hit user queries and improves the performance. In the third method, a bit-map for signature blocks is kept in memory for the discriminating terms used in signature clustering. The block bit-map of a term shows the positions of the signature blocks which contain at least one record in which the term exists. This data structure eliminates the block accesses produced due to high false drops at the beginning of query evaluation. The terms with lower database occurrence frequency are specified more frequently in the queries. Such discriminating terms constitute about 20% of all terms; therefore, the bit-map space overhead is small.

---

[1] e-mail: seyit@bilkent.edu.tr

[2] On sabbatical leave from Department of Systems Analysis, Miami University, Oxford, OH 45056, USA e-mail: sanf@miamiu.acs.muohio.edu

1. INTRODUCTION

Relational database systems, by utilizing set theoretic operations, provide a theoretical and practical data storage and retrieval environment for formatted data [Date 1990, Ullman 1988]. In these systems indexes on frequently used attributes provide efficient retrieval of desired data. Similarly, unformatted data (image, voice, text, etc.) can be stored in variable length data fields. (For simplicity, an instance of any kinds of data, i.e., data items stored in the database, will be referred to as *records* in the rest of this paper.) However, storing and searching unformatted data in tables of a relational database system is inefficient. Therefore, efficient data structures and search techniques must be developed for purely or partially unformatted database records [Aktug and Can 1993, Can 1993, Faloutsos 1988, Salton 1989, Van Rijsbergen 1979].

For search and retrieval purposes unformatted data is described by a set of descriptors (attributes) [Douglas and Stephanie 1989, Rabitti and Savino 1991, Salton 1975, Salton and Buckley 1988]. For example, a document can be described by the words used in the text. These words or terms are obtained by a manual or automatic indexing process. In the rest of the paper *term* or *word* are used interchangeably to mean a descriptor.

The users submit queries consisting of terms, which are assumed to be the exact representation of the desired information. The simplest way to answer a query is sequentially accessing all records, comparing the terms of the query and the records and finally selecting the records which satisfy the query. This is the *physical retrieval* of the information [Blair 1990]. The questions

- Are all of the records really relevant to the query ?
- Are these the only relevant records to the query ?

are related to the semantic meaning of the contents of the records and the information need of the user submitting the query. Answering these questions by logical interpretation of retrieval and relevant is the main purpose of information retrieval (IR). In this paper only the physical meaning of retrieval and relevant is our concern. Some of the retrieval methods are briefly introduced below.

Sequential search: All of the records are read and compared with the search query. Insertion and updates of records are easy and there is no space overhead. The only disadvantage of the method is that the retrieval speed is proportional to the number of records.

Inverted files: A pre-computed list of documents which contain the term are stored with each term [Salton and McGill 1983]. To access terms easily, an index structure is created on the terms. This pre-computed structure provides fast retrieval, but, to keep the pre-computed structure up to date, extra computation is required for

insertion and updates of the records. Also, the pre-computed structure requires an additional memory of 50%-300% of the original records [Haskin 1981, Faloutsos 1985a]. However, a recent study shows that by compression this can be reduced to less than 10% of the space used by the original records [Zobel et al. 1992]. This reduction can be obtained if only conjunctive queries and basic ranking are supported. If better ranking and word sequence queries are supported the index requires %25 of the space used by the actual data. Insertion of new records is complex and database creation can be expensive. Also, there is some possibility of a bottleneck during inverted file entry decoding.

Signature files: To provide a space efficient fast search structure, each term is hashed into a bit string which is called *term signature* [Aktug and Can 1993, Faloutsos 1985b]. Record signatures are generally obtained by superimposing, i.e. bitwise ORing, the term signatures occurring in the record. These record signatures are stored in a separate file, called the signature file. The size of the signature file is approximately 10% of the size of the original records [Christodoulakis and Faloutsos 1984]. Although signature files are also pre-computed structures, insertion and updates may not require as much time as inverted files.

To retrieve the relevant records of a query, first the signatures of the terms occurring in the query are superimposed to obtain a query signature, and then, this query signature is compared with the record signatures in the signature file. This provides nearly 10 times faster retrieval for sequential search due to the reduced size of the search data. Storing a signature file in column-wise order is called bit sliced storage and query evaluation. Bit sliced query evaluation method requires retrieval of the bit slices corresponding to the 1s of the query signature. Consequently, most of the bit slices are eliminated for the queries with a few bits set to '1' in their signatures [Roberts 1979]. This provides further speedup in query evaluation, while introducing extra processing time for insertion and updates.

For very large databases even one bit slice of a signature file may occupy more than 1M bytes disk space, and it can be stored in multiple disk blocks. For example, the bit sliced evaluation of a query with 30 bits set to '1' will require reading more than 30M bytes of the signature file. In a multi-user environment this will cause an I/O bottleneck, and the response time of the system become unacceptable. During query evaluation eliminating some of the blocks will provide further speedup. In this paper a new incremental query evaluation method based on this idea is proposed, and the effect on the performance is investigated. Without any additional pre-computation and space overhead, the proposed method evaluates the queries only retrieving 60% of the data required for bit sliced query evaluation. A stopping condition is defined

for partial evaluation of the queries. Also, additional structures are proposed to decrease the number of block accesses further.

In section 2 the signature file concept is introduced. Section 3 explains the proposed incremental query evaluation method. In section 4 the proposed incremental query evaluation method is integrated with the frame-sliced signature files. In section 5 future work is described. Section 6 provides a summary and conclusion.

2. SIGNATURE FILES

Signature files provide efficient retrieval of both formatted and unformatted data together with a space overhead of about 10% of the original data size [Faloutsos 1985a]. Since it is a tool for efficient retrieval in *information retrieval* (IR), there are some similarities and distinctions between signature files and inverted files. Both methods try to provide an efficient retrieval method for unformatted and formatted data with different additional space overheads. In *inverted file storage method* (IFSM), the records containing a term are stored either as a bit string or a posting list. To access to the corresponding bit string or posting list of the term, a lookup table must be maintained and searched for retrieval.

Two main disadvantages of IFSM are the space overhead and the update of the data structures of the lookup table due to the new records. The signature file concept is proposed to overcome these problems [Faloutsos 1985]. The basic signature generation and storage methods are surveyed in the following sections.

2.1 Basic Signature Generation Methods

2.1.1 Word Signature

In *word signatures* (WS) each term is hashed into a bit pattern of length k [Larson 1983, Tsichritzis 1983]. The length of the word signatures are generally the same for all terms. A record signature is obtained by concatenating the signatures of the non common words contained in the record (see Figure 1). This preserves the positional information present in the original record.

Terms	**Word Signatures**
computer	1 1 0 0
signature	0 0 1 0
extraction	1 0 1 1

d = { computer, signature, extraction }

Record signature for d : 1 1 0 0 0 0 1 0 1 0 1 1

Figure 1. Record signature generation using word signatures.

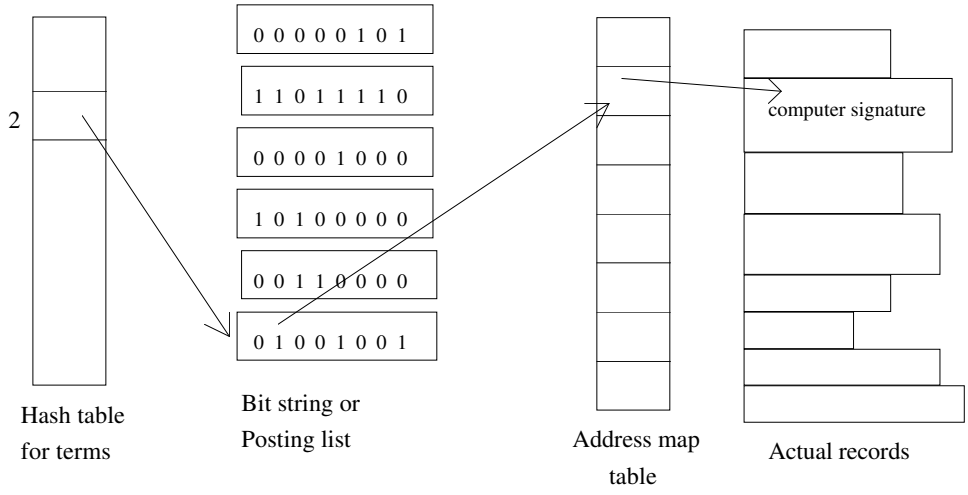
The hashed bit pattern is not an exact representation of the original term; indeed, it is an abstraction obtained from the term. The bit pattern contains not all, but some of the information present in the term. In that sense, the bit pattern is the signature of the original term.

The need for lookup table is eliminated with additional costs. The first cost may come about due to the increase in the number of bits to represent a term. A term is represented with k bits instead of one bit of the bit string representation of IFSM. On the other hand, bit strings of IFSM can store only the presence or absence of the terms, i.e., the positional information is lost. For some database instances the intended space savings may not be obtained. The second cost is incurred by the loss of certainty in the existence of terms in the records. Due to the hashing operation used to obtain word signatures, there is a chance of generating the same word signature for more than one term. Due to this uncertainty, the result of the query evaluation may produce false matches. That is, it may contain *false drops*. (The record signature satisfies the query although the actual record does not.) The probability of occurrence of such an event is called *false drop probability*, fd, which is defined as follows.

$$\text{fd} = \frac{\text{Number of false matches}}{\text{Number of records which does not qualify the query}}$$

WS eliminates the need for a lookup table but the record signatures are of variable length. The inverted version of this method can be obtained by replacing the lookup table of IFSM with a hash table which points to the bit strings (or posting lists). The inversion for WS is shown in Figure 2. The word signature for the term *signature* is '0010' which is equal to 2.

Inversion of WS is the same method proposed by Faloutsos and Chang as *compressed bit slices* (CBS) [Faloutsos and Chang 1988]. In the inverted WS method if two common terms are hashed to the same signature, i.e., the same hash table position, there will be too many false drops. For the solution of this problem refer to [Faloutsos and Chang 1988].



Word signature for the term signature is '0010'
The 2nd position in the hash table points to the bit string (or posting list)
Figure 2. Inversion with word signatures.

