PARTIAL QUERY EVALUATION FOR VERTICALLY PARTITIONED SIGNATURE FILES IN VERY LARGE UNFORMATTED DATABASES

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Seyit KOÇBERBER

January 1996

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Assoc. Prof. Dr. Fazlı CAN (Principal Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Prof. Dr. M. Erol ARKUN (Co-Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Prof. Dr. Asuman DOGAÇ

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Assoc. Prof. Dr. Mustafa AKGÜL

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Assist. Prof. Dr. David DAVENPORT

Approved for the Institute of Engineering and Science

Prof. Dr. Mehmet BARAY
Director of Institute of Engineering and Science

ABSTRACT

PARTIAL QUERY EVALUATION FOR VERTICALLY PARTITIONED SIGNATURE FILES IN VERY LARGE UNFORMATTED DATABASES

Seyit KOÇBERBER
Ph.D. in Computer Engineering and Information Science
Supervisor: Assoc. Prof. Dr. Fazlı CAN
January 1996, 131 Pages

Signature file approach is a well-known indexing technique. The Bit Sliced Signature File (BSSF) method provides an efficient retrieval environment by only accessing on-bits of query signatures. However, its response time increases for increasing number of query terms. For BSSF we define a stopping condition that tries to avoid the processing of all on-bits of query signatures through partial evaluation. The aim of the stopping condition is to reduce the expected number of false drops to the level that also provides the lowest response time within the framework of BSSF. We propose the Partially evaluated Bit-Sliced Signature File (P-BSSF) method that employs the partial evaluation strategy and minimizes the response time in a multi-term query environment by considering the submission probabilities of the queries with different number of terms. Experiments show that P-BSSF provides 85% improvement in response time over BSSF depending on space overhead and the number of query terms.

To provide better optimization of the signature file parameters in multi-term query environments, we propose Multi-Fragmented Signature File (MFSF) method as an extension of P-BSSF. In MFSF, a signature file is divided into variable sized vertical fragments with different on-bit densities to optimize the response time using a similar query evaluation methodology. In query evaluation the query signature on-bits of the lower on-bit density fragments are used first. As the number of query terms increases, the number of query signature on-bits in the lower on-bit density fragments increases and the query stopping condition is reached in fewer bit slice evaluation steps. Therefore, in MFSF, the response time decreases for an increasing number of query terms. The analysis shows that, with no space overhead, MFSF outperforms the P-BSSF and generalized frame-sliced signature file organizations.

Due to hashing and superimposition operations used in obtaining signatures, the signature of an irrelevant record may match the query signature, i.e., it is possible to have false drops. In signature file processing the accurate estimation of the number of false drops is essential to obtain system parameters that will yield a desirable response time. We propose a more accurate false drop estimation method for databases with varying record lengths instead of using an average number of distinct terms per record. In this method, the records of a database are conceptually partitioned according to the number of distinct terms they contain and the number of false drops of each group is estimated separately. Experiments with real data show that depending on the space overhead, the proposed method obtains up to 33%, 25%, and 20% response time improvements for the sequential, generalized frame-sliced, and MFSF methods, respectively.

For very large databases even one bit slice of MFSF may occupy several disk blocks. We propose the Compressed Multi-Fragmented Signature File (C-MFSF) method that stores the bit slices of MFSF in a compact form which provides a better response time. To minimize the number of disk accesses, the signature size and the disk block size can be adjusted such that most bit slices fit into a single disk block after compression. In such environments, C-MFSF evaluates the queries with more than two terms with only one disk access per query term rather than two disk accesses of the inverted file method which are respectively for the pointer of the query term posting list and the list itself. Our projection based on real data shows that for a database of one million records C-MFSF provides a response time of 0.85 seconds.

Keywords: Information Retrieval, Signature Files, Vertically Partitioned Signature Files, Compression.

ÖZET

ÇOK BÜYÜK KALIPSIZ VERİTABANLARINDA
DİKEY DİLİMLENMİŞ İMZA KÜTÜKLERİ İLE
KISMİ SORGU HESABI

Seyit KOÇBERBER
Bilgisayar ve Enformatik Mühendisliği Doktora
Tez Yöneticisi: Doçent Dr. Fazlı CAN
Ocak 1996, 131 Sayfa

İmza kütükleri sorgulara uygun olmayan kayıtların çoğunu eleyerek kalıplı ve kalıpsız bilgi kütüklerine randımanlı bir şekilde erişimi sağlar. İkil Dilimlenmiş İmza Kütükleri (İDİK) yöntemi okunacak ve işlenecek bilgi miktarını azaltarak randımanı daha da artırır. Fakat, artan sorgu sözcüklerinin daha fazla ikil dilim işlenmesini gerektirmesi İDİK yönteminin sorguya yanıt süresini artırır. Tez kapsamında, İDİK yöntemi için sorgu hesabının imza kütüğü işleme safhasını sorgu imzasının bütün "1" ikillerini kullanmadan tamamlamaya çalışan bir durma koşulu tanımlandı. Durma koşulunun amacı, beklenen yanlışlıkla uyan kayıt sayısını ikil dilimlenmiş imza kütükleri için en düşük yanıt süresini sağlayacak düzeye indirmektir. Bu kısmi hesap stratejisini kullanan Kısmi hesaplanan İkil Dilimlenmiş İmza Kütükleri (K-İDİK) yöntemi önerildi. K-İDİK durma koşulu ile birlikte çok sözcüklü sorgu ortamlarında farklı sayıda sözcük içeren sorguların sunulma olasılıklarını da gözleyerek sorgu yanıt süresini enküçük düzeyine indirir. Deneyler K-İDİK'in kullanılan disk alanı ve sorgu sözcük sayısına bağlı olarak İDİK'e göre yanıt süresinde yüzde 85 iyileşme sağladığını göstermektedir.

Çok sözcüklü sorgu ortamlarında imza kütüğü parametrelerinin daha da eniyileştirilmesini sağlamak için, K-İDİK yönteminin geliştirilmişi olan, Çok Kısımlı İmza Kütüğü (ÇKİK) yöntemi önerildi. ÇKİK yöteminde kısmi hesap stratejisini kullanarak yanıt süresini eniyileştirmek amacıyla imza kütüğü herbiri farklı "1" yoğunluğuna sahip değişik büyüklükte dikey kısımlara ayrılmıştır. Sorgu hesabında düşük “1” yoğunluklu kısımlara ait sorgu imzası “1” leri öncelikle kullanılır. Sorgulardaki sözcük sayısı artarken düşük “1” yoğunluğuna sahip kısımlarda bulunan sorgu imzası “1” lerinin sayısı da artar. Böylece durma koşuluna daha az hesaplama adımı ile erişilir ve dolayısıyla artan sorgu sözcük sayısı için ÇKİK’in sorgu yanıt süresi azalır. Analizler ek bir disk alanı gerektirmeden ÇKİK’in K-İDİK ve genellenmiş çerçeve dilimli imza kütüğü yöntemlerinden daha iyi sonuçlar verdiğini göstermektedir.

İmza üretiminde kullanılan, sözcüklerden rasgele ikil konumu elde etme ve üst üste bindirme işlemleri nedeniyle sorguya uygun olmayan bir kaydın imzası sorgu imzasına uygun olabilmektedir. Bu türden kayıtlara yanlışlıkla uyan kayıt denir. İstenir bir yanıt süresi elde edebilmek için yanlışlıkla uyan kayıtların sayısının doğru kestirimi gereklidir. Değişken sayıda farklı sözcük içeren kayıtlardan oluşan veritabanlarında ortalama farklı sözcük sayısı kullanmak yerine yanlışlıkla uyan kayıtların sayısını daha doğru kestiren bir yöntem önerildi. Önerilen yöntemde veritabanındaki kayıtlar içerdikleri farklı sözcük sayılarına göre kavramsal kısımlara ayrılır ve her kısımdaki yanlışlıkla uyan kayıt sayısı ayrı ayrı kestirilir. Gerçek veri ile yapılan deneylerde kullanılan disk alanına bağlı olarak sıradan erişimli, genellenmiş çerçeve dilimli ve ÇKİK yöntemleri için yüzde 33’e, yüzde 25’e ve yüzde 20’ye kadar varan sorgu yanıt süresi iyileştirmeleri elde edilmiştir.

Çok büyük veritabanlarında ÇKİK’in bir ikil dilimi bile birçok disk bloğunu kapsayabilmektedir. ÇKİK’in ikil dilimlerini daha yoğun olarak saklayan Sıkıştırılmış Çok Kısımlı İmza Kütüğü (S-ÇKİK) yöntemi önerildi. ÇKİK’in seyrek “1” içeren ikil dilimlerinin sıkıştırılması daha düşük sorgu yanıt süreleri sağlamaktadır. Disk erişim sayısını eniyileştirmek için disk blok büyüklüğü ikil dilimlerin çoğunun sıkıştırma işleminden sonra bir disk bloğuna sığmasını sağlayacak biçimde ayarlanabilir. Böyle ortamlarda S-ÇKİK ikiden fazla terim içeren sorguları sözcük başına bir disk erişimi ile hesaplayabilmektedir. Aynı ortamlarda tersyüz edilmiş kütükler ise biri sorgu sözcüğünü aramak diğeri de sorgu sözcüğüne ait kayıt listesine erişmek için olmak üzere iki disk erişimine gereksinim duymaktadır.

Açar Sözcükler: Bilgi Erişimi, İmza Kütükleri, Dikey Kısımlanmış İmza Kütükleri, Sıkıştırma.

ACKNOWLEDGEMENT

I would like to acknowledge the valuable help and guidance of Dr. Fazlı Can throughout the development of this thesis. I have always found him ready to help me when I needed. I would also like to thank Prof. Dr. M. Erol Arkun in his efforts to support me. I owe a great debt of thanks to my lovely family for their encouragement and understanding. I also thank to my mother who came to help and took care of me during this hard work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Relational database systems, by utilizing set theoretic operations, provide a theoretical and practical storage and retrieval environment for formatted data [DAT90, ULL88]. In these systems, indexes on frequently used attributes provide efficient retrieval of desired information. Similarly, unformatted data (image, voice, text, etc.) can be stored in variable length data fields. (For simplicity, an instance of any kind of data, i.e., data items stored in the database, will be referred to as record in the rest of this thesis.) However, searching unformatted data in tables of a relational database system is inefficient. Therefore, efficient file structures and search techniques must be developed for purely or partially unformatted database records [AKT93a, CAN85, CAN93, FAL92, KÖK79, SAL89, VAN79].

For search and retrieval purposes, unformatted data is described by a set of descriptors (attributes) [DOU89, RAB91, SAL75, SAL88]. 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 and each record may have different number of terms [SAL75, SAL83b]. In this thesis "term" is used to mean a descriptor and the method used to obtain the terms of a record is not our concern.

Information retrieval (IR) methods for unformatted data can be classified into two groups: *logical information retrieval* and *physical information retrieval* [BLA90].

The aim of logical information retrieval is to answer the following questions.

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

These questions are related to the meaning of the contents of the records and the information need of the user submitting the query.

For logical information retrieval *effectiveness* of IR is important. The effectiveness of an IR system can be measured by the degree at which the information needs of users are satisfied. There are two common measures of effectiveness: *recall* and *precision*. Recall is the ratio of the number of relevant and retrieved records to total number of relevant records. Precision is the ratio of the number of relevant and retrieved records to the total number of retrieved records. An effective IR system must only retrieve relevant records; however, this is difficult if not impossible, since there is no exact method to represent records and queries.

In IR the word relevance does not have a well defined meaning [BLA90]. The users may determine the number of relevant records in the set of records retrieved by the system. However, determining the total number of relevant records to a particular query is a difficult task. Therefore, artificially created environments and small databases are used for measuring the effectiveness of the IR models.

In physical information retrieval, the terms used to describe a record are assumed to be the exact representation of the record. Similarly, user queries consisting of terms are also assumed to be the exact representation of the desired information. The aim of physical information retrieval is to find the matching records to the user query by using minimum system resources. Therefore, the physical information retrieval deals with the *efficiency* of an IR system. The basic measures of efficiency are the *response time*, i.e., time required to answer user queries, and the *disk space* used by the IR system. The efficiency of the update operations is usually secondary, but also important. In this thesis only the physical meaning of retrieval and relevant is our concern.

1.1 File Structures for Information Retrieval

IR systems show variations due to the nature of the records in their databases, the frequency and types of the operations performed, and the properties of the auxiliary storage devices used. For example, data written to a write once disk becomes permanent. The retrieval method of an application that uses write once disks should have appropriate file structures to overcome this difficulty for insertion operations. Various file structures have been proposed in the literature that try to obtain a better performance by considering the properties of IR environments. Some of them are briefly introduced below.

Sequential files: in this structure records are stored sequentially without using any additional data structures. Therefore, insertion of records are easy and there is no space overhead. However, to find the relevant records to a query, all of the records must be read and compared with the search query. Therefore, in sequential files, the retrieval speed is proportional to the number of records. If the queries can be processed in batches, one pass over the records will be sufficient to answer many queries. Sequential files may be preferred for small databases or for the environments where the prompt system response is not crucial.

Inverted files: in this structure to find the relevant records to a term easily, a pre-computed list of documents which contain the term is stored with each term [SAL83b, WIT94, HAR92]. Usually, the pre-computed list of documents is called the *concordance* or the *posting list.* To find the relevant records to a term, first the location of the posting list of the term is obtained, and then the posting list is read. Usually, 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 current, extra computation is required for insertion and updates of the records.

Signature files: in this structure to provide a space efficient fast search structure, each term is hashed into a bit string which is called term signature [AKT93a, FAL85b, FAL92]. 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. To find the relevant records to 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. The signature file acts as a filter and eliminates most of the irrelevant records to a query without retrieving actual records.

Clustered files: in this structure similar records are grouped into clusters and to retrieve relevant records to a query, the query is compared with the representatives of the clusters, known as cluster centroids [CAN90, WIL88]. The clustering hypothesis, which states that “closely associated records tend to be relevant to the same request,” is the justification of the clustering methods [VAN79]. This application of clustering provides a logical IR system. Clustering similar records and assigning the records in the same cluster to the same disk block or close to each other also improves the performance of the physical retrieval methods [OMI90].

1.2 Signature Files as a Physical Retrieval Method

In signature approach, each term is hashed into S positions among F positions where $F > S$. The result is called a term signature. Usually, a signature with F positions is represented with a bit string of length F and each term sets the bits to “1” (on-bit) in the positions it has been hashed (compressed signatures may require less than F bits). In this thesis, unless otherwise stated, a signature with F positions will be represented with a bit string of length F and we will use the signature size to define both the number of bits used to represent the signature and the number of positions that can be hashed.

Record signatures are obtained either by concatenating or superimposing the signatures of the record terms. These record signatures are stored in a separate file, the *signature file*, which reflects the contents of database records. In superimposed signature files, the length of the record signature (F) and term signatures are the same and $F >> S$. In this thesis, we consider only vertically partitioned superimposed signatures (will be defined later in this section) and conjunctive queries, i.e., ANDed terms.

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

Due to hashing and superimposition operations used in obtaining signatures, the signature of an irrelevant record may match the query signature. These records are called *false drops*. The false drop probability is minimized when the *optimality condition* is satisfied, i.e., half of a record signature bits are on-bits [CHR84, ROB79]. In the second phase of the query processing, these possible false drop records are resolved (if necessary) by accessing the actual records [AKT93a, FAL92, KOÇ95a LIN92, ROB79, SAC87]. The description of the query processing with signature files is depicted in Figure 1.1.

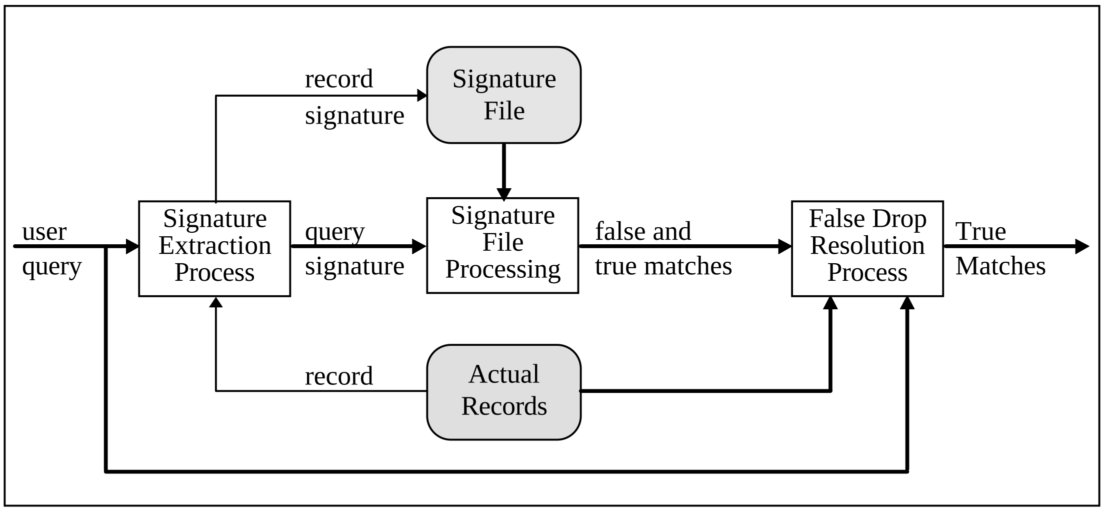


Figure 1.1. Description of query processing with signature files.
(Lines that are active during query processing are boldfaced.)

For a database of N records, the signature file can be viewed as an N by F bit matrix. Sequential Signature Files (SSF) require retrieval and processing of all $N \cdot F$ bits in the signature file. However, off-bits of a query signature have no effect on the result of the query processing, since only the on-bits of the query signature are compared with the corresponding record signature bits. Therefore, the result of the signature file processing can be obtained by processing only the record signature bits corresponding to the on-bits of a query signature.

To retrieve the record signature bits corresponding to a bit position without retrieving other bits, the signature file is vertically partitioned and the bits of a vertical partition are stored sequentially as in bit-sliced signature files (BSSF) [ROB79] and generalized frame-sliced signature files (GFSSF) [LIN92]. Vertical partitioning a signature file improves performance by reducing the amount of data to be read and processed.

1.3 Scope of the Work and Contributions

In BSSF, to satisfy the optimality condition, the number of bits set by each term (*S*) is adjusted according to the signature size (*F*) and the average number of terms in a record (D_{avg}) without considering the number of query terms. For increasing number of query terms the number of on-bits in a query signature increases. Consequently, the time required to complete the first phase of the query evaluation increases [ROB79]. The Generalized Frame Sliced Signature File method (GFSSF) proposed in [LIN92] attacks this problem by adjusting the value of *S* such that the response time becomes minimum for a given number of query terms, *t*. However, in a multi-term query environment, queries containing less than *t* terms will obtain many false drops. Also, the queries with more than *t* terms will unnecessarily process many bit slices.

In multi-media environments, search conditions on various media are expressed in a single query [ZEZ91] which cause an increase in the number of query terms. Therefore, the access method of such an environment should provide acceptable response times for high number of query terms. At the same time, a general purpose access method should also provide acceptable response times for queries containing a few query terms. BSSF and GFSSF do not satisfy these requirements.

Bit-sliced signature files and inverted files have some common properties but they are different methods. However, the differences have not been defined clearly. First we provide a clarification of this.

We propose a new signature file optimization method, Partially evaluated Bit-Sliced Signature File (P-BSSF), which combines optimal selection of *S* value that minimizes the response time with a partial evaluation strategy in a multi-term query environment. The partial evaluation strategy uses a subset of the on-bits of a query signature. During the selection of the optimal *S* value, we considered the submission probabilities of the queries with various number of terms. Therefore, P-BSSF adjusts the trade off between fewer slice processing and resolving more false drops properly and increases the performance.

The stopping condition defined for P-BSSF improves the system performance by processing a limited number of bit slices. To further improve the performance of P-BSSF we propose a new signature file organization and query evaluation method, Multi-Fragmented Signature File (MFSF). In MFSF, the signature file matrix is divided into variable sized vertical fragments. Each fragment is a conceptual BSSF with its own *F* and *S* parameters and each term sets bit(s) in each fragment. Therefore, each fragment may have a different on-bit density (the ratio of the number of on-bits to total number of bits). For query evaluation, the bit slices from the lowest on-bit density fragments are used first. Therefore, as the number of query terms increases, the number of bit slices used from the fragments with lower on-bit density increases. Lower on-bit density eliminates false drops more rapidly and the stopping condition is reached in fewer bit slice evaluations. Therefore, MFSF provides decreasing response time for increasing numbers of query terms.

Experiments with real data reveal that assuming the existence of the same average number of terms per record, D_{avg}, causes some error for the estimation of number of false drop records (FD). We propose a more accurate false drop estimation method, the Partitioned False Drop estimation method (PFD), for the databases with varying number of distinct terms in the records. In PFD, we conceptually divide the records of a database into disjoint partitions according to the number of distinct terms in the records. Each partition is considered as a separate signature file and average number of distinct terms in a partition is used to estimate FD in this partition. The PFD method decreases the differences among the numbers of distinct terms in the records of a partition. Therefore, FD is estimated more accurately. FD affects the performance of a signature file method since these false drop records must be resolved by accessing actual records. Accurate estimation of FD enables better estimation of the signature file parameters to obtain a better response time.

Lower on-bit density in a vertically partitioned signature file method provides reaching the stopping condition in fewer evaluation steps. However, to obtain a lower on-bit density the signature size (*F*) must be increased which results in increased space overhead. To increase the performance without increasing the space overhead we propose Compressed Multi-Fragmented Signature File (C-MFSF) method that extends the MFSF method. In C-MFSF, we compress the sparse bit slices of MFSF for large *F* values. Usually, reading a compressed bit slices of C-MFSF requires a few disk block accesses even for very large databases. Additionally, since the on-bit density is reduced the stopping condition is reached by processing fewer number of bit slices.

1.4 Organization of the Thesis

The rest of the thesis is organized as follows.

In Chapter 2 the previous work on inverted files and signature files is summarized. After this we discuss the distinguishing features of the vertically partitioned signature files and inverted files. Thereby we clarify the features that can be used for the distinction of these two methods.

Chapter 3 provides the definition of the performance measures and the description of our test environment, i.e., the test database, test queries, and relevant attributes of the computer used in the experiments. Additionally, to estimate the performance of signature file methods, the operations involved in query processing with signature files are modeled.

In Chapter 4 and Chapter 5, we describe the P-BSSF and the MFSF methods, respectively. Also, we provide the results obtained by simulations and experiments with real data.

We present the Partitioned False drop Estimation method (PFD) in Chapter 6. The use of PFD in SSF, GFSSF and MFSF are provided along with the results obtained with real data.

In Chapter 7, the Compressed Multi-Fragmented Signature File method is presented. The C-MFSF method is compared with the compressed inverted file method.

Chapter 8 contains the conclusion and the contributions of the thesis and pointers for future research.

We provide the definitions of frequently used acronyms and important symbols used in the equations in Appendices A and B, respectively. Appendix C provides the hashing algorithm used to map terms to their signature. This appendix is provided for reproducibility of the results, since this algorithm slightly affects the results. Appendix C provides the list of stop words.

2. INVERTED FILES AND SIGNATURE FILES

An IR system stores records and provides search and retrieval of these records via descriptors which we call terms. The set of terms used to describe the records of a database is called *vocabulary* or dictionary. The records may have different number of terms [SAL75]. The terms may have different importance in describing different records. Usually, term importance, called term weight, is represented with a real number [SAL83a, SAL88]. In this thesis, for easy association with signature files, binary term weights are assumed. This corresponds to the existence or absence of a term in the record description. An example text database is given in Figure 2.1. The example database contains five records (R_1, R_2, R_3, R_4, R_5) and the vocabulary contains six terms (T_1, T_2, T_3, T_4, T_5, T_6).

Vocabulary	**Records**
T_1 = access	R1 = { computer, information }
T_2 = computer	R2 = { access }
T_3 = database	R3 = { information, retrieval }
T_4 = information	R4 = { signature }
T_5 = retrieval	R5 = { computer, database }
T_6 = signature	(N = 5, V = 6)

Figure 2.1. Example text database.
(N = no. of records, V = no. of terms.)

In the rest of this chapter, we describe Inverted Files (IF) and summarize previous work on Signature Files (SF). Bit-sliced signature files and inverted files have some common properties but they are different methods. However, what makes them different has not been defined clearly in the literature. We provide the answer of this question in Section 2.3.

2.1 Inverted Files

In the inverted file method each distinct term is associated with a list of identifiers, called posting list or concordance, of the records that contain the term [SAL83b]. Usually, the record identifiers are the numbers of the records that contain the term. The vocabulary is organized as a lookup table to access the terms easily.

To obtain the records containing a particular term, first the term is found in the lookup table and then corresponding posting list is read. Also, a Record Pointer Table (RPT) must be stored to obtain the physical addresses of the records corresponding to the record numbers in the posting lists. Inverted file representation of the example database is given in Figure 2.2.

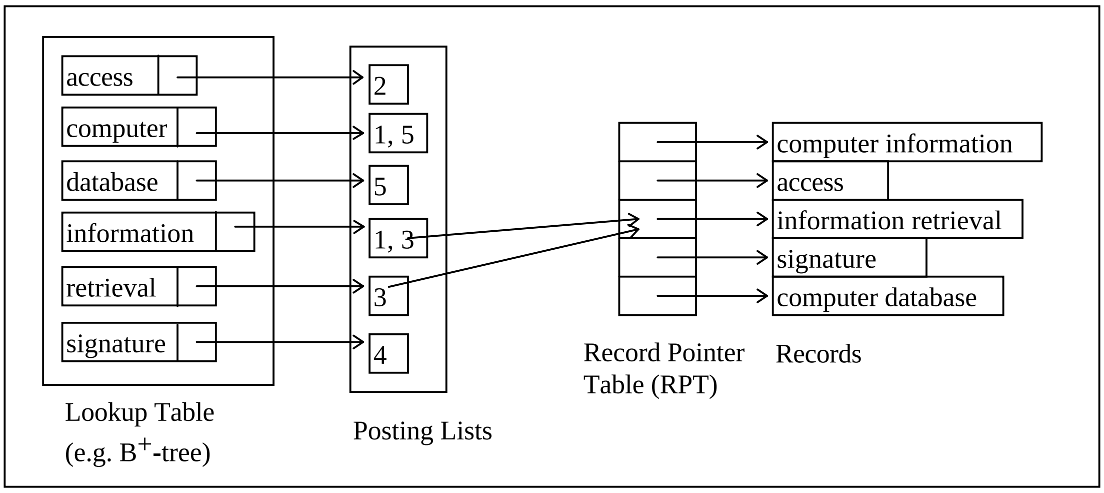


Figure 2.2. Inverted file representation of example database shown in Figure 2.1.

In a dynamic environment there will be new records added to the database. (In our discussion deletions, which are rare in IR systems, are ignored.) To insert a new record to the database the posting lists of the terms occurring in the new record are read, the record identifier is added to these posting lists, and the updated posting lists are written back to the disk. Additionally, the new record may contain new terms which requires expanding the vocabulary by updating the lookup table. Therefore, insertion of new records are costly in the inverted file method.

Another difficulty in a dynamic environment is the maintenance of the posting lists. The posting lists will get longer as new records are added to the database. The space required to add new record identifiers to posting lists can be supplied by either chaining the new record identifiers or allocating some free space at the end of the posting lists. Usually, the posting lists are stored on auxiliary storage and retrieving a chained posting list will require many disk accesses. Therefore, chaining must be avoided if possible. Reserving some free space at the end of the posting lists will increase the space overhead since there will be many posting lists. Also, the reserved free space may be insufficient for some posting lists and either a reorganization or chaining additional free space may be required.

Our presentation so far describes the most conventional implementation of the inverted file approach. However, the same logical structure can be implemented in various ways. Therefore, in the thesis the phrase “inverted files” covers different implementations of the inverted file concept as illustrated in Figure 2.2.

2.1.1 Query Evaluation with Inverted Files

To evaluate a query with inverted files the posting lists of the query terms must be retrieved. Usually, the lookup table is maintained using a B^+-tree and one disk access will be sufficient to obtain the address of a posting list if the interior nodes (non leaf nodes) of the B^+-tree held in memory. Since the branching factor is very large in a B^+-tree, the number of interior nodes will be small and this assumption can be satisfied [SALZ88].

After obtaining the address of a posting list, it can be retrieved with one disk access if the associated blocks are stored contiguously on the disk. These requirements can be satisfied easily for static databases. However, as we mentioned above, the posting lists of a dynamic database may contain chains. Therefore, traversing these chains will require additional disk accesses.

For the conjunctive queries containing many terms, to decrease the query evaluation time the posting lists of the query terms may be sorted in increasing (more correctly non decreasing) posting list lengths and may be processed in this order [MOF95a]. Since the size of the result list will be less than or equal to the size of the shortest posting list, the memory requirements will be minimized. Also, the number of matching record identifiers in the intermediate steps will be minimized. Another strategy for processing conjunctive queries which may decrease the query evaluation time is processing a subset of the query terms. After reducing the number of candidates (possible answers) to a small number, the query evaluation with the inverted file may be completed without processing remaining query terms. The candidate records are checked to eliminate the possible false matches (i.e., the records which do not contain some of the remaining unused query terms) [ZOB92].

2.1.2 Bit maps

The terms of a record can be represented with a bit vector of size *V* (vocabulary size) containing one bit for each entry in the vocabulary. A one-to-one mapping function generates the bit position of a given term. The bit vector of R3 of Figure 2.1 is "000110". The occurrence of terms in the records can be represented with a Binary Record-Term Matrix (BRTM). The BRTM of a database with *N* records and *V* distinct terms in the vocabulary will contain *N* rows and *V* columns. The BRTM of the example text database given in Figure 2.1 is shown in Figure 2.3. One column of BRTM is called *bit map*. The bit map of the term "information," T_4, is "10100" and the "1"s in the first and third bit positions indicate that the records R_1 and R_3 contain the term "information."

	T_1	T_2	T_3	T_4	T_5	T_6
R_1	0	1	0	1	0	0
R_2	1	0	0	0	0	0
R_3	0	0	0	1	1	0
R_4	0	0	0	0	0	1
R_5	0	1	1	0	0	0

Figure 2.3. Binary record-term matrix (BRTM) for the example database of Figure 2.1.

The size of a bit map depends on the number of records in the database. For example, the size of a bit map for a database containing 10^6 records will be 122 Kbytes. If this database contains 100,000 distinct terms the BRTM of this database will occupy 11.64 GBytes which is very high.

The posting lists of common terms may be longer than their bit map representation since they will contain many record identifiers. Therefore, instead of these longer posting lists the corresponding bit map representations may be stored. This will save space and processing time since bit maps can be merged efficiently by bitwise operations.

The method proposed by Faloutsos and Jagadish uses the posting list storage for rare terms and bit map storage for frequent terms [FAL91]. Faloutsos and Jagadish proposed different organizations for using the bit map in different environments. The proposed method maintains the lookup table for all terms. Therefore, the time required to search the lookup table is the same as other inversion methods. Also, the space overhead generated by the lookup table is not eliminated.

2.1.3 Compressed Inverted Files

Inverted files require an additional memory of 50%-300% of the original records depending on the detail of the stored information [HAS81, FAL85a, MOF95a]. However, recent studies show that by compression this space overhead can be reduced to less than 10% of the space used by the original records [ZOB92, WIT94]. This reduction can be obtained if only conjunctive queries or basic ranking are supported. Basic ranking techniques require database level statistics about the terms to estimate the term importance [SAL83b]. If better ranking and word sequence queries are supported the index (i.e., vocabulary and posting list) requires 25% of the space used by the actual data [ZOB92].

If the posting lists are compressed, insertion of new records becomes complex and database creation may be expensive. Also, there is some possibility of a bottleneck during decoding the compressed posting lists [ZOB92]. Adding skips, an index on the entries of a posting list, provides substantial time savings with a small overhead to the compressed inverted file entries [MOF95a].

2.2 Signature Files

One of the factors affecting the space overhead of an inverted file system is the number of distinct terms in the database, i.e., the number of entries in the vocabulary. Signature files eliminate the need for a vocabulary and save the space and time required to search the lookup table.

The signature of a term is obtained by hashing the term to a predetermined number of locations (S) among a given set of hashing locations (F). We provide an algorithm to generate term signatures in Figure 2.4. In this algorithm, the random number generator used in obtaining the bit positions from the term, may produce the same bit positions more than once. Consequently, some term signatures may contain less than *S* "1"s. This can be avoided by ignoring the bit positions that have been already set to "1".

A particular term is always hashed to the same location(s). However, this is an irreversible process, i.e., the term can not be obtained from the locations the term has been hashed unless all terms in the dictionary are hashed and compared with these locations. Sometimes, even this inefficient sequential search may be caught short to obtain the term from the given hashing locations. Depending on the total number of hashing locations, the number of locations a term is hashed, and the hashing function that maps the term to a number, more than one term may be hashed to the same locations with a non-zero probability. Therefore, a term signature is an abstraction of the term and may not contain all of the information about the term. In [FAL87b] the relation between the false drop probability and the information loss is inspected.

```
**Algorithm GenerateTermSignature**
Set all bit positions of the term signature equal to "0".
$r \leftarrow$ Map the term into a number using a hashing function.
$i \leftarrow 1$.
while $i \le S$
  { $BitPosition \leftarrow \max(1, F \cdot random(r))$ .
    Set $BitPosition$ of term signature equal to "1".
    $i \leftarrow i + 1$.
  }
```

The function *random* is a random number generator that returns
a random number in the range [0,1] and resets the value of its argument *r*.

Figure 2.4. Generation of a term signature.

Although the definition of a signature has no restriction for the representation of a term signature, usually a term signature with *F* locations is represented with a bit string of length *F*, since the bit strings can be processed efficiently by available computers. We distinguish the *size of a signature* from the value of *F* and we define the signature size as the number of bits required to store a record signature. Also, we define *F* as the upper bound for the hashing function used to determine the bit positions set by the terms. Note that the signature size and the value of *F* may be different. However, in most of the signature file applications, the signature size and the value of F are the same, i.e., they use uncompressed bit string representation.

Another point we want to clarify is that the values of S and F can be selected freely. In the definition of a term signature there is no limit for the values of F and S. except $S < F$. However, the retrieval methods may limit the values of F and S because of efficiency considerations.

The signature file approach contains some uncertainty due to the hashing operation used in generating a term signature. Due to this uncertainty, the result of a query evaluation may produce false matches (*false drops*), i.e. 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 [FAL87a].

$$fd = \frac{\text{Number of false matches}}{\text{Number of records which do not qualify the query}} \tag{2.1}$$

2.2.1 Record Signature Generation Methods

Record signatures are obtained from the signatures of terms they contain. There are two basic record signature generation methods: word signature and superimposed coding.

2.2.1.1 Word Signature

In *word signatures* (WS), a record signature is obtained by concatenating the signatures of the non common words (terms) of the record (see Figure 2.5) [TSI83]. Generally, the length of the word signatures are the same for all terms. This preserves the positional information present in the original record.