2.1.2 Superimposed Coding

Superimposed coding (SC) solves two basic problems produced by IFSM: the growth of bit strings to the right when new records are added and the need for lookup table to determine the positions of the terms in the bit strings corresponding to the rows of IFSM [Christodoulakis and Faloutsos 1984]. Each term is hashed to a bit string of length *F* (size of the signature) with *m* bits set to '1' to obtain term signatures. There is a relationship between F, m, the total number of 1s in the record signature, and the number of terms in a record [Faloutsos 1985a]. A record signature is obtained by superimposing, bitwise ORing, the term signatures of the record terms (see Figure 3).

Terms	**Word Signatures**
computer	0 1 1 0 0 1 0 0 0
signature	1 0 1 0 0 0 0 0 1
extraction	0 0 0 1 1 0 0 1 0

d = { computer, signature, extraction }

Record signature for d : 1 1 1 1 1 1 0 1 1

Figure 3. Record signature generation using superimposed coding.

To answer a query, first the query signature is obtained by superimposing the signatures of the query terms. Then, record signatures are compared with the query signature. The records whose signatures contain 1s at the corresponding positions to the 1s of the query signature are selected. The additional cost introduced by false drops still remains in SC. In WS false drops are produced by hashing two different terms to the same signature. In SC, false drops are produced partially by hashing two different terms to the same signature as in the WS. However, false drops are mainly

produced due to the superimposition operation used to obtain record signatures. Although all terms are assigned different signatures, combination of term signatures may subsume the signatures of other terms. These records may seem to qualify a query containing the subsumed term. Therefore, the query evaluation is in fact a filtering process to reject unqualifying records. The false drops which pass the filtering process must be eliminated by accessing the actual records. This process is called false drop resolution.

High space overhead generated by the IFSM is reduced to a certain degree with additional processing cost in query evaluation, i.e., false drop resolution. Also, the growth to the right for new terms added to the dictionary by the new records is eliminated. The reduction in space overhead depends on the size of the record signature (F). For small F values, the space overhead will be lower; however, the false drop probability will be higher [Faloutsos 1987].

The efficiency of a physical access method can be measured with the resources used by the method, the time required to answer user queries, and the time required to perform insertion and update operations. Therefore, the gain in space and the loss in query processing due to false drops must be balanced.

2.2 Signature File Organization Methods

2.2.1 Vertically Partitioned Signature Files

Vertical partitioning of a signature file corresponds to IFSM. Vertical partitioning improves performance in query evaluation while insertion operations become expensive. It is preferable for the environments where insertion can be batched or are rare [Sacks-Davis et al. 1987].

2.2.1.1 Bit Sliced Representation

In *bit sliced signature files* (BSSF) the signature file is stored in column-wise order [Roberts 1979]. For query evaluation only the bit slices corresponding to the 1s in the query signature are retrieved. For a single term query, m bit slices (at most) are read, as opposed to one bit slice in the inverted file. Without compression the sizes of the bit slices will be equal to the number of records in the database. In the inverted files, additional time is required to determine the position of the bit slice corresponding to the query term. This requires a lookup table search.

2.2.1.2 Frame Sliced Signature Files

In *frame sliced signature files* (FSSF) the record signature is divided into k equal sized frames. Signature generation is performed in two steps: first a hashing function

is used to select one of the frames. Then, a second hashing function determines the positions of the m bits to be set to '1' in this frame [Lin and Faloutsos 1992]. The method minimizes the number of seek operations. Combining the bits of a term in a frame and storing that frame in consecutive disk blocks minimizes the number of seeks for dedicated storage devices. As a result the insertion and update operations require less time. On the other hand, corresponding bit slices to the '0' bits of the term signature are also transferred.

In the generalized version of FSSF, each word sets bits in n frames (GFSSF) [Lin and Faloutsos 1992]. When there is only one frame in the record signature, GFSSF is equivalent to the sequential signature file method. When there are F frames with length one bit, GFSSF converges to the BSSF method.

2.2.2 Horizontally Partitioned Signature Files

Horizontal partitioning of signature files eliminates the processing of a part of the signature file stored in row-wise order and thus improves performance. Proposed horizontal partitioning methods can be divided into two classes: single level and multilevel. Generally there is some additional space overhead due to additional search structures or unused space at the end of the partitions.

2.2.2.1 Single Level Methods

Single level methods use a part of the signature as a key. Three different methods proposed by Lee and Leng use superimposed coded signatures as record signatures and identify a part of it as the key portion [Lee and Leng 1989]. The key of the query signatures are extracted in the same way, and only these blocks which have the same key portion are accessed.

Linear hashing with superimposed signatures (LHSS) is another single level method proposed by Zezula et al [Zezula et al. 1991]. LHSS determines the number of bits in the key portion of the signature dynamically. A split function converts each signature into a page number between zero and n - 1 where n is the number of pages. Some of the pages are hashed at level h, i.e., the key portion is h bits long, while some of the pages are hashed at level h - 1. A split pointer is used to locate the first page hashed at level h - 1. The pages beginning from the split pointer up to the page with index 2^{h-1} are hashed at level h -1 (2^h-n pages). Performance of LHSS increases as the number of 1s in the key of the query signature increases. For a query containing all 0s, all of the pages must be accessed. The effect of non uniform record and query frequencies of the terms are investigated by Aktug and Can. The results show letting high discriminatory terms to set more bits than low discriminatory terms

increases the performance of LHSS. The effect of multiterm queries are inspected as well [Aktug and Can 1993].

2.2.2.2 Multilevel Methods

The signature trees method divides the signature file into blocks. The signature of the block is then obtained by superimposing the signatures in the group. This grouping operation continues until a few signatures are left at the top [Tharp 1988]. Since there is no pre computation to group similar signatures to the same block, for a query with more than a few relevant records, most of the blocks at leafs of the tree will contain at least one relevant record to the query.

The S-Tree method proposed by Deppish dynamically groups similar signatures during insertion [Deppish 1986]. A new record is added to the leaf page which contains similar signatures. The S-tree is kept balanced in a way similar to B-trees.

Unlike other multilevel methods, the method proposed in [Sacks-Davis et al. 1987] uses two different term signatures: record signatures and block signatures. Block signatures are larger than the record signatures. Signatures of the terms occurring in a record are superimposed to obtain the record signature. Record signatures are grouped in equal sized blocks such that each block occupies only one disk page. The block signature is obtained by superimposing the block signatures of the terms occurring in the records belonging to the block. Block signatures are stored in bit sliced form, while record signatures are stored in row-wise order.

2.2.3 Hybrid Methods

Hybrid methods use inversion and signature files in the same retrieval system. The aim is to combine the advantages of both systems while eliminating the disadvantages.

The method proposed by Faloutsos and Jagadish uses the posting list storage for rare terms and bit map storage for frequent terms [Faloutsos and Jagadish 1991]. For different environments different organizations for the bit map are proposed. The proposed method maintains the lookup table for all terms. Therefore, the space overhead generated by the lookup table is not eliminated. Also, the time required to search the lookup table is the same as in inversion.

The Compressed Bit Slices (CBS) method proposed by Faloutsos and Chan is the inversion of WS [Faloutsos and Chan 1988]. Instead of storing a lookup table, a hash table is used and each term generates an address in this hash table (see Figure 2). To resolve false drops produced by hashing two different terms into the same hash table location, a second hash function is used and the resulting signature of the term is stored in the posting buckets (Doubly compressed bit slices-DCBS). The search in the

lookup table is limited to the terms hashed to the same value with an increase in the space overhead generated by the intermediate blocks. False drops may be generated if both of the hashing functions produce the same signatures. The no false drop method (NFD) solves this problem with a cost of accessing the disk block containing the actual term in the document.

The hybrid access method proposed by Chang et al. uses an inverted index structure to store a lookup table [Chang et al. 1993]. For frequently occurring terms (80% of the terms) a signature file is used. For rare terms a posting list is maintained as in the inverted file representation.

3. PROPOSED EVALUATION METHOD

The performance of the bit slice signature file method depends on the number of on-bits in the query signature, i.e., the query weight. As the query weight increases, the cost of the query evaluation proportionally increases. For a bit slice of length N bits, and a query weight of W(Q),

$$N \cdot W(Q) \qquad (1)$$

bits must be read from the auxiliary storage.

To illustrate how the bit slice method works, and to describe the proposed incremental query processing method an example database is given in Figure 4. The example database contains eight records, six terms and on the average records contain two terms. Signature size is 9 bits and each word set m=3 bits to '1'. The value of m is the optimum value of m using the formula $F \cdot \ln 2/D$ [Faloutsos 1985].

To evaluate the single word query given below, bit slices s_1, s_4, and s_7 must be read and ANDed.

Query : retrieval
Query signature : 1 0 0 1 0 0 1 0 0

The result of the above query is '0 0 1 1 0 0 0 1'. Since the result bit string contains 1s at positions 3, 4, and 8, records d_3, d_4, and d_8 pass the filtering process. After false drop resolution the result set contains only d_3 and d_4, since d_8 is a false drop. At the first step the result bit string is set to s_1 and contains five 1s. Figure 5 illustrates how false drops are successively eliminated during query evaluation steps. Columns shows the result bit string after each evaluation step. For better understanding only on-bits are shown. If the evaluation of the query ceases at $step_2$, there will be two false drops.

Documents	**Terms**	**Signatures**
d1 = { computer, information }	access	0 0 0 1 0 0 1 0 1
d2 = { computer, signature }	computer	0 1 1 0 0 1 0 0 0
d3 = { information, retrieval }	extraction	0 0 0 1 1 0 0 1 0
d4 = { computer, retrieval }	information	0 1 0 0 1 0 0 1 0
d5 = { computer, signature, extraction }	retrieval	1 0 0 1 0 0 1 0 0
d6 = { computer, access }	signature	1 0 1 0 0 0 0 0 1
d7 = { computer }		
d8 = { access, signature }		

Signature file

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
d_1	0	1	1	0	1	1	0	1	0
d_2	1	1	1	0	0	1	0	0	1
d_3	1	1	0	1	1	0	1	1	0
d_4	1	1	1	1	0	1	1	0	0
d_5	1	1	1	1	1	1	0	1	1
d_6	0	1	1	1	0	1	1	0	1
d_7	0	1	1	0	0	1	0	0	0
d_8	1	0	1	1	0	0	1	0	1

N = 8 records, D = 2 words per record, F = 9 bits, m = 3 bits per word

Figure 4. Signature file representation of the example database.