Application of WS in unformatted databases produces variable length record signatures. Therefore, a record is divided into blocks that contain the same number of distinct terms [FAL85b]. The false drop probability of WS for single term queries is computed as follows [FAL85b].

$$fd = 1 - (1 - \frac{1}{S_{\max}})^D \tag{2.2}$$

where D is the number of distinct terms in a block and S_{max} is maximum possible number of distinct term signatures (i.e., S_{max} can be smaller than V). For large S_{max} and small D values the false drop probability can be computed as

$$fd \approx \frac{D}{S_{\max}}. \tag{2.3}$$

Terms	**Term Signatures**
computer	1 1 0 0
signature	0 0 1 0
retrieval	1 0 1 1

R = (computer, signature, retrieval)

Record signature for R : 1 1 0 0   0 0 1 0   1 0 1 1

Figure 2.5. Record signature generation using word signatures.

Ramamohanarao et al. used WS to generate record signatures for formatted databases, i.e., for fixed length records. A block signature is obtained by superimposing the signatures of the records stored in the disk block. The block signatures are stored separately and the qualifying blocks are accessed during query evaluation [RAM83].

To the best of our knowledge, the only storage and search method proposed to use WS in unformatted databases is the sequential storage and search method. Therefore, since the query processing requires retrieval of the whole signature file for sequential storage, known WS methods are unsuitable for large databases.

The record signature generation with WS encodes the content of a record into bit patterns. As we mentioned before the terms may not be obtained uniquely from the term signatures. In that sense the WS method acts as a lossy compression method that do not require reconstruction of original record for query processing. Considerable space savings can be obtained by compressing the records of a text database [BEL93]. Therefore, the efficiency of WS must be compared with the text compression methods.

2.2.1.2 Superimposed Coding

In *superimposed coding* (SC) a record signature is obtained by superimposing, bitwise ORing, the term signatures of the record terms [KNU75]. A term signature can also be generated by superimposing the signatures of its overlapping n-grams [FAL85b]. However, this is out of concern. In SC the number of hashing positions (F_{SC}) is very large compared to the number of hashing positions in WS (F_{WS}). If a block contains b terms in WS, to obtain the same space overhead for SC, the value of F_{SC} must be $b \cdot F_{WS}$. Consequently, the number of possible term signatures is very large in SC and the false drop probability incurred due to producing the same term signature for more than one term is negligible.

To answer a query, first a query signature is produced using query terms. Then, this query signature is compared with the record signatures. If a record contains all of the query terms, i.e., if the record is relevant to the query, the record signature will have on-bits in the corresponding bit positions of all on-bits of the query signature. Therefore, the records whose signatures contain at least one “0” bit (off-bit) in the corresponding positions of on-bits of the query signature are definitely irrelevant to the query. This is the first phase of query processing with superimposed signature files.

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

Figure 2.6. Signature generation and query processing with superimposed signatures.

To illustrate signature generation and query processing with superimposed signatures an example is provided in Figure 2.6. Query signature on-bits shown in bold font have a “0” bit at the corresponding record signature positions. Since the 1st and 7th bits of the record signature are “0” while the signature of the query “retrieval” has “1” at these positions, the record is irrelevant to this query. The record signature matches the signatures of the queries “access” and “information”. The on-bit positions set by the query term “access” (2nd, 6th, and 10th) are also set by the record terms “computer” and “information” (2nd, 5th, 6th, 8th, 9th, and 10th).

Therefore, although the record does not contain the term “access”, the record seems to qualify the query (a false drop).

In SC false drops are mainly produced due to the superimposition operation used to obtain record signatures. Although all terms may be 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, in the second phase of a query evaluation with SC, the false drops which pass the filtering process must be eliminated by accessing the actual records. This process is called the *false drop resolution*.

The false drop probability for SC is examined in [CHR84a, ROB79] and the authors show that to obtain minimum false drop probability half of a record signature bits must be on-bit (the optimality condition). The optimality condition requires selecting a specific *S* value for given *F* and *D* values. The false drop probability of SC for single term queries is computed as follows [ROB79].

$$fd = \left(1 - (1 - \frac{S}{F})^D\right)^S \tag{2.4}$$

Equation (2.4) can be explained as follows. Since each term sets *S* bits to “1” in a bit string that is *F* bit long, the probability of a particular bit of a term signature being “1” is $\frac{S}{F}$. By negating this probability, we obtain $(1 - \frac{S}{F})$, i.e., the probability of leaving a particular bit of the term signature as “0”. There are *D* terms that set bits in a record signature. Therefore, the probability of a particular bit of a record signature being “0” is $(1 - \frac{S}{F})^D$. By negating this probability, we obtain the probability of a particular bit of a record signature being an on-bit (on-bit density). Note that this probability is the probability of a particular bit position of the record signature set to “1” accidentally. Therefore, Equation (2.4) gives the probability of finding “1”s in *S* randomly selected bit positions in the record signature. Since the signature of a single term query contains *S* on-bits, Equation (2.4) gives the false drop probability, i.e., the probability of matching the signature of an irrelevant record and a single-term query signature accidentally.

2.2.1.3 Considering Varying Number of Record Terms

The records of an unformatted database contain different number of distinct terms. The signatures of the records containing many terms will contain more “1”s than the optimality condition requires. This increases the false drop probability. Faloutsos and Christodoulakis suggest dividing a record into blocks that contain equal number of distinct terms and producing a separate signature for each block [FAL88a]. However, the numbers of “1”s in record signatures expose a normal distribution and there may be record signatures containing non-optimal number of “1”s. Leng and Lee call this method Fixed Size Block (FSB) method and they propose the Fixed Weight Block (FWB) method as an alternative [LEN92]. In FWB, instead of controlling the number of terms in a block, the number of “1”s in a block signature is controlled [LEN92].

Dividing a long record into blocks obtains lower false drop probabilities [FAL88a, LEN92]. However, record level search and retrieval operations become complex. For example, the terms of a record that is relevant to a multi-term conjunctive query may be distributed to more than one block. Therefore, for a multi-term query, to determine the relevance of a record all block signatures of a record must be compared with the query signature and the matching query terms must be monitored.

Usually, only single term queries and records containing fixed number of terms are considered in false drop analysis and performance estimations for signature files. This creates an artificial test environment since the records of an unformatted database contain varying numbers of terms and user queries may contain more than one term in real IR applications. As illustrated in Chapter 3 there is no such simplifying assumptions in our test environment. Therefore, the results obtained in our test environment can also be obtained in real applications.

2.2.1.4 Considering Term Occurrence and Query Frequencies

To reduce the false drop probability, there are various proposals that accounts the importance of terms in the queries and frequencies of the terms in signature generation [FAL85c, FAL87a, FAL88a, LEN92, AKT93b]. These methods let the terms set different number of bits according to the importance of the term. The importance of the terms are determined by inspecting the database occurrence frequencies and query frequencies of the terms. However, a lookup table is needed to find the number of bits set by a particular term.

2.2.2 Compressing Record Signatures

If the optimality condition is satisfied, the number of “1”s and “0”s are equal in a record signature. The optimum storage method of such a bit vector is storing it as a bit string. Each on-bit is represented with the expense of two bits. However, storing a record signature with *F* hashing locations as a bit string of the same size is not a requirement for all signature file methods. The record signatures containing considerably less number of “1”s than “0”s (or reverse) can be compressed [FAL85b].

The false drop probability can be reduced by increasing the value of *F* (see Equation 2.4). Note that since the optimality condition is violated, the false drop probability obtained by increasing *F* will be greater than the minimum false drop probability that can be obtained if the optimality condition is satisfied using the optimum *S* value for the larger *F* value. In [FAL85b] Faloutsos proposed the idea of using a large *F* with $S = 1$ and compressing the resulting sparse record signature. In the same work he inspects the Run Length encoding (RL), bit-Block Compression (BC), and Variable bit-Block Compression (VBC) methods and shows that the RL method obtains a lower false drop probability than WS, SC, BC, and VBC methods.

2.2.3 Signature File Organization Methods

Sequential storage of the signature file requires processing of all record signatures for a query evaluation. The time required to retrieve and process all record signatures increases as the number of records in the database increases. To obtain acceptable response times for large databases, various signature file organization methods are proposed. The basic motivation of these methods is processing not all but a part of the signature file for query evaluation.

2.2.3.1 Vertically Partitioned Signature Files

Vertical partitioning methods utilize the fact that only on-bits of a query signature affect the result of a query processing. These methods divide the signature file into vertical partitions and retrieve only required partitions for query evaluation. Vertical partitioning improves performance of query processing; however, insertion operations become expensive.

2.2.3.1.1 Bit-Sliced Signature Files

In Bit-Sliced Signature Files (BSSF) the signature file is stored in column-wise order [ROB79]. For query evaluation only the bit slices corresponding to the “1”s in the query signature are retrieved. To evaluate a single term query, *S* bit slices (at most) must be retrieved and processed, as opposed to retrieving only one posting list 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 posting list corresponding to the query term. This requires a lookup table search.

2.2.3.1.2 Frame Sliced Signature Files

In frame sliced signature files (FSSF) the record signature is divided into *k* equal sized frames and the signatures are stored in a frame-wise fashion. 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 [LIN92]. 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 some of the “0” bits of the term signature are also transferred.

In the generalized version of FSSF, each word sets bits in n frames (GFSSF) [LIN92]. 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.

2.2.3.2 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. The 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.

2.2.3.2.1 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 record signatures and identify a part of them as the records keys [LEE89]. Record signatures are partitioned according to their key value. The key of the query signatures are extracted in the same way, and only those 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. [ZEZ91]. 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 key with all “0”s, 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 [AKT93b]. The results show that letting high discriminatory terms (typically characterized by low document frequency coupled with high query frequency) to set more bits than low discriminatory terms increases the performance of LHSS. The effect of multi-term queries are inspected as well.

2.2.3.2.2 Multi Level Methods

One typical implementation of the multi level methods, the signature tree approach, 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 [THA88]. 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 same idea prevails in [PFA80] where upper levels of signatures (called the block descriptors) are created by superimposing a group of the lower level signatures that are assigned to a block. The number of signatures from level i that are superimposed to form the block descriptor at level (i+1), where ($i \geq 1$), is called the packing factor $p(i)$ which is a design parameter and may vary for different levels of the tree. The structure is called indexed descriptor files.

The S-tree method proposed by Deppish dynamically groups similar signatures during insertion [DEP86]. 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 [SAC87] 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.

2.3 The Differences Between Bit-Sliced Signature Files and Inverted Files

Inverted files and signature files try to find the list of relevant records to a query within a desirable response time. Especially the BSSF method has some common conceptual properties with the inverted files. However, the properties which make these two methods different are unclear in the IR literature. The rest of this chapter provides the necessary explanation.

The BRTM of a database resembles a signature file. For simplicity, we will assume that inverted file methods maintain a BRTM using bit maps. Since the posting lists and bit maps are two different forms of storing one column of the BRTM, the following discussion is also true for the posting list storage method.

The basic characteristic of an IF method is storing the vocabulary in a lookup table. The lookup table is needed to obtain the address of the bit map corresponding to a term during query processing. For insertion of a new record, the address of the bit map will be used to set the bit corresponding to the new record in the BRTM. The aim of storing a lookup table is determining existence or absence of a term in the vocabulary with 100% certainty and obtaining corresponding bit map address. If these requirements (i.e., term lookup table and posting lists) are fulfilled the method used in implementation of the lookup table will not be a distinguishing criteria for inverted file and signature file approaches. For example, the lookup table may be implemented as a hash table or a B^+-tree.

Using a hash table for the lookup table requires hashing each term to a table position (bit string) which is conceptually similar to a term signature. The hashing function may produce the same hash table position (bit pattern) for different terms. This condition is called collision. Various additional data structures and algorithms may be used to resolve the collisions [KNU75]. These data structure may lead to storing the original terms in a linked list or overflow buckets. Anyway, at the end of the search process the hash table method will decide the existence or absence of a term with 100% certainty and will obtain the address of the bit map associated to the term if the term exist in the dictionary. Another possibility is the use of a perfect hashing function with no collisions [FOX91].

In summary, the basic property of an inverted file method is that each term of a record sets only one bit in a conceptual BRTM and the bit position set by a term is never set by another term. (Note that we are using the phrase “conceptual BRTM” since BRTM is never stored as is for large databases due to its excessive storage requirement.) This one-to-one correspondence between a document term and the corresponding bit in BRTM is guaranteed by the certainty in searching the lookup table.

In signature file methods each term may be hashed to more than one bit position to set bits as opposed to a single bit position of the IF methods. In the extreme case, a signature file method may decide to set only one bit for each term like in the inverted file method. Therefore, signature file methods are more general than IF methods in the selection of the *S* value. For example, in [FAL85b] Faloutsos uses a large *F* value and each term is hashed to only one position. Since the record signatures will be sparse, he proposes compressing the record signatures. The proposed compression methods provide efficient usage of the storage space and hence implies a lower false drop probability for a given space overhead. However, the compression is an additional operation over signature generation and neither the compression is a necessity for $S = 1$ nor the compression can only be used for large *F* and $S = 1$.

The difference between an inverted file method and a bit-sliced signature file method starts when more than one term is hashed to the same bit position (this is the synonym of a collision of the hash table implementation of the inverted file method). Signature file methods try to minimize the collisions since they produce false drops. However, they use no vocabulary or additional data structures to resolve the collision. Instead, conceptually in signature files the posting lists of the terms that were hashed to the same bit position are merged in the corresponding bit slice. Therefore, a bit slice may contain the posting list of more than one term where a bit map corresponds to exactly one posting list.

In summary, in signature files there is a non-zero probability of a bit position set by a term is also set by other terms. Therefore, the bit slices of a BSSF are like the posting lists of more than one term. The expected number of terms that were hashed to the same bit position depends on the value of *F* and *S*. As a result, the query evaluation with signature files may produce false drops which are the most important characteristic of SF methods. On the other hand, in the IF methods, each bit map belongs to only one term hence there is no false drops.

The storage structure of posting lists and bit slices cannot be used to distinguish inverted and signature file methods. The aim of both methods is to store the data in the most convenient way. As we mentioned before, the posting lists are more compact representations of sparse columns of the BRTM. However, the posting list representation may not be feasible for all types of terms. For frequent terms, storing a bit map may be more feasible than storing a posting list [FAL91] . Similarly, if it is more compact, a bit slice of a BSSF may be stored similar to posting lists of an IF.

The methods proposed by Faloutsos and Chan demonstrate the distinction between a BSSF and an IF [FAL88b]. The Compressed Bit Slices (CBS) method proposed in [FAL88] is a BSSF method. In CBS each term sets only one bit in a large *F* and the sparse bit slices are compressed.

2.4 Hybrid Methods

The horizontal and vertical signature partitioning methods can be combined to obtain a hybrid method that exploits the desirable characteristics of both approaches [GRA92]; however, in this section our concern is the combination of SF and IF methods.

The hybrid access method proposed by Chang et al. stores the vocabulary in a lookup table that uses an inverted index structure [CHA93]. A block posting file is stored for the primary terms (they assume that about 20% of the terms will receive about 80% of the user interest). For the secondary terms, remaining 80% of the terms, a block signature file is used to reduce the space overhead. The block posting file and the block signature file point the blocks of a record signature file that contains record signatures. In the record signature file, similar record signatures are clustered to improve the performance of the system.

The compressed bit slices of the CBS method are in variable lengths. To access the bit slices a sparse pointer file is needed (note that some bit slices may contain all zeros). If the value of *F* is decreased to avoid the sparcity, there will be many false drops. Therefore, Faloutsos and Chan propose the Doubly Compressed Bit Slices (DCBS) method [FAL88b]. In DCBS, 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 bit pattern (term signature) which is shorter than the term itself is stored in a bucket which was chained if an overflow occurs. This additional data structure resembles storing the terms mapping to the same location in a hash table.

The proposed structure, DCBS, is no longer a BSSF, instead it is in between an IF method with a hash table and a BSSF. Practically, the gain in the space overhead using the signature produced by the second hash function instead of actual term will be small (note that terms may be compressed). To retrieve the bit slice of a term, the term is hashed to a location in the hash table and the first intermediate bucket pointed by the hashing location is read. The address of the bit slice is obtained from the intermediate buckets and the bit slice is retrieved with a second disk access [FAL88b]. Note that the intermediate buckets may be chained and additional disk accesses may be required.

If a B^+-tree is used to store the dictionary there will be two disk accesses. Additionally, there are a non-zero false drop probability for DCBS. False drops may be generated if both of the hashing functions produce the same signatures.

Faloutsos and Chan propose the No False Drop (NFD) method to solve the false drop problem completely [FAL88b]. In NFD a pointer is added to each entry that points the term in the original record. The resulting data structure is no longer a signature file. Instead it is an inverted file that use a term signature in addition to a hash table to search the dictionary.

3. PERFORMANCE MEASURE AND TEST ENVIRONMENT

To estimate the performance of the proposed methods a simulation and test environment is designed. The values of the parameters used in the simulation runs are determined experimentally and reflect a real computing environment. This provides validation of the results obtained by simulation runs in experiments with real data. Figure 3.1 provides a pictorial description of our experimental environment. In short, we use the traditional scientific experiment approach. However, this does not imply that we repeat all experiments for both (simulation and real) cases. We reiterate the experiments that are essential for the validation of the models.

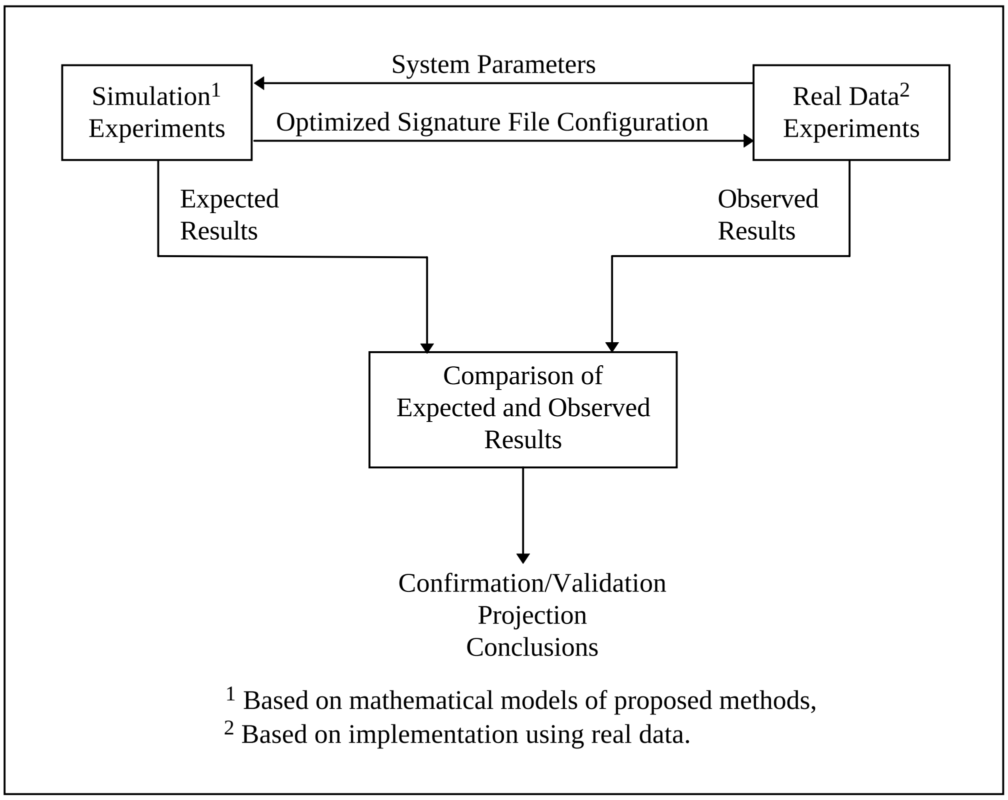

System Parameters
Simulation[1] Experiments
Optimized Signature File Configuration
Real Data[2] Experiments
Expected Results
Observed Results
Comparison of Expected and Observed Results
Confirmation/Validation
Projection
Conclusions
[1] Based on mathematical models of proposed methods,
[2] Based on implementation using real data.

Figure 3.1. Description of test environment.

This chapter is organized as follows. In Section 3.1, the performance measure used to compare the inspected methods is defined. The properties of the test database

(BLISS-1) and the description of the computing (system) environment used in the experiments are provided in Sections 3.2 and 3.3, respectively. In the following section, the method used to model multi-term query environments is described and the probability distributions of the test queries are described. Finally, Section 3.5 contains the formulas used to model the query processing operations.

3.1 Performance Measure

Various performance measures are used in the literature. Some of them are the number of seek operations [KEN90], the signature reduction ratio (the ratio of the number of signatures searched to the total number of signatures in the signature file) [LEE89], the computation reduction ratio [LEE90, LEE95], and the response time [LIN92, ROB79, SAL89]. Some of these measures are not applicable to all indexing methods. For example, the signature reduction ratio is meaningless for the inverted file method. Consequently, there may be difficulties in the performance comparisons of different methods if a common performance measure is not used. Since the primary goal of all physical information retrieval methods is to obtain a desirable response time, we used the response time as the performance measure. In this way, we can compare the performance of a new method with any other indexing method and estimate its performance in real life.

The number of false drop records or the false drop probability may be used as a comparison criteria among the signature file methods [CHR84a]. However, usually, the false drop probability and the number of false drops are affected by the work done in signature file processing phase. A method may obtain lower FD values by spending more time in signature file processing phase. Consequently, unless all methods spend the same processing time for the signature file processing phase, using FD as a comparison criteria will be misleading.

The *response time* is defined as the time required to process the signature file, resolve all false drop records (if any), and find the first relevant record to the query as defined in [LIN88, LIN92]. We use the same response time definition in this thesis. This definition of response time obtains a query instance independent performance measure. If all relevant records to a query are accessed before responding to a query, the response time become query instance dependent. Generally, information retrieval systems display the first screen of the relevant records to a query. Remaining records are retrieved in groups upon user requests. Therefore, the definition of response time coincides with real applications.

We used the *improvement percentage* (IP) value in the comparison of the performance of the methods we inspected. The improvement percentage provided by method A with respect to method B, IP(A,B), is defined as

$$IP(A,B) = 100 \cdot (TR(A)\ -\ TR(B)) / TR(A) \tag{3.1}$$

where TR(A) and TR(B) are the response times obtained by A and B, respectively.

3.2 Test Database: BLISS-1

We used MARC (MAchine Readable Cataloging) records of the Bilkent University Library collection as the test database (BLISS-1). MARC records are widely used to store and distribute the bibliographic information about various types of materials such as books, films, slides, videotapes, etc. Also, MARC records are basic record structure of many library systems such as Melvyl and OCLC [FOX93]. Additionally, other researchers can obtain MARC records easily for test and comparison purposes.

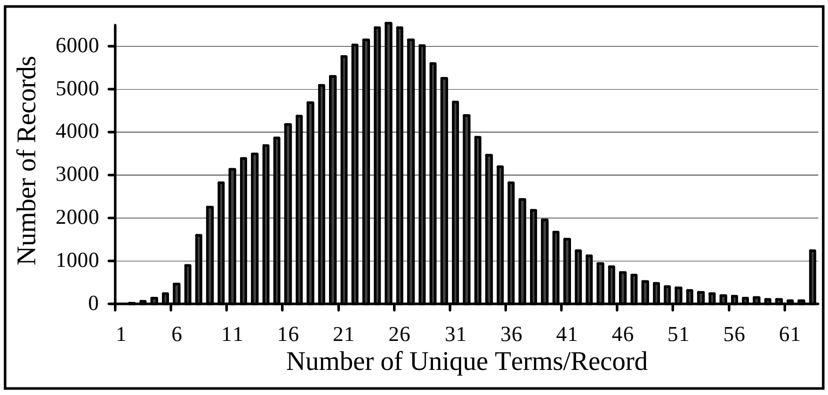


Figure 3.2. Distribution of the numbers of unique terms
in the records of the test database BLISS-1.

BLISS-1 contains 152,850 MARC records with varying lengths. The largest record contains 166 distinct terms while the average number of distinct terms per record is 25.7 (the stop words given in Appendix D are removed). We plotted the distribution of number of terms in records of BLISS-1 in Figure 3.2. (The last bar represents the number of records containing more than 62 terms.) The number of unique terms in the records of the test database expose a normal distribution. One of the most widely used measure of dispersion for such distributions is the standard deviation and for BLISS-1 the standard deviation for the “number of unique terms per record” is 11.12.

In BLISS-1, MARC records are aligned according to disk block boundaries such that reading of each record during false drop resolution requires only one disk block access (RB = 1) unless the MARC record is larger than a disk block. This alignment increases the size of the data file by 4.34%. Record statistics of BLISS-1 are given in Table 3.1.

Table 3.1. Record Statistics of the Test Database BLISS-1

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

Table 3.2 provides the test database sizes of some other signature file studies. Although BLISS-1 can be considered as a medium size database, it is relatively large compared to other test databases. (According to our definition, a database with 10^6 or more records is very large.) Furthermore, our test database size uses the BLISS (Bilkent Library Information Services System) records. We prefer to use records of a real application rather than artificially generating large databases. In artificial databases, the properties of real applications, such as the similarities between the records and the distribution of the terms to records, may be established improperly. To obtain the performance of the proposed methods for very large databases, usually, we assume 10^6 records in mathematical analysis. Also, we project the results obtained with BLISS-1 for larger databases in Section 7.5.

Table 3.2. Size of Some Test Databases Used so Far

No. of Records	Artificial/Real	Reference
10,000	Artificial	ZEZ91
28,000	Semi Artificial*	LIN92
98,732	Real	ZOB95
100,000	Artificial	LEE89
150,000	Real	SAC87, KEN90

* The same 2800 real records are repeated 10 times.

3.3 Computing Environment

A 33 MHz, 486 DX personal computer with a hard disk of 360 Mbyte running under DOS 5.0 is used to test the performance of the proposed method. We prefer to use the DOS environment since it provides exclusive control of all resources. Non-interrupting execution of user programs provides accurate measure of the response time and produces consistent and reproducible results. Physical layout of a signature file on the disk affects the time required to process the signature file and physical layout of a file on the disk can be controlled in the DOS environment. We provide the values of system parameters in Table 3.3. The values related with disk operations and processor operations are determined experimentally.

Table 3.3. System Parameter Values of the Computing Environment

B_{size}, size of a disk block (bytes)	8192
P_{size}, size of a record pointer (bytes)	4
T_{byteop}, time required to perform bit operations between two bytes (milliseconds, ms)	0.00127
T_{read}, time required to read a disk block (ms)	5.77
T_{scan}, average time required to match an actual record with a query for false drop resolution (ms)	4.5
T_{seek}, average time required to position the read head of disk to the desired block (includes rotational latency time) (ms)	30
T_{wordop}, time required to perform bit operations between two memory words (ms)	0.00098
W_{size}, size of a memory word (bytes)	4*

* we used long integers for bit operations

We expect that a multi-user system can offer computing power and I/O speed equivalent to our experimental environment if not better. So the results of the experiments can be achieved in multi-user environments without a performance degradation.

3.4 Simulating Multi-Term Query Environments and Test Queries

We model a multi-term query environment with a bounded normal distribution from left and right. In the simulation runs and in the experiments with real data we limited the maximum number of query terms, t_{max}, to five. For the queries containing more than five terms we assume the query contains only five terms. If required, the performance of the proposed methods were inspected for other t_{max} values.

To measure the performance of the methods by the experiments with real data we considered three different query cases: Low Weight (LW), Uniform Distribution (UD), and High Weight (HW) queries. The values of P_t ($1 \le t \le 5$), where P_t denotes the probability of submitting a *t* term query, for these query cases are given in Table 3.4.

Table 3.4. P_t Values for LW, UD, and HW Query Cases

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

To use in the experiments with real data, we generated a query set containing 1000 zero hit queries randomly by considering the occurrence probabilities of the number of query terms for each query case. For example, since the occurrence probability of a one term query is 0.10 in the HW query case, the HW query set contains 100 ($0.10 \cdot 1000$) one term queries. The observed FD and response time values are obtained by taking the average of the FD and response time values obtained by each query in the query sets. Since there is no relevant record to zero hit queries, all false drop records must be accessed and; therefore, the effect of accessing these false drop records on the response time is maximum. Section 5.6.1 contains an exclusive experimental setting to further investigate the effect of number of query terms on the retrieval performance.

3.5 Modeling Query Processing Operations

To estimate the performance of the proposed methods, we modeled the operations involved in evaluating a query with signature files. The basic operation to be modeled is reading a specified amount of data from the auxiliary storage. Data are written and read in blocks and the physical layout of the data on the auxiliary storage affects the I/O time. Therefore, we incorporate the sequentiality probability, *SP*, into the estimation of the time required to read *b* logically consecutive disk blocks [LIN92]. *SP* is the probability of reading the next logically consecutive disk block without a seek operation. We estimate the time required to read *b* logically consecutive disk blocks as follows.

$$Read(d) = (1 + (b-1) \cdot (1 - SP)) \cdot T_{seek} + d \cdot T_{read} \tag{3.2}$$

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

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

$$T_{slice} = Read(\left\lceil \frac{N}{8 \cdot B_{size}} \right\rceil) + T_{wordop} \cdot \left\lceil \frac{N}{8 \cdot W_{size}} \right\rceil \tag{3.3}$$

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

Usually, data records are variable lengths and a lookup table is used to find the record pointer of the actual record. Since MARC records are variable lengths, we needed a lookup table (see RPT of Figure 2.2) and we modeled obtaining a record pointer as follows. At the database initialization stage PB record pointers, each occupying P_{size} bytes, are read into a buffer of $PB \cdot P_{size}$ bytes. Since this is a one time cost, it is excluded from the response time calculations. The probability of finding a requested record pointer in the buffer is approximately equal to PB/N. For databases with fixed length records or when all record pointers (i.e., the RPT table) are stored in main memory, PB is equal to N, i.e., the cost of finding the record pointers is zero.

For false drop resolution of a record, the record pointer is obtained, the record is read, and the record is scanned to test whether it matches the query. Therefore, the false drop resolution time for one record, $T_{resolve}$, is computed as follows.

$$T_{resolve} = (1 - PB/N) \cdot Read(\left\lceil \frac{PB \cdot P_{size}}{B_{size}} \right\rceil) + Read(RB) + T_{scan} \tag{3.4}$$

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

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

4. PARTIAL EVALUATION OF QUERIES IN BSSF

For a database of N records, the signature file can be viewed as an N by F bit matrix. For a given query Sequential Signature Files (SSF) require retrieval and processing of all $N \cdot F$ bits in the signature file. However, off-bits of a query signature have no effect on the result of the query processing, since only the on-bits of the query signature are compared with the corresponding record signature bits. Therefore, the result of the signature file processing can be obtained by processing only the record signature bits corresponding to the on-bits of a query signature.

To retrieve the record signature bits corresponding to a bit position without retrieving other bits, the signature file is vertically partitioned and the bits of a vertical partition are stored sequentially as in bit-sliced signature files (BSSF) [ROB79] and generalized frame-sliced signature files (GFSSF) [LIN92]. Vertical partitioning a signature file improves performance by reducing the amount of data to be read and processed.

In BSSF, especially for multi-term queries, the time required to complete the first phase of the query evaluation increases as the query weight increases [ROB79]. In this chapter we propose the Partially evaluated Bit-Sliced Signature File (P-BSSF) method to solve this problem. In P-BSSF the response time is minimized in a multi-term query environment by employing the partial evaluation strategy and considering the submission probabilities of the queries with different number of terms [KOÇ95b, KOÇ95c]. The technique employs a stopping condition that tries to complete the first phase of query evaluation without using all on-bits of the query signature, i.e., by *partial evaluation*. The aim of the stopping condition is to reduce the number of expected false drops to an acceptable level that will also provide the lowest response time within the framework of the bit-sliced signature file environment [KOÇ95b, KOÇ95c].

This chapter is organized as follows. The query processing with BSSF is explained with an example in Section 4.1. In Section 4.2, we derive a formula that finds the optimum signature size (space overhead) of BSSF for a given database instance. In Section 4.3, previous works that improve the performance of BSSF are summarized. In Section 4.4, the stopping condition that tries to complete the first phase of query evaluation without using all on-bits of the query signature is defined. In Section 4.5, the response time is minimized in a multi-term query environment by considering the submission probabilities of the queries with different number of terms. The results of the comparison of P-BSSF and BSSF with simulation runs are given in Section 4.6. Finally, Section 4.7 contains the results of the experiments with real data.

4.1 Query Processing with BSSF

BSSF requires retrieval of $W(Q)_t \cdot N$ bits instead of $F \cdot N$ bits where $W(Q)_t$ is the number of on-bits in the query signature (query weight) of a t term query. Usually, $W(Q)_t << F$; hence the amount of retrieved and processed data is reduced. Therefore, the response time of BSSF is less than the response time of SSF except for very small N values [ROB79].

We repeat the formulas to compute the number of on-bits in the query signature (query weight) and the expected number of false drops given in [ROB79]. The false drop probability ($fd_{W(Q)_t}$) for a t term query is computed as follows.

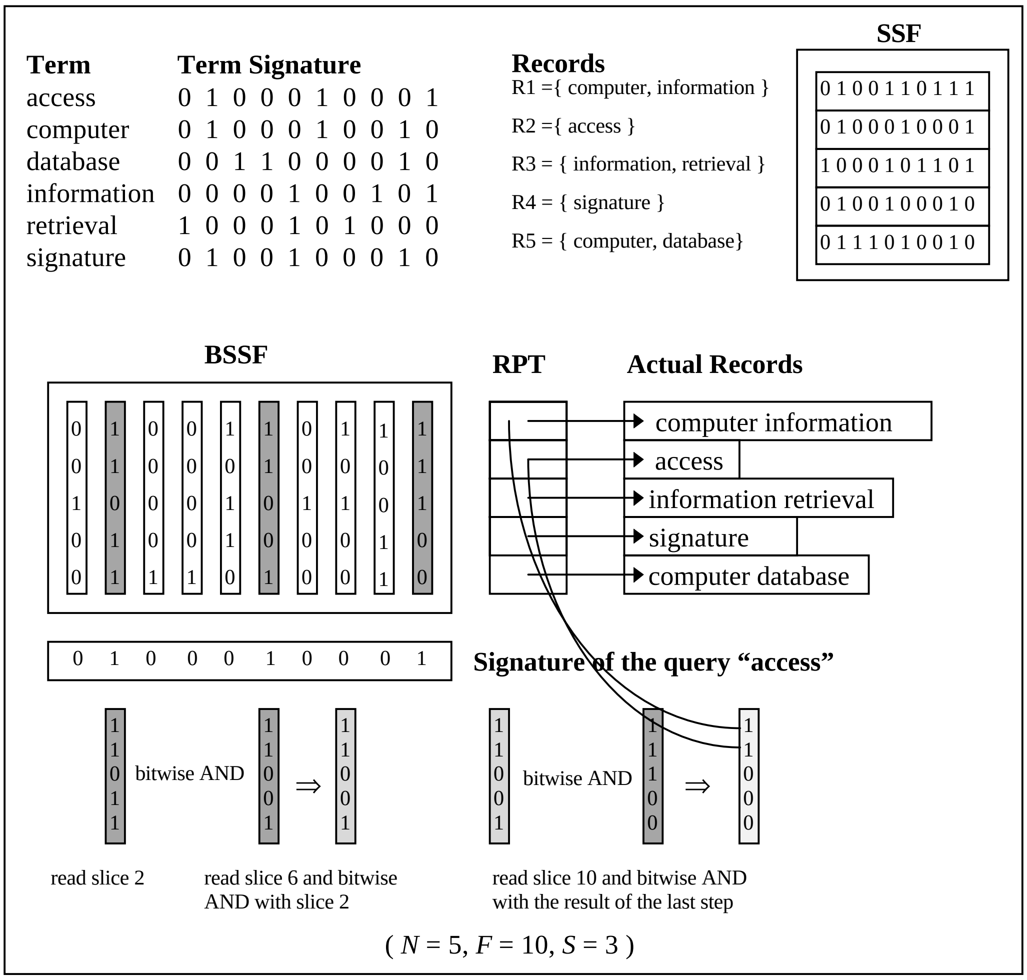
$$fd_{W(Q)_t} = (1-(1-S/F)^D)^{W(Q)_t} \tag{4.1}$$

$$W(Q)_t = F \cdot (1-(1-S/F)^t) \tag{4.2}$$

where D is the average number of terms per record and $W(Q)_t$ is the query weight of a t term query [ROB79]. S and F are design parameters. Previous works show that the false drop probability becomes minimum when the *optimality condition* is satisfied, i.e., half of the bits in a record signature are on-bits [CHR84a, ROB79]. The expected number of false drops after processing $W(Q)_t$ bit slices, $FD_{W(Q)_t}$, is proportional to the number of records in the database (N) and computed as follows.

$$FD_{W(Q)_t} = N \cdot fd_{W(Q)_t} \tag{4.3}$$

Query processing with BSSF is demonstrated in Figure 4.1. The term signatures, the records with the record signatures, and sequential storage of these record signatures are shown at the top of the figure. The bits in the horizontal boxes of SSF are stored sequentially from the left to the right.