	$step_1$	$step_2$	$step_3$
d_1			
d_2	1		
d_3	1	1	1
d_4	1	1	1
d_5	1	1	
d_6			
d_7			
d_8	1	1	1

$Step_1 \leftarrow s_1$

$Step_2 \leftarrow Step_1$ *and* s_4

$Step_3 \leftarrow Step_2$ *and* s_7

Figure 5. Evaluation steps using bit slices.

In bit-slice query evaluation method, as it can be observed in the example query evaluation, if a bit becomes an off-bit (has a value '0'), then it never becomes an on-bit (has a value '1') again. Therefore, the number of on-bits in the result of an AND operation will always be less than or equal to the number of on-bits of the result of the previous step. The positions of on-bits at the final step correspond to the record numbers which pass filtering. The numbers of on-bits in example query evaluation given in Figure 5 are 5, 4, and 3 through $step_1$ to $step_3$. If a particular bit becomes an off-bit, the corresponding bit of the bit slices which will be used in future evaluation steps will not be needed. Specific bits can not be read selectively from the auxiliary storage, since I/O operations are performed in terms of blocks. Indeed, for small databases, a bit slice may be stored in a few disk blocks. For example, a 4K bytes disk block can hold 32,768 bits. Reading a bit slice of a database with 100,000 records requires only four disk accesses. Therefore, for small databases the time wasted in reading unnecessary bits may have insignificant effect on the query processing time.

The number of disk blocks required to store one slice of the signatures of very large databases will be much more than a few disk blocks. The number of disk blocks required to store one slice of various very large database sizes is given in Table I. The database sizes vary from 10^6 to 10^8 records while block sizes vary from 256 to 4K bytes. (According to our definition, a database with 10^6 or more records is very large.)

Table I . Number of Disk Blocks Required to Store One Bit Slice

Block size (bytes)	Database size in no. of records		
	10^6	10^7	10^8
256	489	4,883	48,829
512	245	2,442	24,415
1,024	123	1,221	12,208
2,048	62	611	6,104
4,096	31	306	3,052

For very large databases, the cost of reading unnecessary bits during query evaluation will be much more than a few disk accesses. Therefore, it must be avoided as much as possible. Physical storage of some bit slices of the example database is given in Figure 6. A block size of two bits is assumed. Each bit slice is divided into four partitions with two records. (In real databases the last block may not be completely full.) Result blocks are obtained by making the same partitioning in the result bits of the evaluation steps. Each result block corresponds to a physical disk block of the bit slices.

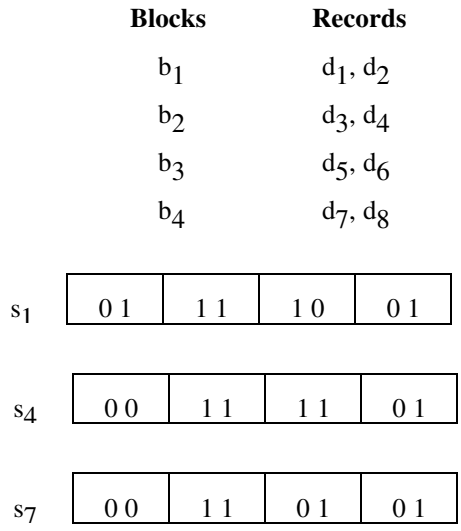


Figure 6. Storage of bit slices in blocks.

If all bits in a result block become off-bits, that block is called an off-block, otherwise it is called an on-block. Initially, all of the result blocks are filled by reading the first bit slice being processed. Then starting from the second bit slice to the last one, if a result block is an on-block, the corresponding disk block is read and ANDed with this result block. For off-blocks, corresponding disk blocks are not read from the disk, since the outcome of the AND operation with an off-block will always be an off-block. Figure 7 shows on-blocks for the example query evaluation. The columns are the on-blocks at the result of each step (see Figure 5). Disk blocks needed at each step are dynamically determined during query evaluation. For this reason the proposed method is called *incremental query evaluation*. For ease of reference, the incremental bit slice query evaluation is called INCBIT, and the standard bit slice query evaluation is called STANBIT.

	Step1	Step2	Step3
b1	1		
b2	1	1	1
b3	1	1	
b4	1	1	1

Figure 7. On-blocks for the example query.

The total number of disk block reads in INCBIT is 11, as opposed to 12 disk block reads in STANBIT. This corresponds to a 9% performance gain. In another words, 91% of STANBIT block reads are required in INCBIT. The later measure will be

used as the performance evaluation criterion. A lower g_i values mean better performance.

$$g_i = \frac{\text{Number of page accesses in INCBIT up to step}_i}{\text{Number of page accesses in STANBIT upto step}_i} \qquad (2)$$

Table II. g_i Values for Example Query Evaluation

	STANBIT Blocks	INCBIT Blocks	Blocks Saved	Total STANBIT Blocks	Total INCBIT Blocks	g_i
$Step_1$	4	4	0	4	4	1.000
$Step_2$	4	4	0	8	8	1.000
$Step_3$	4	3	1	12	11	0.916
Total	12	11	1	12	11	0.916

3.1 Estimating Number of On-Bits for Incremental Query Evaluation

Let on_i be the number of on-bits at $step_i$ for a query with q_{rel} relevant records. It may take any value between zero and N. (Definitions of important symbols are provided in Table III.) on_i contains q_{rel} on-bits as well as fdb_i false drop on-bits. In the later steps of query evaluation, most of the false drop bits are eliminated. Since q_{rel} depends on the query issued, to simplify the analysis, on_i will be estimated for zero hit queries. In section 3.4 the formulation will be modified for non zero q_{rel} values.

$$on_i = q_{rel} + fdb_i \qquad (3)$$

3.1.1 False Drop Probability

The signature of a particular record, on the average, will contain

$$\lceil F \cdot op \rceil$$

on-bits where F is the signature size and op is on-bit probability. By assuming a random distribution of on-bits in a record signature, i.e., no interdependencies between on-bits, the number of on-bits in a slice can be computed as follows:

$$\text{Number of } on-bits \text{ in a slice} = \frac{N \cdot F \cdot op}{F} = \lceil N \cdot op \rceil \qquad (4)$$

Note that, the probability of a particular bit being an on-bit in a slice is the same with the on-bit probability in a record signature. op is a measure of the on-bit density

in the signature file. In INCBIT higher on-bit density implies lower performance, i.e., more number of disk accesses. Under optimal design conditions, for a given signature size F, half of the bits in the signature must be on-bit [Faloutsos 1988]. This corresponds to a probability of 0.5 for a particular bit being on-bit.

Table III. Definition of Important Symbols

Symbol	**Meaning**
dp	: no. of blocks required to store a record
fb_i	: no. of off-blocks at $step_i$
fd_i	: false drop probability at $step_i$
fdb_i	: no. of false drop bits in the result bit string at $step_i$
k	: no. of frames
m	: no. of bits in a block
n	: no. of blocks for a bit slice
nb_i	: no. of off-blocks detected at $step_i$
ob_i	: no. of on-blocks at $step_i$
on_i	: no. of on-bits in result bit string at $step_i$
op	: probability of a bit being '1' (on probability)
q_{rel}	: no. of relevant records to the query
r	: no. of frames selected to set bits for each term
rd_i	: Total no. of blocks read up to $step_i$
s	: the length of each frame in bits
sv_i	: Total no. of blocks saved up to $step_i$
t_{read}	: time required to read a block
t_{seek}	: time required to position reading head
D	: average number of words in a record
F	: size of signature in bits
M	: no. of on-bits in a record signature
N	: no. of records
R	: no. of bits set in each frame
S	: no. of bits set by each word
W(Q)	: no. of on-bits in the signature of query Q

To find the relationship between the number of on-bits processed from the query signature (i.e., step number i) and the false drop probability fd_i (for $step_i$), the following formula will be used [Sacks-Davis et al. 1987].

$$fd_i = \left[1 - \left(1 - \frac{S}{F} \right)^D \right]^i \qquad (5)$$

The above formula assumes that each word of the record sets exactly S bits to '1'.

The false drop probability decreases as more on-bits from the query signature is processed. Note that, in general even after processing the last on-bit of the query signature, there will be a non zero false drop probability. For low weight queries, the evaluation may be completed with a high fd value. Thus, there will be many false drops which must be resolved by referring to the original records. If the query evaluation is considered as a step by step process, intuitively, the probability of finding an on-bit in the result bit string of $step_i$ is the following.

$$\begin{aligned} fd_1 &= op \\ &\vdots \\ fd_i &= fd_{i-1} \cdot op = op^i \end{aligned} \qquad (6)$$

To prove the above relationship between the false drop probability, op, and the number of on-bits processed from the query signature, a modified version of the formula provided by Christodoulakis and Faloutsos [Christodoulakis and Faloutsos 1984] can be used. This formula computes the number of on-bits for given F, D, and S values.

$$M = F \cdot \left[1 - \left(1 - \frac{1}{F} \right)^{S \cdot D} \right] \qquad (7)$$

Equation 7 assumes the bits a word sets to '1' may not be distinct, i.e., each word may not set exactly S bits to '1'. On the other hand, false drop is computed with the assumption that each word sets exactly S bits (see equation 5). Therefore, equation 7 is modified to consider this distinction.

$$M = F \cdot \left[1 - \left(1 - \frac{S}{F} \right)^D \right] \qquad (8)$$

To find op, both sides of equation 8 is divided by F. The result is one bit version of equation 5.

$$op = \frac{M}{F} = \left[1 - \left(1 - \frac{S}{F} \right)^{D} \right] = fd_1 \qquad (9)$$

$$fd_i = op^i \qquad (10)$$

This result is not surprising. At $step_1$ the probability of a result bit being an on-bit is the same as the probability of a bit of a slice being on-bit, which is op. At $step_2$, since the result bit string is the ANDed with the corresponding bit slice of the second on-bit of the query signature, on-bit probability in the result bit string will be op^2. To generalize, since the result bit string will be obtained by ANDing i bit slices, at $step_i$ on-bit probability will be op^i.