Term
Term Signature
access 0 1 0 0 0 1 0 0 0 1
computer 0 1 0 0 0 1 0 0 1 0
database 0 0 1 1 0 0 0 0 1 0
information 0 0 0 0 1 0 0 1 0 1
retrieval 1 0 0 0 1 0 1 0 0 0
signature 0 1 0 0 1 0 0 0 1 0
Records
R1 ={ computer, information }
R2 ={ access }
R3 = { information, retrieval }
R4 = { signature }
R5 = { computer, database}
SSF
0 1 0 0 1 1 0 1 1 1
0 1 0 0 0 1 0 0 0 1
1 0 0 0 1 0 1 1 0 1
0 1 0 0 1 0 0 0 1 0
0 1 1 1 0 1 0 0 1 0
BSSF
RPT
Actual Records
computer information
access
information retrieval
signature
computer database
0 1 0 0 0 1 0 0 0 1
Signature of the query “access”
bitwise AND
⇒
read slice 2
read slice 6 and bitwise AND with slice 2
read slice 10 and bitwise AND with the result of the last step
(N = 5, F = 10, S = 3)

Figure 4.1. SSF and BSSF organizations and BSSF query processing example.

BSSF storage of the signature file is shown in the middle of Figure 4.1. The bits in the vertical boxes are stored sequentially from the top to the bottom. A record pointer table (RPT) is needed to store the addresses of the records. For SSF the associated record pointers can be stored with the record signatures.

Evaluation of the query “access” is illustrated at the bottom of Figure 4.1. To evaluate the query three bit slices (2nd, 6th, and 10th), shown with dark gray background color in BSSF, are read. The result of the signature file processing is also a bit string of length five where an on-bit indicates that the corresponding record is found relevant to the query. Only the first and second bits of the result bit string are on-bits. Therefore, the first and second record pointers are obtained by accessing RPT and then the actual (corresponding) records are read and compared with the query for false drop resolution. Since the first record does not contain the query term “access,” it is a false drop.

4.2 Space Optimization for BSSF

The query evaluation with BSSF requires processing all bit slices corresponding to the on-bits of a query signature. Since the optimality condition is satisfied in BSSF the on-bit density is always 0.5 and the false drop probability for a *t* term query can be computed as follows (see Equation (4.1)).

$$fd_{W(Q)t} = \left(\tfrac{1}{2}\right)^{F\cdot\left(1-(1-\frac{S}{F})^{t}\right)} \tag{4.4}$$

Since the value of *S* must satisfy the optimality condition for given *F* and *D* (the number of distinct terms in a record) values, the value of *S* can be expressed in terms of *F* and *D* as follows [CHR84a].

$$S = \frac{F\cdot\ln 2}{D} \tag{4.5}$$

By substituting Equation (4.5) in Equation (4.4) we obtain

$$fd_{W(Q)t} = \left(\tfrac{1}{2}\right)^{F\cdot\left(1-(1-\frac{\ln 2}{D})^{t}\right)}. \tag{4.6}$$

Equation (4.5) and (4.6) show that the query weight increases while the false drop probability decreases for increasing *F* values. However, after reducing FD to a negligible value, continuing to reduce the false drop probability by increasing *F* is meaningless since it unnecessarily requires more disk space and more bit slice processing.

To obtain the optimum space overhead (since the optimality condition is satisfied we will obtain the optimum query weight at the same time), we express the response time of BSSF for a *t* term query as a function of *F* as follows.

$$RT(F) = c\cdot F\cdot T_{slice} + N\cdot(\tfrac{1}{2})^{c\cdot F}\cdot T_{resolve} \tag{4.7}$$

where $c = 1 - (1 - \frac{\ln 2}{D})^t$. To find the *F* value which provides the minimum response time we take the derivative of Equation (4.7) with respect to *F* and we obtain

$$\frac{dRT(F)}{dF} = c \cdot T_{slice} + N \cdot T_{resolve} \cdot c \cdot \ln\tfrac{1}{2} \cdot (\tfrac{1}{2})^{c \cdot F} . \tag{4.8}$$

To find the optimum *F* value that provides minimum response time, we let Equation (4.8) equal to zero and we obtain

$$c \cdot (T_{slice} + N \cdot T_{resolve} \cdot \ln\tfrac{1}{2} \cdot (\tfrac{1}{2})^{c \cdot F}) = 0 \tag{4.9}$$

Since $D \geq 1$, *c* is always greater than zero. Therefore, we drop it and we solve the remaining part of Equation (4.9) for *F*.

$$F = \frac{\ln N + \ln \frac{T_{resolve}}{T_{slice}} + \ln \ln 2}{c \cdot \ln 2} \tag{4.10}$$

There are two important outcomes of Equation (4.10);

- The optimum *F* value decreases for increasing *t* values (for increasing *t* values *c* increases and consequently *F* decreases).
- The optimum *F* value increases for increasing *N* values.

For single term queries (*t* = 1), *c* becomes equal to $\frac{\ln 2}{D}$. By substituting this value in Equation (4.10) we obtain a special case of Equation (4.11) for single term queries as follows.

$$F = \frac{D}{(\ln 2)^2} \cdot \left(\ln N + \ln \frac{T_{resolve}}{T_{slice}} + \ln \ln 2 \right) \tag{4.11}$$

4.3 Previous Proposals to Improve the Performance of BSSF

There are previous proposals to improve the performance of BSSF. Sacks-Davis et al. [SAC87] proposed using *S* bit slices in the first phase of the query evaluation of a multi-term query without providing a formal stopping condition.

For the extended bit-sliced signature file (B’SSF) and the generalized frame-sliced signature file (GFSSF) methods Lin and Faloutsos proposed adjusting the value of *S* for a specific number of query terms, *t*, such that the response time is minimized

[LIN88, LIN92]. However, in a multi-term query environment, queries containing less than *t* terms will obtain many false drops. Also, since no stopping condition was defined, the queries with more than *t* terms will unnecessarily process many bit slices. Panagopoulos and Faloutsos defined a partial fetch policy with spooling the bit slices on a parallel machine architecture [PAN94].

Ishikawa et al. [ISH93] tried to find the optimum *S* value experimentally by measuring the response time for changing *S* values for a specific database instance without providing any formal method. For the queries containing many terms, they proposed using only randomly selected two query terms in the first phase of the query evaluation. However, the records containing the selected terms but missing some of the remaining query terms will be false drop records.

Our method, P-BSSF, combines optimal selection of *S* with a partial evaluation strategy in a multi-term query environment. The partial evaluation strategy uses a subset of the on-bits of a query signature and oversees the equal contribution of each query term to the query evaluation process until it reaches the stopping condition. During selection of the optimal *S* value, we consider the submission probabilities of the queries with various number of terms.

4.3.1 B'SSF: the Enhanced Version of BSSF

For BSSF, the optimality condition requires a larger *S* value for a larger signature size (*F*) [ROB79] (see Equation 4.5). For small *F* values, the false drop probability is high and many false drop records are obtained at the end of the signature file processing. Therefore, an increase in *F* decreases FD while it increases the query weight and the number of retrieved bit slices. In Section 4.2 we obtained a formula to compute the optimum *F* value for a given number of query terms. Increasing *F* after reaching the optimum value also increases the response time in the BSSF method.

In the B'SSF method, the optimality condition is relaxed and the response time is minimized for single term queries instead of minimizing the false drop probability [LIN88]. An optimized B'SSF configuration may have a smaller *S* value than a BSSF requires. The value of *S* decreases for increasing *F* value. Therefore, the response time of B'SSF decreases for increasing F value. The formula to find the optimum S value can be found in [LIN88].

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

4.3.2 GFSSF: Generalized Frame-Sliced Signature Files

Current auxiliary storage seek time is much larger than the read time per disk block. GFSSF provides improvement over B'SSF [LIN88] by minimizing the number of seek operations [LIN92]. GFSSF optimizes the signature file parameters for a given number of query terms.

In GFSSF, a signature is divided into k frames, each of size s bits ($s = F/k$). Each term first randomly selects n ($1 \leq n \leq k$) frames, then randomly sets m ($1 \leq m \leq s$) bits (not necessarily distinct) in each of the selected frames [LIN92]. In this method, the size of a frame is $s \cdot N$ bits and each frame is stored separately as a SSF. The methods SSF, BSSF, and B'SSF are special cases of GFSSF [LIN92].

4.4 Partial Evaluation of Queries in BSSF: P-BSSF

Our objective is to obtain the minimum response time for BSSF in a multi-term query environment. In a multi-term query environment, high-weight queries may require processing of the bit slices after reducing the expected number of false drops to a negligible value if the value of S is optimized for a lower query weight. Therefore, using a subset of the on-bits in the first phase of the query evaluation may further reduce the response time. We find an exact stopping condition which provides the minimum response time for given F, D, N, and S values.

To estimate the false drop probability in partial evaluation of the first phase, we use the *on-bit density* (*op*) which is the probability of a particular bit of a bit slice being an on-bit. Total number of on-bits in a signature file is $N \cdot F \cdot (1-(1-S/F)^D)$. Since there are $N \cdot F$ bits in the signature file, by assuming the on-bits are uniformly distributed in a record signature and there are no interdependency among the records and among the terms, the on-bit density becomes

$$op = 1 - \left(1 - \frac{S}{F}\right)^{D}. \qquad (4.12)$$

The value fd_i, the false drop probability after processing i bit slices $0 \le i \le W(Q)_t$, is computed as follows.

$$fd_i = op^i \qquad (4.13)$$

The stopping condition will minimize the query evaluation time. Therefore, for given S, t, F, D, and N values, we write the response time as a function of i, the number of bit slices used in the first phase for a t term query, as follows.

$$RT(i) = i \cdot T_{slice} + N \cdot op^i \cdot T_{resolve} \quad where\ \ 0 \le i \le W(Q)_t \qquad (4.14)$$

To find the i value for the minimum response time we take the derivative of $RT(i)$ with respect to i. The result is:

$$\frac{dRT(i)}{di} = T_{slice} + N \cdot T_{resolve} \cdot op^i \cdot \ln op \qquad (4.15)$$

To find the optimum number of evaluation steps, i, we let Equation (4.15) equal to 0 and solve it for i.

$$i = \ln\left(\frac{T_{slice}}{N \cdot T_{resolve} \cdot (-\ln op)}\right) \Big/ \ln op \qquad (4.16)$$

If reaching the stopping condition requires more on-bits than the query signature contains, i.e., $i > W(Q)_t$, i is taken as $W(Q)_t$. The on-bits used in the query evaluation are selected from the query terms using a round robin approach (the first on-bit comes from the first query term, the second on-bit comes from the second query term, and so on). This ensures that each query term contributes to the query evaluation.

To find an intuitive explanation of the stopping condition, we substitute $\ln op \cong op - 1$[1] in Equation (4.15) and we obtain (since the optimum number of

* Since $0 < op \le 0.5$ holds, by taking $k = op - 1$ we can apply the linear approximation $\ln(k+1) \cong k$.

evaluation steps is an integer we relax the equality condition as “greater than or equal to”):

$$T_{slice} \geq N \cdot op^{i} \cdot (1 - op) \cdot T_{resolve} \tag{4.17}$$

In Equation (4.17), $N \cdot op^{i} \cdot (1 - op)$ gives the expected number of false drops which will be eliminated if we process the i+1st bit slice after processing *i* bit slices. At the stopping step the time required to process a bit slice becomes equal to the time required to resolve these false drops by accessing the actual records.

4.5 Considering Multi-Term Query Environments

The stopping condition may leave unused on-bits in the query signatures. For such configurations decreasing the *S* value while keeping the *F* value unchanged decreases the on-bit density. Each step eliminates more false drops with lower on-bit density. Consequently, the stopping condition is satisfied by processing less number of bit slices and the response time decreases. On the other hand, the reduced *S* value must provide enough on-bits in the query signatures to reach the stopping condition.

Optimizing the signature file parameters according to a specific number of query terms may give poor performance in a multi-term query environment. Therefore, the submission probabilities of queries with varying number of terms must be considered in the optimization of signature file parameters. The expected response time, *TR*, in a multi-term query environment can be computed as follows.

$$TR = \sum_{t=1}^{t_{\max}} P_t \cdot RT(S,t) \tag{4.18}$$

where P_t is the probability of submission of a *t* term query, and t_{max} is the maximum number of terms that can be used in a query. *RT(S,t)* is the expected response time of a *t* term query expressed as a function of *S* and *t* as follows.

$$RT(S,t) = i \cdot T_{slice} + (1 - (1 - S/F)^{D})^{i} \cdot N \cdot T_{resolve} \tag{4.19}$$

where *i* is computed with Equation (4.16) and $0 < i \leq F \cdot (1 - (1 - S/F)^{t})$ holds.

The derivative of *TR* with respect to *S* is very complicated. Since *S* must be an integer between 1 and $\lceil F \cdot \ln 2 / D \rceil$ (the upper bound corresponds to the *S* value which satisfies the optimality condition), the domain of *S* is finite and very small (note that *S* << *F*). Therefore, the value of *S* that minimizes the value of *TR* can be found with a linear search in this range. The linear search algorithm outlined in Figure 4.2 finds the optimum *S* value for given values of *F*, *D*, *N*, and P_t distribution.

```
Algorithm FindOptimumS
MinimumResponseTime ← infinity
for NewS = 1 to $\lceil F \cdot \ln 2 / D \rceil$
  { NewTime ← Compute the expected response time with Equation (4.18) using NewS
    if NewTime < MinimumResponseTime then
      { S ← NewS
        MinimumResponseTime ← NewTime
      }
  }
```

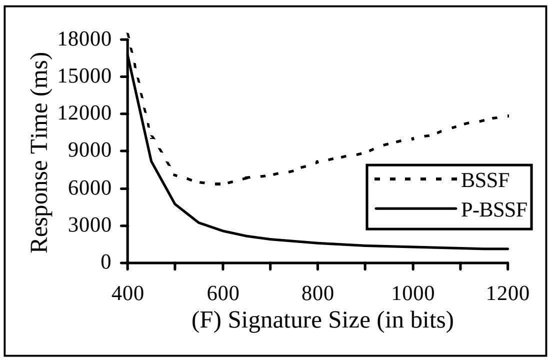
Figure 4.2. Algorithm to find the optimum *S* value for P-BSSF.

The optimum S value for the queries containing only *b* terms can be obtained by taking $P_b = 1$ and $P_t = 0$ for $t \neq b$ and $1 \leq t \leq t_{\max}$. Therefore, B'SSF and GFSSF with *s* = 1, i.e., each frame is a bit slice, are special cases of P-BSSF.

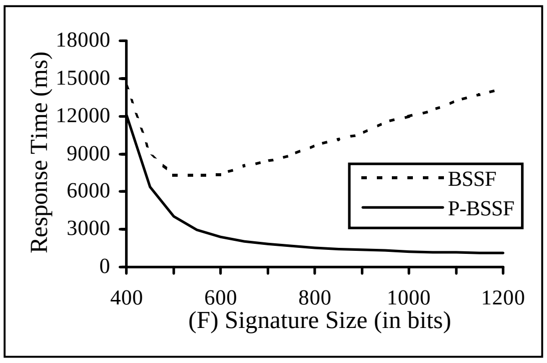
4.6 BSSF vs. P-BSSF: Performance Comparison with Simulation Runs

Expected response time values of LW, UD, and HW query cases obtained by simulation runs for $SP = 1.0$, $N = 10^6$, and changing *F* values are plotted in Figure 4.3. In (d) the improvement percentage obtained by P-BSSF over BSSF is plotted. For increasing *F* values, the response time of BSSF first decreases than starts to increase. The minimum response times are obtained at the *F* values 570, 530, and 500 for LW, UD, and HW query cases, respectively. Therefore, in computing IP values, we used the minimum response time values obtained at these *F* values instead of using increased response time values of BSSF for larger *F* values.

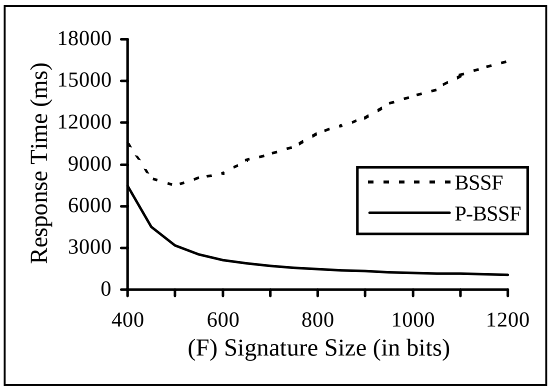
In BSSF, *S* increases for increasing *F* since *S* is adjusted to satisfy the optimality condition for each *F* value. At lower space overheads, the query signature contains insufficient on-bits which produces many false drops. Since the weight of the query signature increases for increasing *F* value, the response time decreases rapidly until the expected number of false drops is reduced to an optimum value. Increasing the *F* value after reaching the optimum point just increases the response time due to processing additional bit slices without eliminating any false drops. Therefore, there is an optimum space overhead for each *N* value that provides minimum response time for BSSF. For smaller *N* values or higher *t* values minimum response time is obtained at lower space overheads.