For zero hit queries q_{rel} will be zero and the expected number of on-bits at $step_i$, on_i, will be equal to the number of false drops, fdb_i, at this step.

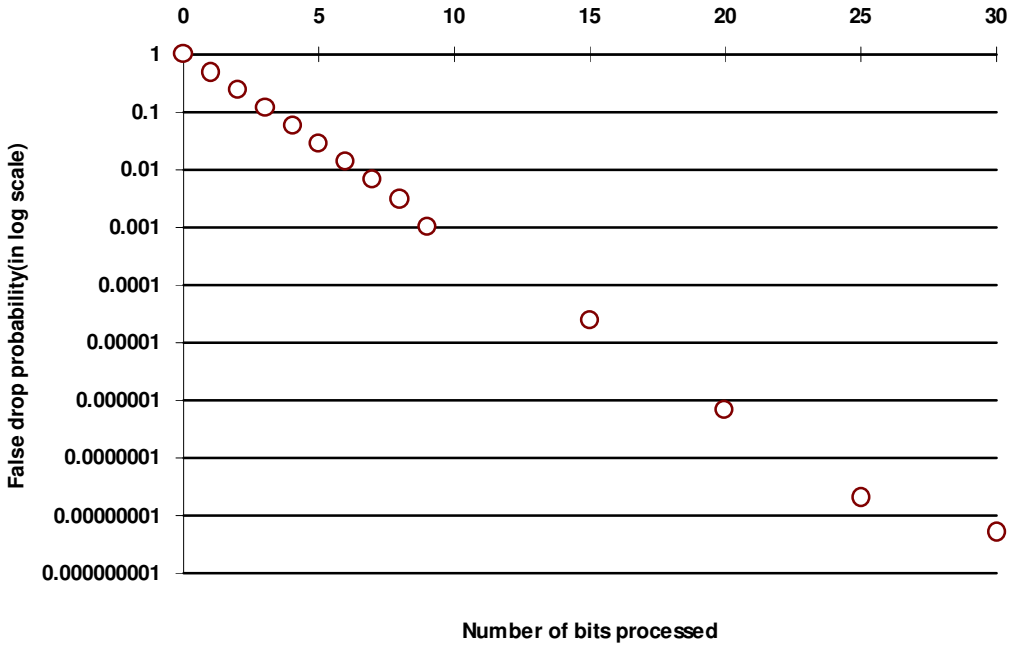
$$on_i = fdb_i = N \cdot fd_i \qquad (11)$$

False drop probabilities computed by equation 5 are given in Table IV. The parameters used are, $N = 10^7$, D = 20, F = 300, and S = 10. (In the rest of the paper numeric examples-values will be for this database. Note that, after processing 23 bit slices the expected number of false drops becomes only one record (see Figure 8). For both STANBIT and INCBIT, no need to continue with the evaluation of the query after the first 23 on-bits of the query signature are processed. Partial evaluation of bit slices are discussed in section 3.5.

Table IV. Step Number, i, and the Corresponding fd_i Values
($N=10^7$, D=20, F=300, S=10)

i (Step No)	fd_i
1	0.49238451
2	0.24244250
3	0.11937493
4	0.05877837
5	0.02894155
10	0.00083761
15	0.00002424
20	0.00000070
23	0.00000008

In table IV, an important problem reveals itself. Assume that a one word zero hit query is submitted to the database inspected in Table IV. Since each word sets at most 10 bits to '1', there will be at most 10 on-bits in the query signature. After processing the bit slice corresponding the last on-bit of the query signature, there are still 8,376 false drops, which must be eliminated by accessing the original records. Although the ratio of the false drops to the number of records in the database is constant, the time required to answer a query is linearly proportional to the number of records in the database. One of the solutions to that problem is to increase S, i.e., the number of bits set by each word. If D is assumed a constant to fulfill the optimal design conditions, F must also be increased. If the number of bits set by each word is increased by $\propto$ bits, to satisfy the optimal design conditions, the size of F must be increased by $\propto \cdot D/\ln 2$ bits. In the example given above, to obtain 243 false drops for single word queries S must be 15, which means 50% increase in S. Also the size of F must be increased to 445 bits, which means 48% increase in space overhead.



D = 20, F = 300, S = 10

Figure 8. fd_i values and number of bit slices processed.

For very large databases, single word queries with a non selective word will produce very large result sets. With the high number of non reducible false drops, there will be too many on-bits in the result string. Accessing the original records and resolving false drops will require unacceptable query processing time. Unless the result is used as an input to another computerized system, the user can not screen and inspect that many records. Therefore, in very large databases the users will use, or

will be forced to use, additional query term(s) if the query term is non selective. The expected number of on-bits for a query with three terms will be 29 (F = 300, S = 10), which is sufficient to reduce the expected number of false drops to a negligible level.

3.2 Estimating the Number of Block Accesses

INCBIT requires only reading of on-blocks for query evaluation. The number of on-blocks depends on the number of on-bits, the size of each block, and the total number of blocks. There is a relationship between block size, m, and the total number of blocks, n

$$n = \left\lceil \frac{N}{m} \right\rceil \qquad (12)$$

where N is the database size in terms of number of records.

For the same number of on-bits, the probability of finding an off-block decreases as the block size increases. For the extreme case of n = 1, i.e., one block is large enough to store a whole bit slice, if there is at least one on-bit, the probability of finding an off-block is zero. In the case of the other extreme, i.e., if m = 1, each bit is stored in a separate block, the probability of finding an off-block is 1 - fd_i. If *k* on-blocks are found for *d* on-bits, the amount of data read but not used is the following:

$$k \cdot m - d$$

For large block sizes, the amount of unnecessary bits transferred will be larger compared to small block sizes. In real applications the block size is determined by the physical sector size of the secondary storage used.

To estimate the number of blocks that contain at least one of the on-bits, the formula provided by Yao can be used [Yao 1977].

$$e(r,m,n) = \left[1 - \left(1 - \frac{m}{m \cdot n}\right) \cdot \left(1 - \frac{m}{m \cdot n - 1}\right) \cdots \left(1 - \frac{m}{m \cdot n - r + 1}\right) \right] \qquad (13)$$

In the above formula, *n* is the number of blocks, *r* is the number of on-bits, *m* is the size of each block in bits, and *e(r,m,n)* is the probability that an arbitrary block contains at least one on-bits, i.e., the probability of being an on-block. For efficiency a noniterative approximation of e(r,m,n) given by Whang et al. will be used [Whang et al. 1983].

$$e(r,m,n) \approx \left[1-\left(1-\frac{1}{n}\right)^{r} + \left[\frac{1}{n^2 m} \times \frac{r(r-1)}{2} \times \left(1-\frac{1}{n}\right)^{r-1} \right] + \left[\frac{1.5}{n^3 m^4} \times \frac{r(r-1)(2r-1)}{6} \times \left(1-\frac{1}{n}\right)^{r-1} \right] \right] \quad (14)$$

when $r \rangle m \cdot n - m$

$$e(r,m.n) = 1$$

when $r \leq m \cdot n - m$.

At $step_1$, although there may be off-blocks; however, they can not be predicted without inspecting, hence at $step_1$ all of the blocks are read. At $step_2$, only on-blocks of the bit slice processed at $step_1$ are read. Starting from $step_2$, in succeeding steps, each new detected off-block decreases the number of blocks (n) by one. In INCBIT off-blocks of a step will stay as off-blocks in the further steps. Since they contain only off-bits, none of the on-bits in further steps may fall into an off-block. The distribution of bits are no longer random for off-blocks, the distribution is in fact definite, off-blocks only contain off-bits.

Let the number of off-blocks at $step_i$ be fb_i. For zero hit queries, ob_i (the number of on-bits at $step_i$) is estimated as the number of false drops after processing *i* bit slices. These ob_i on-bits are distributed in the

$$ob_i = n - fb_i$$

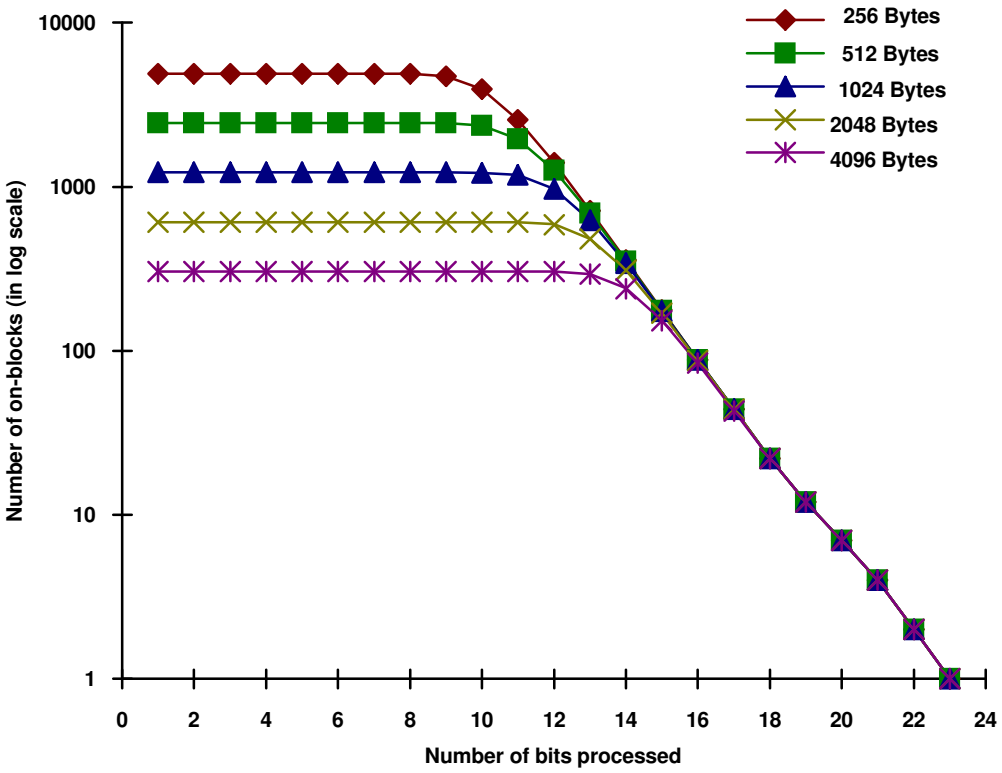
blocks. By inspecting the distribution of the on-bits in the on-blocks of $step_{i-1}$, the number of expected *new off-blocks*, nb_i, is determined. The sum of fb_i and nb_i will be fb_{i+1}, i.e., the number of off-blocks for $step_{i+1}$. This iterative process is formulated as follows.

$$\begin{aligned} fb_1 &= 0 \\ ob_1 &= n - fb_1 = n \\ &\vdots \\ ob_i &= ob_{i-1} \cdot e(N \cdot fd_{i-1}, m, ob_{i-1}) \\ nb_i &= ob_i - ob_{i-1} \end{aligned} \quad (15)$$

By substituting previous ob values into equation 15, a general formula is obtained for $i > 1$ where $ob_1 = n$.

$$ob_i = n\prod_{r=1}^{i-1} e(N \cdot fd_{i-r}, m, ob_{i-r}) \qquad (16)$$

For our example database (of 10^7 records), the expected number of false drops for different block sizes are shown in Figure 9. For the first nine bit slices nearly all of the blocks are on-block. At $step_9$ the number of expected false drops is 43,782. After $step_9$ the number of on-blocks exponentially drops to one in 14 additional steps. Another important property is that for small block sizes fewer number of bit slices must be processed to reach the starting point of drop tendency: For a block size of 256 bytes the decrease in the number of on-blocks begins at $step_8$, while for a block size of 4,096 bytes the decrease begins at $step_{13}$. As a result, for INCBIT small block sizes will improve the performance.



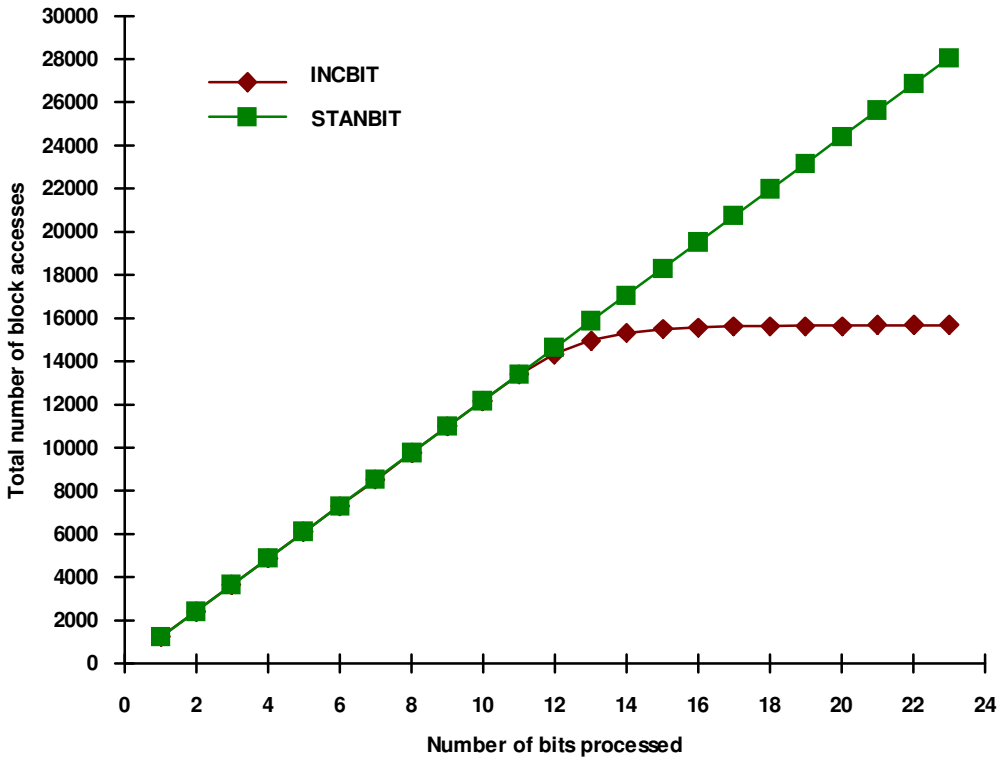
D = 20, F = 300, S = 10, N = 10^7

Figure 9. The number of on-blocks vs. number of bits processed.

The total number of block accesses at $step_i$, rd_i, is computed by summing ob_i values for all steps starting from $step_1$ to $step_i$ (including $step_i$).

$$rd_i = \sum_{k=1}^{i} ob_i = \sum_{k=1}^{i} n\prod_{r=1}^{k-1} e(N \cdot fd_{k-r}, m, ob_{k-r}) \qquad (17)$$

Also, the summation of fb_i values gives the number of blocks saved (sv_i). Total number of block accesses for a block size of 1,024 bytes are shown in Figure 10. For the first 11 steps, INCBIT requires the same number of block accesses with STANBIT. After $step_{14}$ the total number of block accesses is nearly constant.



D = 20, F = 300, S = 10, N = 10^7, m = 1,024, n = 1220

Figure 10. Total number of block accesses vs. number of bits processed.

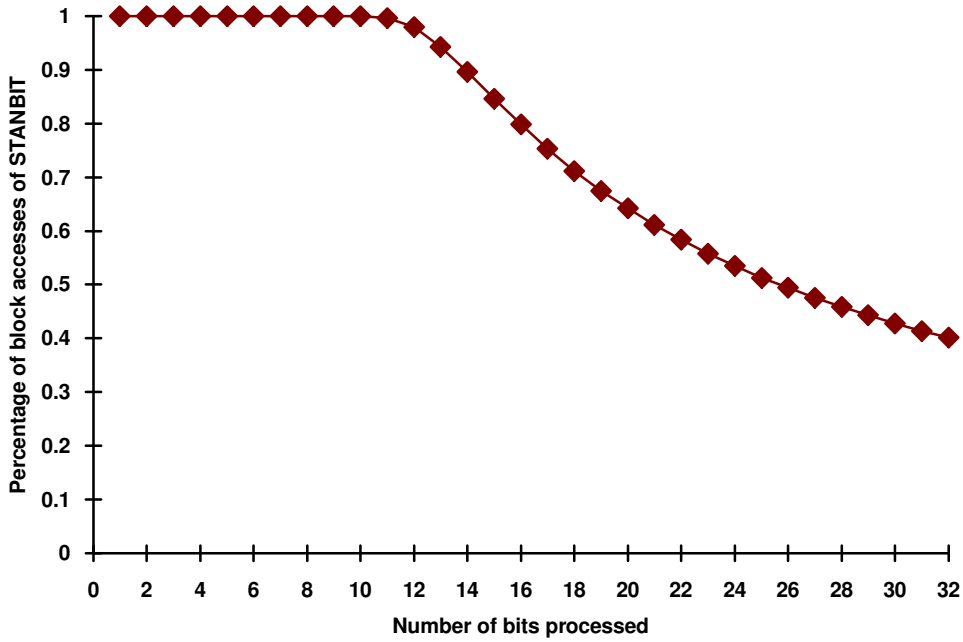
3.3 Comparing INCBIT with STANBIT

The comparison criterion, g_i, which is the ratio of the total number of block accesses for INCBIT (rd_i) to the total number of block accesses for STANBIT, is given in equation 2. Equation 18 is obtained by substituting rd_i given in equation 17 into equation 2.

$$g_i = \frac{rd_i}{n \cdot i} = \frac{\sum_{k=1}^{i} n \prod_{r=1}^{k-1} e(N \cdot fd_{k-r}, m, ob_{k-r})}{n \cdot i}$$

$$g_i = \frac{\sum_{k=1}^{i} \prod_{r=1}^{k-1} e(N \cdot fd_{k-r}, m, ob_{k-r})}{i} \qquad (18)$$

As g_i decreases, the performance of INCBIT increases. For a database with 10^7 records, the decrease in g_i decreases as more bit slices are processed is shown in Figure 11. At $step_{23}$, g_i has a value of 0.535. This is the expected performance gain for zero hit queries. Since the expected number of false drops can not be decreased beyond that step, although the decrease in g_i continues after that point, the evaluation of the query should be ceased at that step, since the expected number of on-bits will not change in the future steps.



D = 20, F = 300, S = 10, N = 10^7, m = 1,024, n = 1220

Figure 11. g_i values vs. number of bits processed.

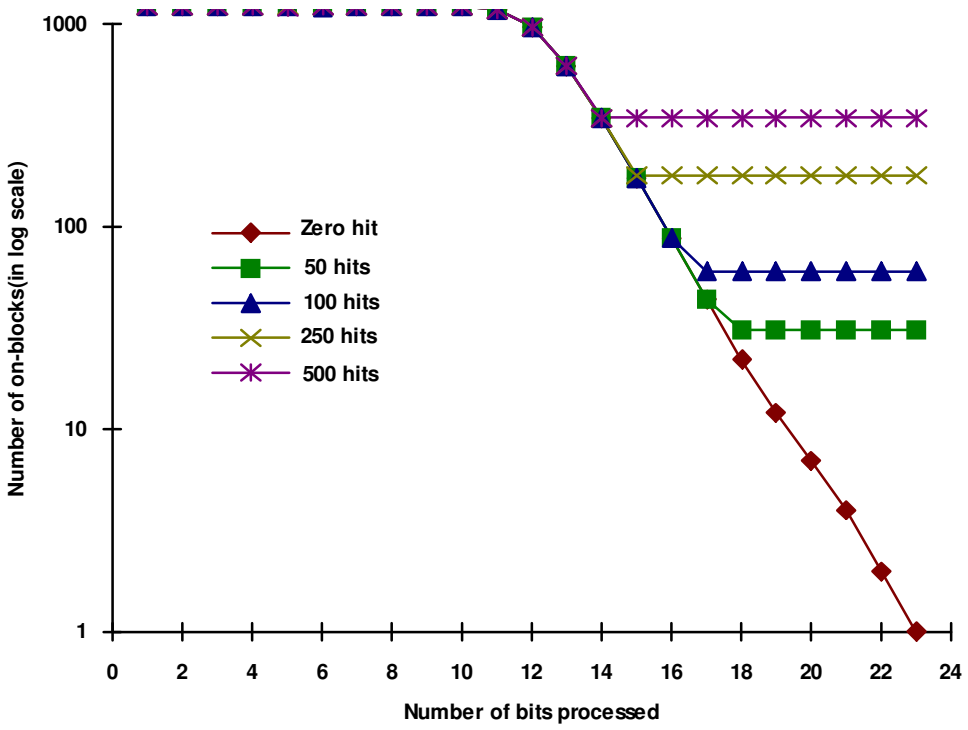
3.4 Analysis of Non Zero Hit Queries

For non zero hit queries, in equation 3, q_{rel} is non zero. Therefore, on_i is no longer equal to the fdb_i. Since the number of expected false drops becomes one at $step_{23}$, the number on-bits will be $q_{rel}+1$. This means on_i can not be less than q_{rel} at any step. Equation 19 accommodates this lower bound.