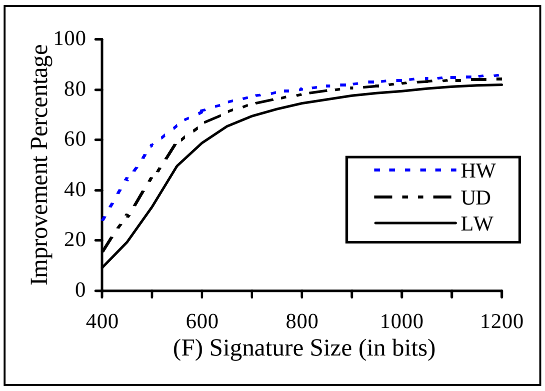
a. LW query case.



b. UD query case.



c. HW query case.



d. IP(BSSF, P-BSSF).

($SP = 1$, $N = 10^6$)

Figure 4.3. Expected response time versus *F* for BSSF and P-BSSF and IP(BSSF, P-BSSF) for LW, UD, and HW.

In P-BSSF, *S* is adjusted for each *F* value to obtain minimum response time. At lower space overheads, the weights of the queries are insufficient to reduce the expected number of false drops to the optimum value. Therefore, both methods produce similar results until sufficient on-bits are obtained in the query signatures. For P-BSSF, unlike the BSSF method, increasing the signature size after obtaining sufficient on-bits in the query signature reduces *op*, that causes a decrease in the response time. For $N = 10^6$, $SP = 1$, and $F = 1200$ the LW, UD, and HW query cases yield expected response times of 1.12, 1.11, and 1.06 seconds, respectively.

Higher number of query terms provide more on-bits in the query signature. Therefore, for P-BSSF, *S* can take smaller values that provide low *op* values.

Consequently, the stopping condition is reached by processing fewer number of bit slices and the response time of P-BSSF decreases for increasing number of query terms. This property makes P-BSSF a promising method for the applications with high number of query terms, such as image databases [ZEZ91].

Since the response time of BSSF increases for the *F* values greater than the optimum space overhead, the space overhead must be fixed at the optimum *F* value. We can compute the performance improvement of P-BSSF over BSSF with respect to additional space overhead incurred by selecting a higher *F* value for P-BSSF. For example, for the UD query case P-BSSF with $F = 1200$ provides a query processing time improvement of 84.30% over the optimum BSSF with $F = 530$ (for this case the response time values are 1110 ms and 7072 ms for P-BSSF and BSSF, respectively).

The simulation runs for various *SP* values show that similar performance improvements are achieved for smaller *SP* values while the response times of both methods increase for decreasing *SP* value. Also, for other *N* values similar performance improvement values are obtained.

The stopping condition usually requires processing of more than *S* bit-slices for the queries containing more than one term in the optimized configurations of P-BSSF. Therefore, the suggestion of Sacks Davis et al., using *S* bit slices for multi-term queries [SAC87], obtains a higher response time than the response time of P-BSSF due to increased number of false drops. Optimizing signature file parameters by considering only single term queries reduces expected false drops by increasing *S*. Consequently, the number of bit slices used in the query evaluation increases which yields higher response time.

4.7 Experiments with Real Data

First, we inspect the distribution of the on-bits in the bit slices. Although the computed *op* value is approximately equal to the average on-bit density value, there are bit-slices with too low or too high *op* values. Since the partial evaluation approach may use a subset of the on-bits of a query signature, the selection method of the on-bits used in the query evaluation affects the results of the experiments. Therefore, other query signature on-bit selection strategies may obtain better results without providing equal contribution of each query term to the query evaluation.

For this reason, we tested three different query on-bit selection methods. Sequential Selection (SS): on-bits are selected sequentially starting from the leftmost on-bit of the query signature to the right. Minimum *op* First (MF): all on-bits in the query signature are sorted in increasing on-bit density value and processed in this order. Round Robin (RR): on-bits of each query term are ordered in increasing on-bit density value and used in this order. The first on-bit is selected from the first term, the second on-bit is selected from the second term, and so on.

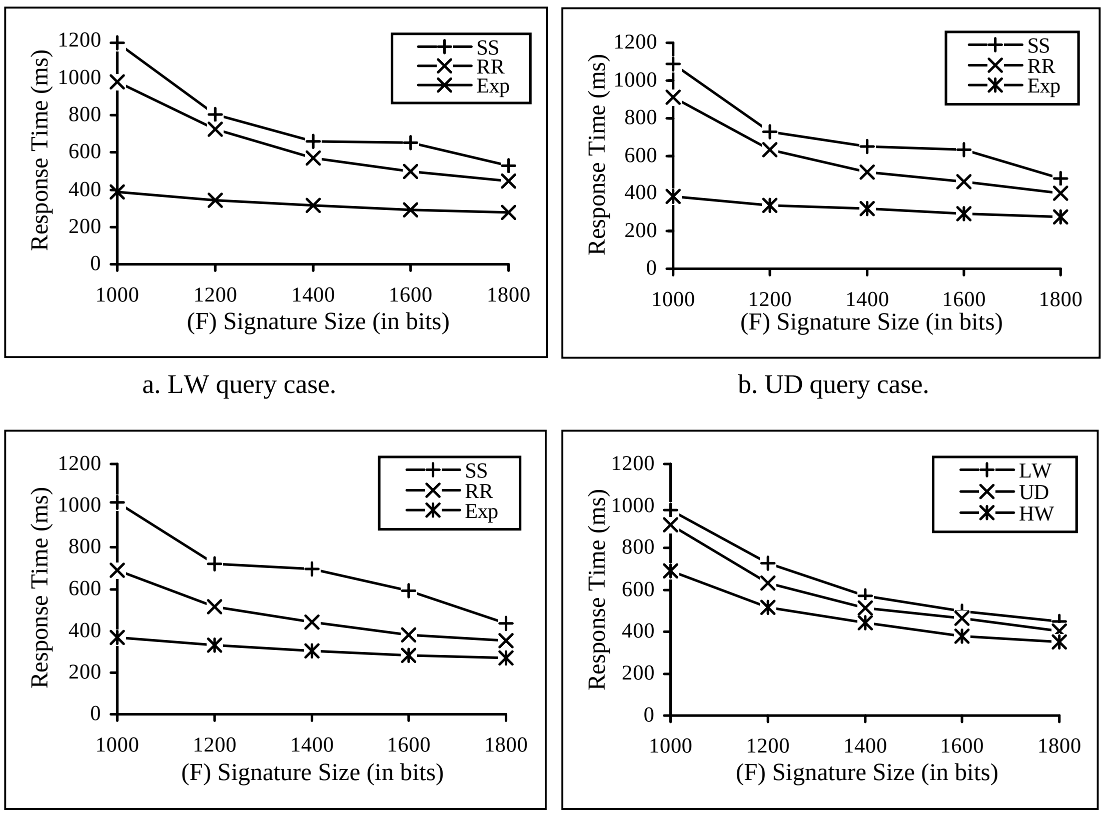
For each query case and query signature on-bit selection method the expected (denoted by Exp) and the observed (denoted by the on-bit selection method) average false drop values are given in Table 4.1. Since all query signature on-bit selection methods process the same number of bit slices, the number of observed false drops can be a decision criteria for this case. However, to show the effect of the decrease in the observed number of false drops on the response time, we plotted the response time values obtained in these experiment for LW, UD, and HW query cases in Figure 4.4 (since MF and RR obtain similar results we omitted the results of MF). To illustrate the effect of increasing query weight on the response time, we combined the results of the RR bit selection method for LW, UD, and HW query cases in Figure 4.4 (d).

Table 4.1. Expected and Observed Average FD Values for the Query On-Bit Selection Methods

	LW				UD				HW			
F	**Exp**	**SS**	**MF**	**RR**	**Exp**	**SS**	**MF**	**RR**	**Exp**	**SS**	**MF**	**RR**
1000	0.53	6.55	4.78	4.88	0.42	5.49	3.34	3.41	0.82	5.72	2.65	2.76
1200	0.62	4.64	3.86	3.86	0.47	3.77	2.79	2.82	0.32	2.65	1.56	1.59
1400	0.29	2.71	2.40	2.44	0.22	1.83	1.42	1.43	0.63	2.90	2.06	2.06
1600	0.42	3.60	2.22	2.22	0.42	3.10	1.28	1.31	0.22	2.75	0.79	0.85
1800	0.24	2.46	1.72	1.76	0.36	1.74	0.92	0.97	0.22	1.34	0.46	0.49

Due to the non-uniform distribution of the number of distinct terms in the records, the observed average false drop values are greater than the expected values. The difference between expected and observed false drop values decreases dramatically for increasing *F* values. For $F \leq 800$, we obtain too many false drops. Consequently, the response time is very high and using a signature file with such a space overhead is impractical.

The SS method obtains highest false drop values in all experiments. Generally, MF obtains slightly better results than RR for all query cases. However, since RR selects the bits from the query terms in a round robin approach, it maximizes the contribution of all terms to the query evaluation. For non zero hit queries, excluding some query terms from the query evaluation may increase the number of false drops. Therefore, we prefer the RR method.



a. LW query case.

b. UD query case.

c. HW query case.

d. RR for LW, UD, and HW.

Figure 4.4. Expected and observed response time values for SS and RR in P-BSSF for LW, UD and HW (*SP* = 1).

The observed false drop values of the SS method should be similar to the expected false drop values. Also, the observed false drop values of MF and RR methods should be smaller than the estimated false drop values since they use the on-bits with minimum *op* value first. However, the results of the experiments show that the reduction in the observed number of the false drops is less than the estimated ones. To obtain fewer numbers of false drops, the distribution of the number of distinct terms in the records must be considered in the optimization of the signature file parameters. This issue will be considered in detail in Chapter 6.

The observed response time decreases for increasing signature size. Also, differences between the expected and the observed response time values decrease for increasing *F* value. The experiments show that obtaining a response time around 0.6 seconds is possible for the test database with a signature size greater than or equal to 1200 bits (this corresponds to 24% or more space overhead) by using a personal computer. We repeated the same experiments in the UNIX environment by using a Sparc Server Model 10-51. About 55 other users were running SQL processes using the library collection database of Bilkent University during the experiments. We obtained very promising response times in such a multi-user environment where the value of *SP* can be considered as zero. For example, for *F* = 1200 the LW, UD, and HW query cases obtain the response times of 0.68, 0.51, and 0.45 seconds, respectively. (All UNIX numbers are obtained using the “elapsed time” feature of the system.)

5. THE MULTI-FRAGMENTED SIGNATURE FILE METHOD

In vertical signature partitioning low on-bit density (the probability of a particular bit of a bit slice being on-bit) provides rapid reduction in the expected number of false drops. Thus, the stopping condition defined for P-BSSF is reached by processing fewer number of bit slices. For a given *D* value, *op* can be reduced by either increasing *F* or decreasing *S* (see Equation 4.12). For P-BSSF, the value of *S* is selected to obtain the minimum response time in a multi-term query environment. Therefore, decreasing *S* will produce insufficient on-bits in the query signature of low-weight queries and the number of false drops will increase for these queries. This will also increase the response time.

The performance of P-BSSF can be improved if the on-bit density can be reduced while providing enough on-bits in the query signature of low weight queries. In this chapter we propose a new signature generation and query evaluation method, Multi-Fragmented Signature File (MFSF), which improves the performance of P-BSSF without increasing the space overhead (*F* value).

MFSF decreases the response time in multi-term query environments by dividing the signature file into variable sized sub-signature files, *fragments*. Each fragment is a separate BSSF with its own *F* and *S* parameters and the optimality condition is relaxed. Therefore, in MFSF each fragment may have a different on-bit density as opposed to the uniform on-bit densities of the BSSF, B'SSF, GFSSF, and P-BSSF methods.

This Chapter is organized as follows. The proposed method, MFSF, is described in Section 5.1. The false drop estimation formula for MFSF is derived in Section 5.2. In Section 5.3, a stopping condition is defined which provides decreasing response time for increasing numbers of query terms. A heuristic search algorithm to obtain the optimized configuration of MFSF is given in Section 5.4. In Section 5.5, an example

MFSF configuration with false drop computations is provided. In Section 5.6, the performance of MFSF is compared with P-BSSF and GFSSF with simulation runs. Finally, Section 5.7 contains the results of experiments with real data.

5.1 MFSF: Multi-Fragmented Signature File

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

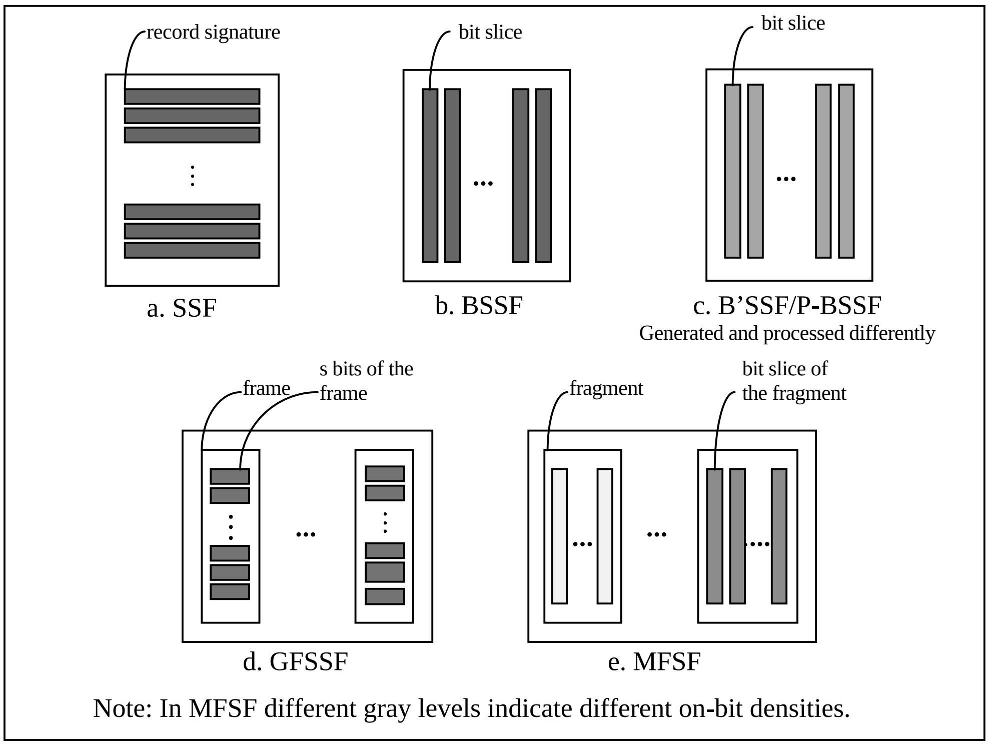

record signature
bit slice
bit slice
a. SSF
b. BSSF
c. B'SSF/P-BSSF
Generated and processed differently
frame
s bits of the frame
fragment
bit slice of the fragment
d. GFSSF
e. MFSF
Note: In MFSF different gray levels indicate different on-bit densities.

Figure 5.1. Graphical representation of SSF and vertical partitioning methods.

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

$$W(Q)_{(r,t)} = F_r \cdot (1 - (1 - S_r/F_r)^t) \quad for\ \ 1 \le r \le f \tag{5.1}$$

$$w_t = \sum_{r=1}^{f} W(Q)_{(r,t)} \tag{5.2}$$

The on-bit density values of the fragments are

$$op_r = 1-(1-{}^{S_r}\!/\!{}_{F_r})^D \qquad for\ 1 \le r \le f\,. \tag{5.3}$$

Graphical representations of SSF, BSSF, B'SSF, P-BSSF, GFSSF, and MFSF are illustrated in Figure 5.1. A horizontal box represents the sequential storage of the bits in the box. First are stored the bits of the first box, then the bits of the second box and so on. A vertical box represents the sequential storage of the bits in the box from the top to the bottom. The on-bit density values of the bit strings are represented with the gray level of the box. A darker area has higher on-bit density than the lighter one. Note that the highest on-bit density is 0.5 and the on-bit densities of SSF and BSSF are always 0.5. A summary of the vertical partitioning methods is given in Table 5.1.

Table 5.1. Properties of Vertical Signature File Partitioning Methods

Properties \ Signature File Methods	**BSSF**	**B'SSF**	**GFSSF**	**P-BSSF**	**MFSF**
On-bit Density (op) < 0.5 is Allowed	No	Yes	Yes	Yes	Yes
Exploits different bit slice densities	No	No	No	No	Yes
Optimized in Multi-Term Query Env.	No	No	No	Yes	Yes
Partial Evaluation Strategy Defined	No	No	No	Yes	Yes
Obtaining the Optimum Configuration	Exact	Exact	Heuristic	Exact	Heuristic

5.2 False Drop Computation for MFSF

In Section 4.4 we defined fd_i as the false drop probability if i bit-slices ($0 \le i \le w_t$) are used in the first phase of a query evaluation. For MFSF, fd_i is computed by multiplying the on-bit densities of the bit slices used for the query evaluation as follows.

$$fd_i = \prod_{s=1}^{i} b_s \tag{5.4}$$

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

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

$$op_r \le op_{r+1} \quad \text{for } 1 \le r < f \tag{5.5}$$

holds for all fragments.