$$on_i = \begin{cases} fdb_i & if \quad fdb_i \geq q_{rel} \\ qrel & otherwise \end{cases} \qquad (19)$$

This lower bound imposed on the expected number of on-bits directly affects the expected number of on-blocks. If q_{rel} is high enough, the expected number of on-blocks will be nearly n (all of the blocks), i.e., the performance of proposed method

deteriorates. The effect of non zero q_{rel} on the number of on-blocks can be seen in Figure 12. The reason of this deterioration is random distribution of the records. To overcome this difficulty, an improved method is proposed in section 5.2.



D = 20, F = 300, S = 10, N = 10^7, m = 1,024, n = 1,220

Figure 12. Number of on-blocks vs. number of bits processed in case of non zero hit queries.

3.5 Partial Evaluation of Queries

The query evaluation process can be considered as a false drop resolution operation. If the expected number of resolvable false drops at any step can be checked by accessing the original records in less time than eliminating those false drops, there is no need to continue the evaluation of the query any further.

Assume that the query evaluation is ceased at $step_r$. There will be

$$q_{rel} + N \cdot fd_r$$

on-bits in the result bit string. That many records must be accessed and compared with the query. Since the number of relevant records to the query is q_{rel}, at least q_{rel} records must be accessed independent of the stopping step. So, the additional cost produced by stopping at $step_r$ is the cost of access and comparison of the uneliminated false drops. If the evaluation is conducted until all on-bits in the query are processed, there will be

$$N \cdot fd_{W(Q)}$$

false drops, where W(Q) is the weight of the query. The number of resolvable false drops will be

$$N \cdot \left(fd_r - fd_{W(Q)} \right).$$

The cost of stopping at $step_r$ is given in equation 20. In this formula t_{seek} and t_{read}, respectively, indicate the time required to position the read head and the time required to read one disk block.

$$Loss_r = N \cdot \left[fd_r - fd_{W(Q)} \right] \cdot \left[dp \cdot t_{read} + t_{seek} \right] \qquad (20)$$

In cost calculations memory operations, such as comparing a record with the query, is ignored.

The gain of stopping at $step_r$ is the cost of performing the query evaluation from $step_{r+1}$ to $step_{W(Q)}$. With a pessimistic assumption, time required to complete the evaluation is seeking and reading all of the on-blocks in the future steps. (There may be contiguous on-blocks, but they are assumed non contiguous.) The stopping condition can be defined as follows.

$$Gain_r = \left(t_{seek} + t_{read} \right) \cdot \sum_{i=r+1}^{W(Q)} ob_i$$

$$Gain_r \geq Loss_r \qquad (21)$$

Whenever the above inequality holds, the evaluation of the query must be ceased. Note that, the evaluation of the query may be completed before the stopping point, i.e., all of the on-bits of the query signature may be processed.

The same approach with a small modification for stop condition can be applied to STANBIT. *Lost* will be the same, but *Gain* is modified to reflect the evaluation method of STANBIT as follow:

$$Gain_r = t_{seek} + t_{read} \cdot n \cdot \left(W(Q) - r \right) \qquad (22)$$

For small databases equation 22 will give close approximations to the experimental results with a dedicated disk. However, for very large databases, even one slice will require much more space than a track size for some disks. For example, one bit slice

of a database with 10^7 records requires 1,220 K bytes. There will be head movements from sector to sector, even if they are close to each other, which will take time. On the other hand, reading more than 1 MB of data at once will require special hardware and buffering mechanism. If reading of data is stopped, there will be the same additional time to wait for the required sector to be read to pass under the reading head.

4. USING INCBIT in FRAME-SLICED SIGNATURE FILES

In frame-sliced signature files the record signature is generated by using two different hashing functions [Lin and Faloutsos 1992]. The record signature (F bits long) is divided into *k* frames of *s* bits each. The first hashing function selects one of the k frames. The second hashing function is used to set *R* bits in that frame.

The signature file is stored in frame-wise order. Since each frame is s bits long, a frame will contain $s \cdot N$ bits. Consecutively, the number of blocks in frame-sliced storage will be $s \cdot n$, where n is the number of blocks in the bit-sliced storage. The increase in the number of blocks increases the performance of INCBIT (see Figure 9). (Storing s bits for each record instead of one bit decreases the number of records in a block.) On the other hand, $s - R$ unnecessary off-bits are read with on-bits for each retrieved frame. This is a performance decreasing factor for INCBIT, since unnecessary bits are read.

In GFSSF, each word selects *r* distinct frames and sets *R* bits in each frame. The false drop probability of GFSSF derived by Lin and Faloutsos [Lin and Faloutsos 1992] is given below. (P_{set} is modified to be consistent with INCBIT: each term sets exactly R bits in each selected frame.)

$$fd \cong \sum_{t_1=0}^{D} B(D,t_1,r/k) \cdot P_{set}(t_1,R) \cdot \left\{ \sum_{t_2=0}^{D} B(D,t_2,r/(k-1)) \cdot P_{set}(t_2,R) \left[\cdots \left(\sum_{t_r=0}^{D} B(D,t_r,r/(k-r+1)) \cdot P_{set}(t_r,m) \right) \cdots \right] \right\} \quad (23)$$

where

P_{set}(t,R) = Probability(R bits found set / t words in that frame)

$$= \left[1 - \left(1 - \frac{R}{s} \right)^t \right]^R$$

B(D,t,p) = Probability(t words being hashed into one frame)

$$B(D,t,p) = \binom{D}{t} \cdot p^t \cdot (1-p)^{D-t}$$

The false drop probability given in equation 23 for one term can be used to obtain an approximation of false drop probability for multi-term queries [Lin and Faloutsos 1992].

$$fd_i \cong fd^i \qquad (24)$$

The on-bits of a term are grouped in a frame and can only be read together. Therefore, in frame-sliced query evaluation s successive evaluation steps of INCBIT are performed in one step. The number of steps in INCBIT formulas correspond to the number of frames used in incremental frame-sliced query evaluation. The number of frames to be retrieved for multi-term queries can be computed by using equation 25, where t is the number of terms used in the query.

$$\text{no. frames(t)} = k\left[1-\left(1-\frac{r}{k}\right)^t\right] \qquad (25)$$

Since BSSF is a special case of GFSSF [Lin and Faloutsos 1992], equations 12, 16 and 17 are rewritten to cover bit-sliced and frame-sliced query evaluation. For BSSF the parameter values of GFSSF are s=1, k=F, and r=S.

$$n = s \cdot \left\lceil \frac{N}{m} \right\rceil \qquad (26)$$

$$ob_i = n \prod_{r=1}^{i-1} e(N \cdot fd_{i-r}, m/s, ob_{i-r}) \qquad (27)$$

$$rd_i = \sum_{k=1}^{i} ob_i = \sum_{k=1}^{i} n \prod_{r=1}^{k-1} e(N \cdot fd_{k-r}, m/s, ob_{k-r}) \qquad (28)$$

5. FUTURE WORK

The proposed method, INCBIT, imposes a conceptual horizontal partitioning by means of blocks used to store bit slices. By arranging the records in the signature file, this horizontal partitioning may be used to increase performance of the proposed method. Compared to STANBIT, without any additional pre-computation or space overhead, INCBIT provides over 40% reduction in the amount of data transferred from secondary storage to main memory. In the rest of this section three methods to increase the performance of INCBIT are described. The effects of these methods on the performance of INCBIT will be measured by analytical and experimental methods.

Basically, to improve performance, the number of on-blocks in all steps must be decreased. The number of on-blocks depends on the number of on-bits. The sources of the final on-bits are false drops, and the bits corresponding to the records that satisfy the query. False drops can be eliminated to a certain degree. On the other hand, the on-bits generated by the retrieval set of the query cannot be eliminated.

Another research topic will be the investigation of the effect of sequentiality assumption on the performance of the proposed method. (It assumes that for a bit slice consecutive disk blocks can be allocated and can be read without any interruption.) The motivation behind the original bit slice method and many other methods is the sequentiality assumption which cannot be satisfied in some operating system environments such as UNIX. The sequentiality assumption and random distribution of blocks are two extreme cases. As the percentage of the file in which the sequentiality assumption holds changes, the corresponding change in performance will be inspected.

5.1 Accelerating the Reduction in Expected Number of False Drops

One way of improving the performance is that on-blocks produced by false drops can be decreased by providing rapid reduction in false drop probability as we process more number of bit slices. According to equation 10, the false drop probability is directly related to the probability of a particular bit being an on-bit, i.e., the on-bit density. Figure 9 shows that most of the total number of on-blocks are produced at the first nine steps. If the on-bit density of the first bit slice to be processed were 0.25 instead of 0.5, the cut-off point for the start of the exponential reduction in the number of on-blocks would have been $step_8$, instead of $step_9$. Also, the expected number of false drops would drop to one in 22 steps, instead of 23 steps.

Increasing the signature size for the same S and D values, decreases the on-bit probability, hence also decreases the false drop probability (see equation 5). For our example database (S = 10 and D = 20), if F is increased from 300 to 450, the on-bit probability becomes 0.362. Then the expected number of false drops becomes one in 16 steps, as opposed to 23 steps for F = 300 (op = 0.492). The main disadvantage of this solution is the increase in space overhead [Faloutsos 1985b].

5.1.1 Vertical Fragmentation of Signature

The optimal value for the on-bit density is 0.5 [Christodoulakis and Faloutsos 1984]. On the other hand, for the multiword queries all of the on-bits of the query signature are not used (see Figure 9). For our example database the expected number of false drops becomes one in 23 steps. If a query has more than 23 on-bits, the rest of the on-bits are not used in query evaluation. So, the number of bits set by each word may be

decreased. This also decreases the on-bit density, hence the performance of INCBIT increases. For example, if each word sets eight bits instead of 10 bits, on-bit density will be 0.417. The number of evaluation steps to obtain an expected false drop value of one will be 21.

For queries with one or two words, reducing the number of bits set by each word also reduces the number of on-bits in the query signature. As a result, the query evaluation is completed with a high number of expected false drops. For example, a one word query will produce 9,246 expected number of false drops with eight bits, as opposed to 8,377 false drops with 10 bits. As a result there will be 869 more false drop resolution operations.

The discussion presented above can be summarized by the facts listed below. The approaches sketched should consider these facts.

- The optimality condition is valid in a signature. Since for high weight queries all of the on-bits are not used by INCBIT, the optimality condition should be reexamined.
- Between low weight queries and high weight queries, there is a cost benefit relationship to decide on the number of bits set by each word.
- The reduction in the false drop probability at the very beginning of the query evaluation steps must be high.

To reduce the effect of unused query signature on-bits, the signature can be divided into *f* fragments, such that

$$F = F_1 + F_2 \cdots F_f .$$

(The space overhead does not change.) Each word sets S_1 bits in the first fragment, S_2 bits in the second fragment, and S_f bits in the last fragment. By using equation 29 S_i value can be computed for the given F_i and op_i values. On-bit densities of the fragments satisfy the condition below.

$$k < m \;\Rightarrow\; op_k \le op_m .$$

$$S_i = \frac{F_i \cdot \ln\left(\dfrac{1}{1-op_i}\right)}{D} \qquad (29)$$

For query evaluation, if needed, all bits from the lowest on-bit density fragment are first used. Later, the bits from the higher on-bit density fragments are used. For single-term queries, the expected number of false drops will be higher than the

expected number of false drops obtained for optimal conditions. As the number of query terms increases, more bits from the lowest on-bit density fragment will be used. Hence, the loss due to unused on-bits of the query will be smaller. There is a cut-over point depending on the number of query words used. The idea behind this method is to reduce the amount of unused information present in the record signatures due to unused on-bits of the query signature. Three example fragmentation schemes are given below. Expected false drop probability values for example fragmentation schemes are given in Table V.

Parameters of one fragment version:

F= 300, S= 10, D= 20, op= 0.492

Parameters of two equal sized fragments:

F_1 = 150, S_1 =3, D= 20, op_1 =0.332
F_2 = 150, S_2 =5, D= 20, op_2 =0.492

Parameters of three equal sized fragments:

F_1 = 100, S_1 =1, D= 20, op_1 =0.182
F_2 = 100, S_2 =2, D= 20, op_2 =0.332
F_3 = 100, S_3 =3, D= 20, op_3 =0.456

Table V. fd Values for Example Fragmentation Schemes

No. of	One Fragment		Two Fragments			Three Fragments		
Terms	fd	Steps	fd	Steps	Gain	fd	Steps	Gain
1	0.00083761	10	0.00106285	8		0.00191019	6	
2	0.00000070	20	0.00000113	16		0.00000364	12	
3	0.00000004	24	0.00000004	19	21%	0.00000003	16	33%
4	0.00000004	24	0.00000003	18	25%	0.00000003	14	42%
5	0.00000004	24	0.00000002	16	33%	0.00000002	13	46%
6	0.00000004	24	0.00000002	16	33%	0.00000004	12	50%
7	0.00000004	24	0.00000002	16	33%	0.00000002	12	50%
8	0.00000004	24	0.00000002	16	33%	0.00000004	11	54%
9	0.00000004	24	0.00000002	16	33%	0.00000002	11	54%
10	0.00000004	24	0.00000002	16	33%	0.00000004	10	58%

To find the optimal values for S_i and op_i a cost function depending on the frequencies of the queries with different word counts will be prepared and optimized.

A more sophisticated extension of the method described above is obtained by including the query and record frequencies of terms (i.e., the term discriminating values) and increasing the number of the vertical fragments [Faloutsos 1985a, Aktug and Can 1993]. To take into account non-uniform frequencies may require a lookup table which increases the space overhead.

5.2 Record Clustering

There is a lower bound in the number of expected on-blocks for non-zero hit queries. Assuming all of the false drops are eliminated, this lower bound is given below.

$$ob_{i+1} = ob_i \cdot e(q_{rel}, m, ob_i) \qquad (30)$$

The basic assumption in equation 30 is that the on-bits of the result bit string indicating possibly relevant records to the query are randomly distributed. However, the number of on-blocks may be decreased by distributing the records to the blocks so that the records that may be retrieved together are placed in fewer number of blocks. The record clustering problem is recognized as being NP-hard [Yu et al. 1981, Omiecinski and Scheuermann 1983]. This problem was investigated by various researchers [Willet 1988]. Some of these heuristic methods are the sketched below.

Jacobsson obtains a sort key by ordering the attributes in decreasing order of usage in the queries [Jacobsson 1980]. A subset of the blocks are read and the records in these blocks are sorted in the key order, and then, reassigned to the same blocks. This operation continues until no more unsorted blocks are left. This method is called partial sort method. Also, clustering around centroids, i.e., cluster representatives, are inspected.

Omiecinski uses the query frequencies and the relevant records to the queries [Omiecinski and Scheuermann 1983]. The proposed method, called SPLITMERGE, first generates disjoint subsets of the records by using relevant records to the queries; then these subsets are merged to minimize expected query evaluation cost. The cost function is given in equation 31. $F(Q_i)$ is the frequency of the query Q_i and $P(Q_i)$ is the number of blocks which contain records for the query Q_i, where M is the total number of queries.

$$C = \sum_{i=1}^{M} F(Q_i) \cdot P(Q_i) \qquad (31)$$

Yu et al. propose an adaptive method based on the retrieval set of queries [Yu et al. 1985] First, all records are randomly distributed on a line. Then, repeatedly relevant records to the queries are moved towards their central point. To prevent collapsing all records, some randomly chosen records are moved away. The algorithm stops when a convergence criteria is satisfied. Geometrically close records are most likely will be retrieved by the same queries. Therefore, they are placed in the same block.

McErlean et al. use simulated annealing algorithms in record clustering [McErlean et al. 1990]. Heuristic methods usually follow cost decreasing steps. In the simulated

annealing method, depending on the Boltzmann probability factor, cost increasing steps are sometimes accepted. This prevents trapping in a local minima.

The cover coefficient clustering method (C^3M) proposed by Can and Ozkarahan uses the cover coefficient concept (CC) [Can and Ozkarahan 1990]. CC indicates the relationship among the records (or terms) based on a two-stage probability experiment. The retrieval experiments show that the effectiveness of C^3M is compatible with the complete linkage method, where the complete linkage method is known to have a good performance. However, the complete linkage method can only be used by some approximations in very large databases due to high memory and computation time overhead. The incremental version of C^3M provides efficient processing time for dynamic environments [Can 1993].

There are some differences between the methods described above, i.e. record clustering, and the clustering of signatures for bit slice query evaluation. In bit slice query evaluation each record is represented with only one bit in a block, which means very many records may be assigned to the same block. For example, a 1K block can hold 8,192 records while with a record length of 100 bytes it can hold only 10 records. Some experiments were performed to measure the affect of various block sizes on the performance [Jakobsson 1980]. Only block sizes 5, 10 and 20 were used in experiments. The results were found quite similar and only the results for 20 records per block were reported. Also, Yu et al. did not investigate the effect of change in the number of records in a block. There is a very large difference in the number of records which can be placed in a block. Consequently, the effect of this property will be inspected.

In bit slice query evaluation only conjunctive queries are considered. (Queries containing disjunctive parts must be expressed in disjunctive normal form. Each part containing only conjunctions are processed and the results are merged.) This restriction makes the clustering of records easy. Assume two queries ($q_1 = t_1$, $q_2 = t_1$ *and* t_2) are submitted. Also, assume that the records are placed to the blocks such that the records containing t_1 are distributed in a few blocks, i.e., the records are clustered with respect to t_1 (t_1 is called clustering term). Although the records containing t_2 are distributed randomly, the number of on-blocks at the result of q_2 will be less than or equal to the number of on-blocks at the result of q_1.

Assuming that the 80-20 rule holds [Knuth 1975, Faloutsos and Jagadish 1991], if only frequently-used terms (about 20% of all terms) are used as clustering terms, for the majority of one-word queries (about 80% of all queries) there will be a few on-blocks at the result. Due to the discussion presented above, the probability of obtaining a random on-bit distribution at the result will decrease as the number of terms in the query increases. For a three term query, if we assume that query terms

are independent of each other, the probability of none of the terms being a clustering term will be $0.2^3 = 0.008$.

The records will be clustered by considering only frequently-used terms. The objective function is minimizing B as given in equation 32, where k is the number of clustering terms and b_i stands for $block_i$.

$$B = \sum_{j=1}^{k} \left| \left\{ b_i \,|\, t_j \in record \wedge record \in b_i \right\} \right| \qquad (32)$$

5.3 Storing Bit-Map of On-Blocks for Discriminating Terms

The improvement methods presented above do not require extra storage space. By storing some information in memory the performance of INCBIT can be improved drastically. As shown before, the majority of the total on-blocks are produced in the first nine steps (see Figure 9). If the resulting on-blocks were known at the first step, most of the on-blocks produced in the first steps would have been eliminated. Storing bit maps of on-blocks for clustering terms in memory provides this information.

Let us assume a one word query ($q_1 = t_1$ and t_1 is a clustering term) is submitted, and after clustering, the number of blocks containing the records with the term t_1 is three (the total number of blocks, n, is equal to 1,221). Since the resulting on-blocks are known at the first step, all of the on-bits in other blocks are false drops and they will be eliminated at future steps. Therefore, the number of on-blocks at $step_1$ will be three, instead of $n \cdot e(fd_1, m, n)$. Also, the number of on-blocks at future steps will be three.