In the query evaluation the on-bits of the lower on-bit density fragments are used first. This rule is specified as

$$b_s \le b_{s+1} \quad for \;\; 1 \le s < w_t \tag{5.6}$$

and ensures that the stopping condition is reached in fewest number of evaluation steps. As the number of query terms increases, the number of query signature on-bits in the lower on-bit density fragments increases. Therefore, the stopping condition will be reached in a fewer number of evaluation steps and hence the query evaluation time will decrease for increasing number of query terms (in Section 5.5 we provide a numerical example for this).

If we consider only one fragment, say fragment *r*, and all query signature on-bits of this fragment are used in the query evaluation, the false drop probability of this fragment becomes $op_r^{W(Q)(r,t)}$ (note that $W(Q)_{(r,t)}$ is the query weight of the *r*th fragment for a *t* term query). If *d* on-bits are used from a fragment, say the *h+1*st fragment, the inequalities (5.5) and (5.6) ensure that all of the query signature on-bits of the lower numbered fragments (the first *h* fragments) were already used in the query evaluation. Therefore, the number of bit slices used in the query evaluation, *i*, is computed by adding the query weights of these lower numbered fragments and *d*.

$$i = d + \sum_{r=1}^{h} W(Q)_{(r,t)} \quad \text{where } h < f,\; 0 \le d \le W(Q)_{(h+1,t)} \tag{5.7}$$

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

$$fd_i = op_{h+1}^{d} \cdot \prod_{r=1}^{h} op_r^{W(Q)_{(r,t)}} \tag{5.8}$$

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

5.3 Stopping Condition for MFSF

In P-BSSF, the first phase of the query evaluation with the signature file stops when the stopping condition given in Equation 4.17 is satisfied. This is to say in P-BSSF the query evaluation stops when the P-BSSF bit slice processing time becomes equal or greater than the time required to resolve the false drops which will be eliminated by processing this bit slice by accessing the actual records.

To derive the stopping condition for MFSF, first we obtain a general stopping condition for vertically partitioned signature files and then we will apply this formula to MFSF.

In P-BSSF, the expected number of false drops after processing i bit slices, FD_i, is computed as follows.

$$FD_i = N \cdot fd_i = N \cdot op^i \tag{5.9}$$

We define RFD_{i+1}, the number of *reduced false drops*, as the number of false drops which will be eliminated by processing an additional bit slice after processing i bit slices. We derive the formula for RFD_{i+1} as follows.

$$\begin{aligned} RFD_{i+1} &= FD_i - FD_{i+1} \\ &= N \cdot op^i - N \cdot op^{i+1} \\ &= N \cdot op^i \cdot (1 - op) \end{aligned} \tag{5.10}$$

We substitute RFD_{i+1} in the stopping condition of P-BSSF (Equation (4.17)) and we obtain

$$T_{slice} \geq RFD_{i+1} \cdot T_{resolve} \, . \tag{5.11}$$

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

To obtain the stopping condition for MFSF we derive the formula to compute RFD_{i+1}. The false drop probability after processing $i+1$ bit slices is

$$fd_{i+1} = fd_i \cdot b_{i+1} \tag{5.12}$$

where b_{i+1} is the false drop probability of the $i+1$st bit slice used in signature file processing. All query signature on-bits of the first h fragments and d on-bits of the $h+1$st fragment are used to process i bit slices (see Equations (5.8) and (5.9)). Therefore, if there is an unused on-bit in the $h+1$st fragment, i.e., if $d < W(Q)_{(h+1,t)}$, b_{i+1} will be equal to op_{h+1}. If all on-bits of the $h+1$st fragment are already used, i.e., $d = W(Q)_{(h+1,t)}$, the $i+1$st on-bit will be selected from the $h+2$nd fragment if the $h+2$nd fragment exists (i.e., if $h+2 \le f$). By considering this discussion the value of b_{i+1} is determined as follows.

$$b_{i+1} = \begin{cases} op_{h+1} & if\ d < W(Q)_{(h+1,t)} \\ op_{h+2} & otherwise\ (if\ h+2 > f\ query\ evaluation\ is\ completed) \end{cases} \tag{5.13}$$

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

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

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

We obtain the following stopping condition for MFSF by substituting RFD_{i+1} in Equation (5.11).

$$T_{slice} \ge N \cdot fd_i \cdot (1 - b_{i+1}) \cdot T_{resolve} \tag{5.14}$$

To prove the stopping condition given in Equation (5.14) is valid in subsequent steps we have to consider the following theorem.

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

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

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

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

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

Since the value of on-bit density is a probability and signatures with *op* = 0 or *op* = 1 are meaningless, $0 < b_s < 1$ holds for $1 \le s \le w_t$. Consequently, $fd_i > 0$ holds for $1 \le i \le w_t$. We cancel fd_i and N ($N > 0$). Since $b_{i+2} \ge b_{i+1}$, we can replace b_{i+2} with $b_{i+1} + \alpha$ such that $1 > \alpha \ge 0$ and we obtain

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

$1 - 2 \cdot b_{i+1} + b_{i+1}^2 + \alpha \cdot b_{i+1} > 0$

$(1 - b_{i+1})^2 + \alpha \cdot b_{i+1} > 0$

Since $\alpha \ge 0$ and $0 < b_{i+1} < 1$ hold, the above inequality holds and *RFD* is decreasing.□

Since the cost of processing a bit slice is the same in all fragments, the above proof guarantees that once the stopping condition given in Equation (5.14) is satisfied, it will be valid in subsequent steps.

5.4 Searching the Optimum Configuration

To find the first relevant record, the first phase must be completed which requires retrieval and processing of *i* bit slices (*i* is determined by using the stopping condition given in Equation (5.14)). Since P-BSSF and MFSF optimize the response time in multi-term query environments, the response time computation formulas are the same. However, for the sake of completeness we repeat these formulas in this section. The response time for a *t* term query with *i* slice processing and FD_i actual record accesses is computed as follows.

$$RT_t = i \cdot T_{slice} + FD_i \cdot T_{resolve}. \tag{5.15}$$

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

$$TR = \sum_{t=1}^{t_{max}} P_t \cdot RT_t \qquad (5.16)$$

where RT_t, given in Equation (5.15), is the time required to evaluate a t term query, P_t is the probability of submission of a t term query, and t_{max} is the maximum number of terms that can be used in a query.

The values of the parameters N and D involved in the response time computation depend on the database instance. Therefore, minimizing the response time, TR, with the stopping condition given in Equation (5.14) requires determination of parameters f, F_r, and S_r ($1 \le r \le f$) for a given F value. The heuristic search algorithm outlined in Figure 5.2 is used to search the optimum configuration and to determine the TR value for this case.

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

Figure 5.2. Algorithm to search optimal fragmentation scheme.

The algorithm starts with a randomly determined initial fragmentation scheme. A candidate configuration is obtained by changing the value of a randomly chosen parameter. Since the algorithm minimizes the response time for a given F value, *Join Fragments, Increase* F_r (add 1 to F_r), and *Decrease* F_r (subtract 1 from F_r) operations of the algorithm require random selection of another fragment, p, and adjusting the F_p value of this fragment accordingly. In the algorithm, joining of two fragments to form one fragment is initiated when *decrease* S_r is selected and the S_r value in the selected fragment is one. The split operation divides the selected fragment into two fragments of different sizes. Their sizes are selected randomly. Actually one size is determined randomly, since their total size is the same as the split fragment.

After obtaining the candidate configuration, the consistency of the parameters is ensured such as $1 \le S_r \le F_r$ holds for $1 \le r \le f$. To prevent trapping in a local minima, a sufficient number of initial configurations must be tried. The results given in this study are obtained with 20 initial trials (similar results are obtained with higher numbers of initial configurations).

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

5.5 Example MFSF Configuration

To illustrate the computation of *TR* values of P-BSSF and MFSF, we provide a numerical example in Figure 5.3. The configurations are obtained using the values of the experimental system parameters. The optimized configuration and the stopping step, i, for P-BSSF is obtained as proposed in Chapter 4.

Except t = 1, the response time of P-BSSF remains unchanged for increasing number of query terms. In MFSF, fragmenting a signature file reaches the stopping condition in fewer evaluation steps and the response time decreases for increasing number of query terms.

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

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

P-BSSF Configuration
S = 6, op = 0.121, i = 7

i stands for the number of slices used to reach the stopping condition

Response Time Calculation For MFSF

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

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

Response Time Calculation For P-BSSF

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

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

(TR is obtained for query case UD)
Figure 5.3. Example response time calculations for P-BSSF and MFSF.

5.6 Performance Comparison with Simulation Runs

We used the IP (improvement percentage) value in the comparison of the performance of MFSF with GFSSF and P-BSSF. Note that BSSF and B'SSF are special cases of both P-BSSF and GFSSF. Therefore, we exclude BSSF-MFSF and B'SSF-MFSF cases in the comparisons.

The same $T_{resolve}$ value is used in response time calculations of GFSSF, P-BSSF, and MFSF. The GFSSF approach uses a different storage structure. Therefore, T_{slice} for GFSSF is computed by considering the frames of GFSSF. The false drop probability estimation method proposed in [LIN92] requires extensive computations to optimize a configuration. Therefore, we computed the false drop probability of GFSSF by using the approximation proposed in [KOÇ95a]. The false drop probability computation method proposed in [KOÇ95a] converges to the false drop computation method of B'SSF for a frame size of one bit. B'SSF is a special case of both GFSSF and MFSF and in most of the inspected cases, GFSSF converges to B'SSF producing a frame width of one bit. Therefore, this approximation works well for GFSSF.

The optimization method of GFSSF is defined for a given number of query terms [LIN92]. Since there may be queries with different number of query terms in a multi-term query environment, we obtained *TR* value for GFSSF as follows. First, we obtained the optimized configuration of GFSSF $t = 1$ as proposed in [LIN92]. Then, we computed *TR* value of this configuration by considering the probability distribution of the number of query terms (P_t values) in the inspected multi-term query environment. We repeated the same computations for $t = 2$, $t = 3$, $t = 4$, and $t = 5$ and we obtained five different *TR* values. We selected the minimum *TR* value among these five *TR* values as the *TR* value of the inspected case. In other words, in our comparisons our treatment of GFSSF is more than fair. In most of the inspected cases, the configuration optimized by taking $t = 1$ gives minimum response time in a multi-term query environment.

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

5.6.1 Effect of Number of Query Terms, Signature Size and Placement of Disk Blocks

To simulate a multi-term query environment, P_t values are determined by assuming a bounded normal distribution from left and right. The change in P_t values are modeled by changing variance, *V(t)*, and the expected number of query terms, *E(t)*, values ($1 \le t \le 5$). P_t values for the inspected *V(t)* and *E(t)* values are given in Table 5.2. The difference among P_t values, hence the effect of changing *E(t)* values, decreases for

V(t) values greater than or equal to 10. Consequently, P_t values are approximately equal for these distributions (*V(t)* ≥ 10) and they are modeled by the uniform distribution (UD) where all P_t values are equal to 0.2 and invariant of the change in *E(t)*. Therefore, we consider only *V(t)* = 1 and *V(t)* = 5. The case *V(t)* = 0 implies an environment with queries only with *t* number of terms, i.e., P_t = 1. Since it is unrealistic we omit this case.

Table 5.2. P_t Values for *V(t)* = 1 and *V(t)* = 5

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

IP values of GFSSF-MFSF and P-BSSF-MFSF for varying *V(t)* and *E(t)* values are plotted in Figure 5.4. In general the effect of the fragmentation on the performance of MFSF increases as the possibility of queries with various number of query terms increases, i.e., as more P_t ($1 \leq t \leq t_{max}$) cases assume non-zero probability values. (Notice that the exclusive experimental setting of Table 5.2 gives us an opportunity to further investigate the effects of the number of terms on query processing performance.) For example, for *V(t)* = 1 and *E(t)* = 1 (P_1 = 0.553 and P_2 = 0.351 and other P_t values are negligible, see Table 5.2) the IP value for GFSSF-MFSF case is 35.37%. In the UD case all P_t values are equal to 0.2 and the IP value for the GFSSF-MFSF case increases to 70%. Since the UD cases exhibit an average performance, we will use only the UD case in the following comparisons.

Improvement percentage values for varying signature sizes (for *SP* = 0.0 and 1.0) are plotted in Figure 5.5. For large databases ($N \geq 10^6$), signature sizes less than 800 bits, corresponding to a space overhead less than 20%, produce many false drops and the response time is relatively high. Therefore, for practical purpose, we consider *F* values greater than 800 for such large databases. For *F* > 800, the IP value varies between 11% and 12.7% for the P-BSSF-MFSF case and between 65% and 70% for the GFSSF-MFSF case. This shows that, except for small *F* values ($F < 800$), the performance improvement is quite invariant to changing *F* values.

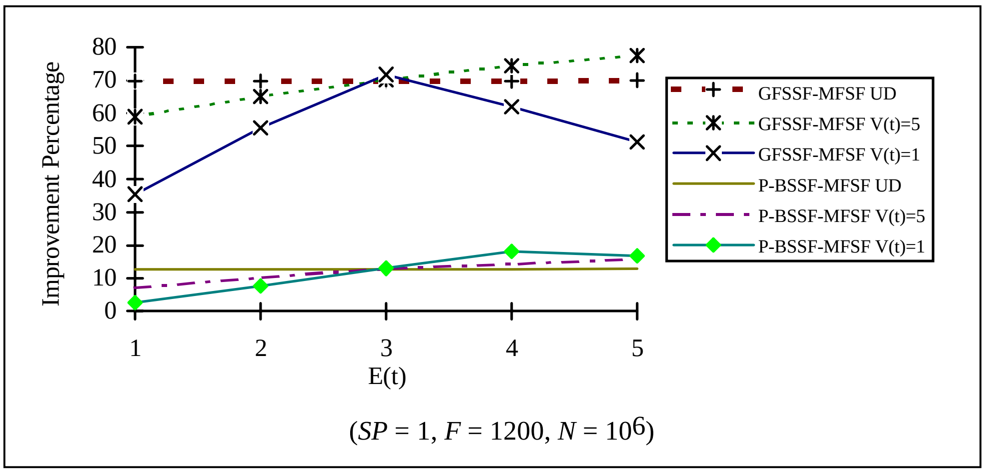


Figure 5.4. IP values of GFSSF-MFSF and P-BSSF-MFSF versus varying *E(t)* -and *V(t)*- values.

To inspect the effect of sequentiality probability (*SP*) on the IP values for changing *F* values we included the extreme cases for SP in Figure 5.5. Except for small *F* values, the same relative performances were obtained for all SP values.

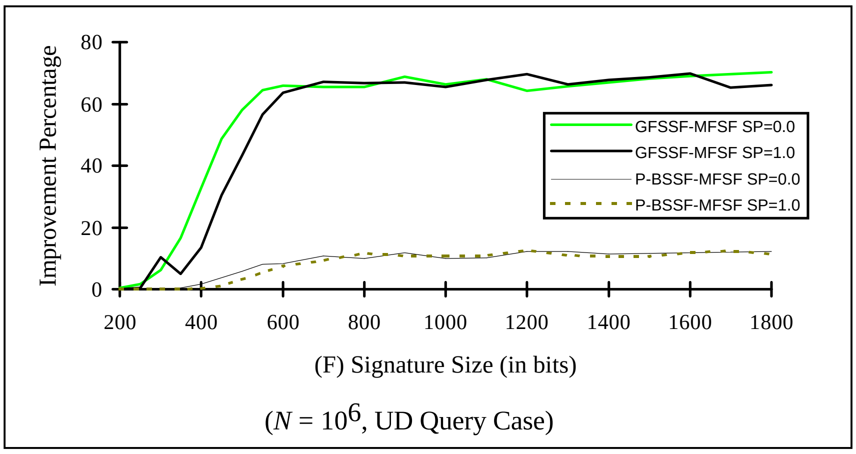


Figure 5.5. IP values of GFSSF-MFSF and P-BSSF-MFSF versus varying *F* values.

We want to revisit the effect of the number of query terms on performance one more time. As shown in Figure 5.4, the effect of fragmenting a signature file increases if the possibility of queries with different number of query terms increases. The maximum number of query terms is limited by t_{max} in our optimization model.

We plot the IP values for increasing t_{max} values in Figure 5.6. For $t_{max} = 1$, i.e., when there are only single term queries, all methods obtain the same response time and IP values are zero. For increasing t_{max} values, the number of queries with different number of query terms increases. This increases the performance of MFSF over P-BSSF and GFSSF. Note that t_{max} value used in other comparisons (t_{max} = 5) is below the saturation point (t_{max} = 10) where IP values of P-BSSF-MFSF and GFSSF-MFSF cases are 16.9% and 84.78%, respectively.

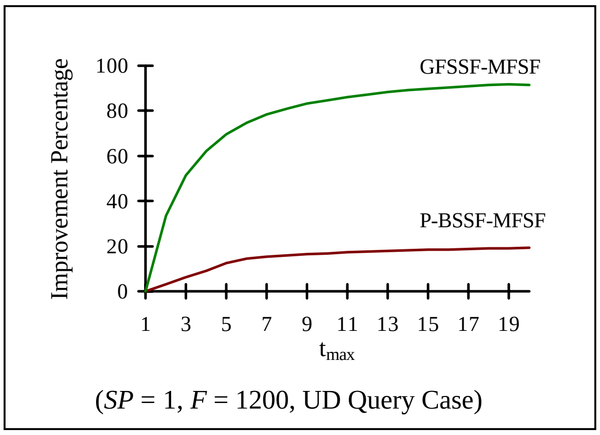


Figure 5.6. IP values of GFSSF-MFSF and P-BSSF-MFSF versus varying t_{max} values.

5.6.2 Effect of Database Size

The performance improvement values for changing *N* values are plotted in Figure 5.7. For *N* values near 2000, IP values of P-BSSF-MFSF reach 10% and vary between 11% and 12.7% for increasing *N* values. IP values of GFSSF-MFSF rise to 65% for *N* = 30,000 and vary between 65% and 70% for increasing *N* values. Therefore, except for very small *N* values (*N* < 2000), MFSF performs better than GFSSF.