Bit slice representation eliminates most of the bit slices by considering only on-bits of the query signature. This provides a column-wise masking on the signature file. The proposed method introduces a row-wise masking, which is orthogonal to the masking introduced by bit slice computation. As a result of this double masking on the signature file, the incremental query evaluation will require fewer block accesses independently of the number of terms used in the query.

If a query contains more than one clustering term, all of the on-block maps for the clustering terms are ANDed. The on-bits at the result block map shows the on-blocks. Therefore, as the number of terms in the query increase; the number of on-blocks at the first step decrease. Thus, the performance of the proposed method improves. In inverted file organization, the cost of the query evaluation increases as the number of terms increases. This makes the proposed method promising for queries with multiple terms.

5.4 Generating Sample Data for Experiments

For the proposed methods, performance and space overhead estimation formulas will be derived. Expected values of the performance can be obtained by substituting constant values in the formulas. To verify these formulas, the performance of the proposed methods must be measured via experiments which reflect real life environment. The most important factor that can affect the results is the test data.

One of the methods to obtain the test data will be generating random records which produces an unrealistic approximation to the real applications. There will be similar records in a database, and these similar records will contain similar terms. Trying to obtain this complex relationship with randomly generated data may not give accurate and acceptable results. Therefore, reliable test data of sufficient volume must be obtained to conduct the experiments.

For instance, the Bilkent University Library collection database contains more than 150,000 bibliographic MARC (MAchine Readable Cataloging) records. This data may be inflated by permuting the records. To prevent generating unusual permutations, only the records with the same LC (Library of Congress) class letters will be permuted.

In MARC records information is classified and assigned a three digit tag number; for example the title of a book is designated the tag number 245. Two randomly chosen records with the same LC class letters, called parent records, will be merged to obtain a new record. The new record will contain the tags which exist in both of the parent records, and additionally some random fraction of the tags which exist in only one of its parents. The information in the same tag numbers will be merged; then, a new tag information will be created by randomly selecting half of the terms from the merged term list.

6. CONCLUSION

Signature files provide efficient retrieval of formatted and unformatted data with a small space overhead. For large databases, vertical partitioning of signature files avoids reading the useless portions of the object signatures, thus they increase the performance, and provide desirable response times for user queries. For very large databases even one slice of a vertical partition may occupy several disk blocks. Due to the increase in the amount of data to be read, time required to evaluate search queries increases.

An incremental query evaluation method is proposed which dynamically avoids unnecessary disk blocks by utilizing the conjunctive nature of the queries. Without any pre-computation and additional space overhead, the proposed method eliminates 40% of the data accessed by standard bit sliced query evaluation.

The main idea of the proposed method can be summarized as follows. At the beginning of the query evaluation, all of the records are candidates to be relevant to the query. If optimality condition is satisfied, half of the bits of one bit slice of a signature file is on-bit. Consequently, after processing the first bit of the query signature, half of the records are eliminated, i.e., they are definitely not relevant to the query. As more bits from the query signature are processed, the number of eliminated records increases. In that sense, the query evaluation process can be considered as a false drop elimination process. The records, whose bits from the same vertical partition are in the same block, are logically connected to each other. If all of the records in a block are eliminated, this block is not needed in the future steps.

The contributions of this paper include

- An incremental query evaluation method which reduces the amount of the data to be read in vertically partitioned signature files for query evaluation is designed,
- The formulas to estimate the expected amount of data reduction in the proposed method are derived,
- A stopping condition for partial evaluation of the queries is defined,
- Finally, three methods to enhance the performance of the proposed method are described.

As future work, performance estimation formulas for the proposed performance improvement methods will be derived, and the results of the formulas will be validated by experimental methods and extended to frame sliced signature files.

REFERENCES

AKTUG, D., and CAN, F. 1993. Signature files: An integrated access method for formatted and unformatted databases. *Working Paper #93-006 May*, System Analysis Department, Miami University Oxford, OH 45056.

BLAIR, D. C. 1990. *Language and Representation in Information Retrieval.* Elsevier, Netherlands.

CAN, F., OZKARAHAN, E. A. 1990. Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases. *ACM Transactions on Database Systems*, 15, 4 (Dec.), 483-517.

CAN, F. 1993. Incremental clustering for dynamic information processing. *ACM Transactions on Information Systems.* 11,2 (Apr.), 143-164.

CHANG, J. W., YOO, J. S., KIM, M. H., and LEE, Y. J. 1993. A signature-based hybrid access scheme for text databases. In International Symposium on Next Generation Database Systems and their Applications, Fukuoka, Japan, Sep.

CHRISTODOULAKIS, S., and FALOUTSOS, C. 1984. Design considerations for a message file server. *IEEE Trans. on Software Engineering,* 10, 2 (March), 201-210.

DATE, C. J. 1990. *An Introduction to Database Systems, 5th edition.* Addison Wesley, Reading, MA.

DEPPISH, U. 1986. S-tree: A dynamic balanced signature index for office retrieval. In *Proceedings of the 9th International ACM-SIGIR Conference on Research and Development in Information Retrieval* (Pisa, Sept.), ACM, N.Y., 77-87.

DOUGLAS, P. M., and STEPHANIE, W. H. 1989. The constituent object parser: Syntactic structure matching for information retrieval. *ACM Transactions on Information Systems*, 7, 3 (July), 292-316.

FALOUTSOS, C. 1985a. Access methods for text. *Computing Surveys*, 17, 1 (March), 49-74.

FALOUTSOS, C. 1985b. Signature files: Design and performance comparison of some signature extraction methods. In *Proceedings of the ACM SIGMOD Conference*, Austin, Tex., May, New York.

FALOUTSOS, C. 1988. Signature files: An integrated access method for text and attributes, suitable for optical disk storage. *BIT.* 28, 4, 736-754.

FALOUTSOS, C., and 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.

FALOUTSOS, C., and CHRISTODOULAKIS, S. 1985. Design of a signature file method that accounts for non-uniform occurrence and query frequencies. In *Proceedings of the 11th International Conference on VLDB* (August 1985). VLDB Endowment, 165-170.

FALOUTSOS, C., and JAGADISH, H. V., 1991. Hybrid index organizations for text databases, *CS-TR-2621, Computer Science, Univ. of Maryland, 1991.*

HASKIN, R. L. 1981. Special-purpose processors for text retrieval. *Database Engineering*, 4, 1 (Sept.), 16-29.

JAKOBSSON, M., 1980. Reducing block accesses in inverted files by partial clustering. *Inform. Systems* 5, 1-5.

KNUTH, D. E. 1975. *The Art of Computer Programming. vol. 3: Sorting and Searching*. Addison-Wesley, Reading, Mass.

LARSON, P.A. 1983. A method for speeding up text retrieval. In *Proceedings of ACM SIGMOD Conference*, May, San Jose, CA.

LEE, D. L., and LENG, C. W. 1989. Partitioned signature files: Design issues and performance evaluation. *ACM Transactions on Information Systems*. 7, 2 (Apr.), 158-180.

LIN, Z., and FALOUTSOS, C. 1992. Frame-sliced signature files. *IEEE Transactions on Knowledge and Data Engineering*. 4, 3 (June), 281-289

MCERLEAN, F. J., BELL, D. A., MCLEAN, S. I., 1990. The use of simulated annealing for clustering data in databases. *Information Systems* 15, 2, 233-245.

RABBITI, F., and SAVINO, P. 1991. Image query processing on multilevel signatures. In *Proceedings of the 14th Annual International ACM-SIGIR Conference* (Chicago, Illinois, September), ACM, New York, 305-314.

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

SACKS-DAVIS, R. KENT, A., RAMAMOHANARAO, K. 1987. Performance of multikey access method based on descriptors superimposed coding techniques. *Information Systems.* 10, 4, 391-403.

SALTON, G. 1975. A Theory of Indexing. *Regional Conference Series in Applied Mathematics*, Philadelphia, PA.

SALTON, G. 1989. *Automatic Text Processing: The Transformation Analysis, and Retrieval of Information by Computer.* Addison Wesley, Reading, MA.

SALTON, G. and BUCKLEY 1988. Term-weight approaches in automatic text retrieval. Information Processing and Management. 24,5 (May), 513-523.

SALTON, G. and MCGILL, M. J. 1983. *Introduction to Modern Information Retrieval*. McGraw-Hill.

THARP, A. L. 1988. *File Organization and Processing*. John Wiley and Sons, New York, N.Y.

TSICHRITZIS, D and CHRISTODOULAKIS, S. 1983. Message files. *ACM Trans. on Office Information Systems*, 1, 1 (Jan.), 88-98.

OMIECINSKI, E., SCHEUERMANN, P., 1983. A global approach to record clustering and file reorganization. *Tech. Rep., Dept. of EECS, Northwestern Univ.,* Dec.

ULLMAN, J. D. 1988. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, Rockville, Maryland

VAN RIJSBERGEN, C. J. 1979. *Information Retrieval, 2nd edition.* Butterwortths, London.

WILLET, P. 1988. Recent trends in hierarchic document clustering: A critical review. *Information Processing and Management*, 24, 5, 577-597.

WHANG , K. Y., WIEDERHOLD, G., and SAGALOWICZ, D. 1983. Estimating block accesses in database organizations: A closed noniterative formula. *Commun. ACM 26, 11* (Nov.), 940-944.

YAO, S. B. 1977. Approximating block accesses in database organizations. *Commun. ACM 20, 4* (Apr.), 260-261.

YU, C. T., SUEN, CHEING-MEI, LAM, K., SIU, M. K. 1985. Adaptive record clustering. *ACM Transaction on Database Systems,* 10, 2 (June) , 180-204.

YU, C. T., SIU, M. K., LAM, K., TAI, F. 1981 Adaptive clustering schemes: General framework. *IEEE COMPAC ,* Nov., 81-89.

ZEZULA, P, and RABITTI, F. 1991. Dynamic partitioning of signature files. *ACM Transaction on Information Systems*. 9, 4 (Oct.), 336-367.

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.