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

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

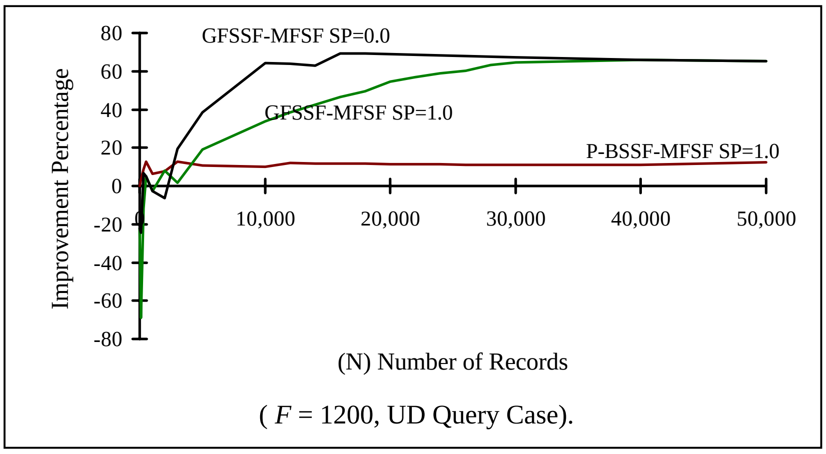


Figure 5.7. IP values of GFSSF-MFSF and P-BSSF-MFSF for varying *N* values.

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

5.7 Experiments with Real Data

5.7.1 Determining the Query Signature On-Bits Used in the Query Processing

In Section 4.7 we inspected SS, MF, and RR query signature on-bit selection methods and we showed that RR is the best one. In RR, to maximize the number of query terms that contribute to the first phase of the query evaluation, the first on-bit is selected from the first query term, the second on-bit is selected from the second term, and so on. In MFSF, generally, each term sets only one bit in the lower numbered fragments and the on-bits of a lower numbered fragment are used first. Therefore, in MFSF, the RR method and random selection of the query signature on-bits for query evaluation produce similar results.

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

Although the observed and estimated average on-bit density values of the bit slices of MFSF agree, we observed higher *op* values than the estimated value at the bit positions where a high frequency term (a term occurring in many records) sets bits. When possible, to prevent using bit slices with high *op* value in the query evaluation, we sorted the on-bits of a query term in non-decreasing *op* value. The RR bit selection method uses on-bits of each query term in this order. Sometimes, this may cause using an on-bit from a higher numbered fragment before using the on-bits of the same term in the lower numbered fragments. Since this policy may prevent using the bit slices with high *op* value, the number of observed false drops and the response time decrease.

5.7.2 Results for False Drops and Query Processing Time

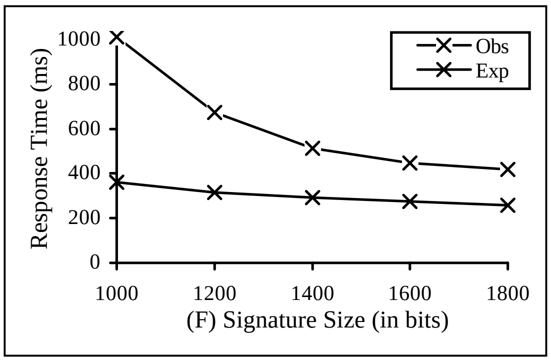
The expected (denoted by Exp) and the observed (denoted by Obs) average false drop values of MFSF for the query cases are given in Table 5.3. The expected and observed response times of the query cases are plotted in Figure 5.8. To illustrate the effect of increasing query weight on the response time, we combined the observed response time values for LW, UD, and HW query cases in Figure 5.8 (d).

Table 5.3. Expected and Observed Average False Drop Values for the Query Cases LW, UD, and HW

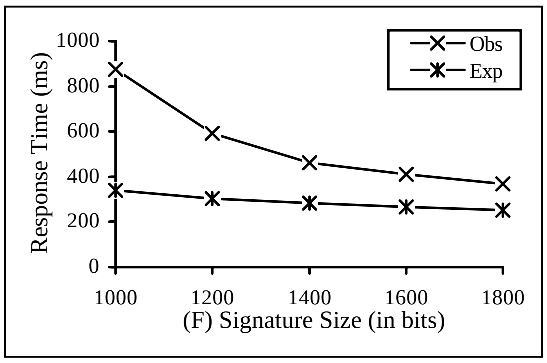
	LW		UD		HW	
F	**Exp**	**Obs**	**Exp**	**Obs**	**Exp**	**Obs**
1000	0.60	5.37	0.64	4.82	0.47	3.11
1200	0.54	2.92	0.43	2.57	0.29	1.37
1400	0.43	2.15	0.30	1.50	0.35	1.45
1600	0.43	1.51	0.35	1.26	0.27	0.97
1800	0.37	1.31	0.30	1.02	0.21	0.60

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

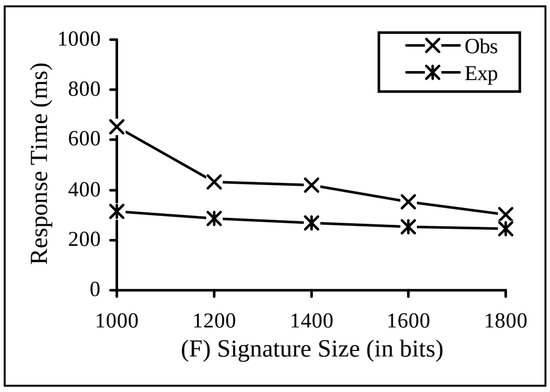
A small fraction of the records in the test database are very large with respect to the average record size (D_{avg}); for example, there are 584 MARC records containing more than 75 terms which constitute 0.38% of the test database and the maximum number of distinct terms in the records is 166. These large records cause an increase in the observed number of false drops. This also causes an increase in the observed response time.



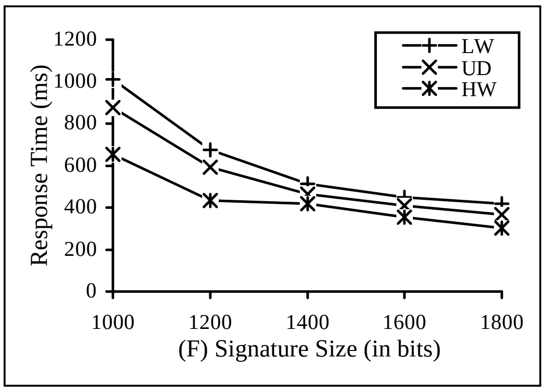
a. LW query case.



b. UD query case.



c. HW query case.



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

Figure 5.8. Expected and observed response time of MFSF versus *F* for LW, UD and HW (*SP* = 1).

We tested the effect of these large records on the response time by removing them from the test database. The signature file parameters for the reduced databases are optimized by using new average number of distinct terms and taking *F* = 1200. The results for the UD query case are given in Table 5.4. The percentage deviation from the expected response time is computed as

$$\%\ Deviation\ of\ TR = 100 \cdot (TR_{Observed} - TR_{Expected}) / TR_{Expected},$$

and *Percentage Deviation of FD* is computed similarly.

The difference between the expected and observed response time values decreases dramatically as the maximum number of distinct terms in the records (second column of Table 5.4) decreases. Since these large records constitute a small fraction of the database, they can be stored in a separate file and searched separately to provide a faster response time.

Table 5.4. Results of Limiting Maximum Number of Terms in the Records

	Max	**Avg.**	**Standard**	**Expected**		**Observed**		**% Deviation**	
N	**D**	**D**	**Deviation of D**	**FD**	**TR(ms)**	**FD**	**TR(ms)**	**FD**	**TR**
152,850	166	25.70	11.12	0.43	303	2.57	541	498	79
152,686	100	25.60	10.72	0.42	302	1.46	440	248	46
152,266	75	25.44	10.24	0.40	301	1.34	402	235	34
149,408	50	24.82	9.26	0.35	296	0.93	342	166	16
140,901	40	23.64	8.12	0.47	284	0.86	308	83	8

6. OPTIMIZATION OF SIGNATURE FILE PARAMETERS FOR VARYING RECORD LENGTHS

Due to hashing and superimposition operations used in obtaining signatures, the signature of an irrelevant record may match the query signature. Each false matching record, *false drop,* must be accessed and compared with the query after processing the signature file. Consequently, to estimate the response time properly, we need to estimate FD accurately. Also, the signature file methods that use the accurately estimated FD value in the optimization of the signature file parameters can achieve the estimated performance in real applications.

The false match probability of a record signature and a query signature increases as the number of on-bits in the record signature increases. (Note that a record signature with only on-bits matches all queries irrespective of the query terms.) Parameters affecting the number of on-bits in a record signature are the length of the record signature (F), the number of distinct terms in the record (D), and the number of bits set to "1" by each term (S).

Generally, signature file methods use the same F value for all records and optimize the value of S as a function of F and D. For example, to minimize the false drop probability, the optimality condition must be satisfied, i.e., half of a record signature bits must be on-bit [CHR84a, ROB79]. To satisfy the optimality condition the following relationship must hold among F, S, and D.

$$S = \frac{F \cdot \ln 2}{D} \qquad (6.1)$$

If the value of D of an individual record increases while S and F remain unchanged, the number of on-bits in the record signature increases and the optimality condition is violated, i.e., a false drop probability higher than the optimality condition can provide is obtained. This implies an increase in FD and response time.

To provide a uniform *D* value in databases with *varying record lengths* (we will use *varying record length* to mean that records may contain different number of distinct terms), long records are divided into blocks containing fixed number of terms [CHR84a]. However, the terms of a multi-term query may be in different blocks of the same record and, especially for the vertical partitioning methods, record level retrieval become complicated for multi-term queries. Another method is using average number of terms in records, D_{avg}, in FD estimation. In this chapter we experimentally show that the use of D_{avg} under estimates *FD* and this causes a performance degradation.

We propose a more accurate false drop estimation method, the Partitioned False Drop estimation method (PFD), for the databases with varying record lengths. In PFD, we conceptually divide a database into disjoint partitions according to the number of distinct terms in the records. Each partition is considered as a separate signature file and average number of distinct terms in a partition is used to estimate FD in this partition. PFD decreases the differences among the number of distinct terms in the records of a partition. Therefore, FD is estimated more accurately.

The FD value estimated by using D_{avg} may be less than the FD value estimated by PFD for the same database instance and signature file parameters (later in Section 6.2 we provide a numerical example for this). Therefore, analytical comparisons of signature file methods estimating FD by using D_{avg} and PFD will be misleading. For this reason, we tested the performance of the inspected signature file methods by the experiments performed with real data. We developed a test environment and implemented the sequential, generalized frame-sliced, and multi-fragmented signature file methods in the C programming language. We extended these methods to use PFD and tested their performances with real data. Experiments show that PFD increases the performance of the inspected methods by reducing the observed FD and the response time. This is achieved by better estimation of signature file design parameters using PFD. Since the PFD approach estimates FD accurately, the number of records which may be added to a dynamic database without any performance degradation can be determined safely. The experiments of this chapter show that PFD obtains better response times than using D_{avg} in FD estimation if a necessary reorganization is delayed.

Minimizing FD is one of the objectives of signature file methods. Since using PFD in a signature file method provides a lower FD and response time without increasing the space overhead, it can be used in other signature file methods to obtain a better performance. For example, PFD will increase the performance of the horizontal partitioning methods that stores the signatures of records of a horizontal partition sequentially [SAC83, SAC85].

The organization of the chapter is as follows. In Section 6.1, the traditional FD estimation method used so far is given. The proposed FD estimation method, PFD, is described in Section 6.2. In Section 6.3, we propose a new method to find the optimum value of *S* for SSF by using PFD and we compare the performance of the proposed SSF optimization method with the SSF method optimized by using average number of distinct terms. In Section 6.4 and 6.5, we apply PFD to the GFSSF and MFSF methods, respectively. Also, the performance improvements obtained by PFD is measured experimentally with real data. In Section 6.6, we inspect the effect of the distribution of the number of terms in the records on the performance of PFD. Section 6.7 inspects the reorganization need for dynamic databases and measures the change in response time experimentally for increasing database size.

6.1 Using Average Number of Terms Per Record in Estimating FD

A record signature qualifies a query accidentally if the record does not contain some query terms and all on-bits of the query signature were also set by the terms of the record. Since more bits in the query signature will be on-bit for higher number of query terms, the false drop probability will decrease for increasing number of query terms. The following exact formula was derived in [ROB79] to compute the false drop probability of a particular record with D terms for a t ($t > 0$) term query.

$$fd(F,S,D,t) = \sum_{i=0}^{F} P(F,S,t,i) \cdot P(F,S,D,i) \tag{6.2}$$

where $P(F,S,n,i)$ is the probability of setting i bit positions of a bit string that is F bits long to “1” by n terms each setting S bit positions to “1”. $P(F,S,n,i)$ is computed as follows [ROB79].

$$P(F,S,n,i) = \sum_{k=0}^{i} (-1)^k \cdot \binom{i}{k} \cdot \left[\binom{F-k}{S} \Big/ \binom{F}{S} \right]^n \tag{6.3}$$

where $\binom{m}{j}$ denotes a binomial coefficient and n is the number of term signatures superimposed. Instead of the exact formula given in Equation (6.2), the following approximate formula were used to estimate the false drop probability of a record with D distinct terms due to its simplicity [ROB79]. (These formulas are given in Section 4.1. We repeat them here for easy reference.)

$$fd(F,S,D,t) = \left(1-(1-\tfrac{S}{F})^D\right)^{W(Q)_t} \tag{6.4}$$

where $W(Q)_t$ is the expected number of on-bits in the signature of a t term query (*query weight*) and it is computed as follows.

$$W(Q)_t = F \cdot \left(1-(1-\tfrac{S}{F})^t\right) \tag{6.5}$$

These approximations are valid for small values of S, D, and t and they give close results to the exact formula [ROB79].

In [FAL88, LIN92] the false drop probability for the whole database is defined as

$$\textit{false drop probability (fd)} \ = \ \frac{\textit{number of false matches (FD)}}{N \ - \ \textit{number of true matches}}.$$

By assuming the number of true matches will be negligible with respect to N, FD is computed by multiplying the false match probability of a record by the number of records in the database (N) as follows [LIN92].

$$FD = N \cdot fd \tag{6.6}$$

Since fd is computed for a specific D value, Equation (6.6) can be used safely for the databases whose individual records contains exactly D terms. In databases with varying number of distinct terms, an average fd value is obtained by using the average number of distinct terms per record, D_{avg}, instead of D in Equation (6.4) [KOÇ95a, b, c, d; LIN88, LIN92, ROB79, SAC87]. We call this approach Average False Drop computation method (AFD).

6.2 Proposed False Drop Estimation Method

For the databases with records containing varying number of distinct terms, each record may have a different *D* and consequently a different *fd* value. Therefore, the expected number of false drops for a database with *N* records can be computed more accurately by adding the individual false drop probabilities of the records as follows.

$$FD = \sum_{r=1}^{N} fd\ of\ record\ r \ = \sum_{r=1}^{N} (1-(1-\tfrac{S}{F})^{D_r})^{W(Q)_t} \tag{6.7}$$

where D_r is the number of distinct terms in rth record. Since individual *fd* values of the records are used in computing FD, we call this method Individual False Drop computation method (IFD). In Equation (6.7) we assume the same *S* and *F* values are used for all records. If *F* and *S* can be adjusted according to the D_r values of the records, a lower false drop probability may be obtained. Later we will discuss these alternatives.

AFD and IFD are extreme cases for FD estimation. IFD requires more information about the database instance than AFD, but would provide a more accurate estimation of FD than AFD. Example FD computations are provided in Figure 6.1 for an intuitive explanation.

Case I: $N = 2$, $F = 200$, $D_1 = 25$, $D_2 = 35$, $D_{avg} = 30$, $S = 5$, $t = 1$, Standard deviation of $D = 5$

Case II: $N = 2$, $F = 200$, $D_1 = 20$, $D_2 = 40$, $D_{avg} = 30$, $S = 5$, $t = 1$, Standard deviation of $D = 10$

AFD	IFD: Case I	IFD: Case II
$FD = 2 \cdot (1-(1-5/200)^{30})^5$ $FD = 2 \cdot 0.04266$ $FD = 0.0853$	$FD = (1-(1-5/200)^{25})^5 +$ $(1-(1-5/200)^{35})^5$ $FD = 0.0227 + 0.0701$ $FD = 0.0928$	$FD = (1-(1-5/200)^{20})^5 +$ $(1-(1-5/200)^{40})^5$ $FD = 0.0099 + 0.1047$ $FD = 0.1146$

$$Deviation\ for\ Case\ I = 100 \cdot \frac{0.0928-0.0853}{0.0853} = 8.79\%$$

$$Deviation\ for\ Case\ II = 100 \cdot \frac{0.1146-0.0853}{0.0853} = 34.35\%$$

Figure 6.1. Example *FD* computations by using average *D* and individual *D* values.

In the example of Figure 6.1, the FD value computed by AFD is less than the FD value computed by IFD. Although D_{avg} values are the same for both cases, different FD values are obtained for IFD. The difference between the FD values of AFD and

IFD would increase if individual D values of the records show sharper deviations from D_{avg} value (i.e., for higher standard deviation for D values).

We propose a new method to estimate FD which covers AFD and IFD as special cases. In the proposed FD computation method, a database is conceptually divided into p sub-databases, horizontal partitions, according to D_r ($1 \le r \le N$) values of the records. We call the proposed FD estimation method Partitioned False Drop estimation method (PFD). Graphical representations of estimating FD with AFD and PFD for SSF are illustrated in Figure 6.2. A darker area indicates a record with a higher D value (we assume that both methods use the same S value for all records). In the rest of this section, we provide formal definition for conceptual partitioning of a database and FD estimation with PFD.

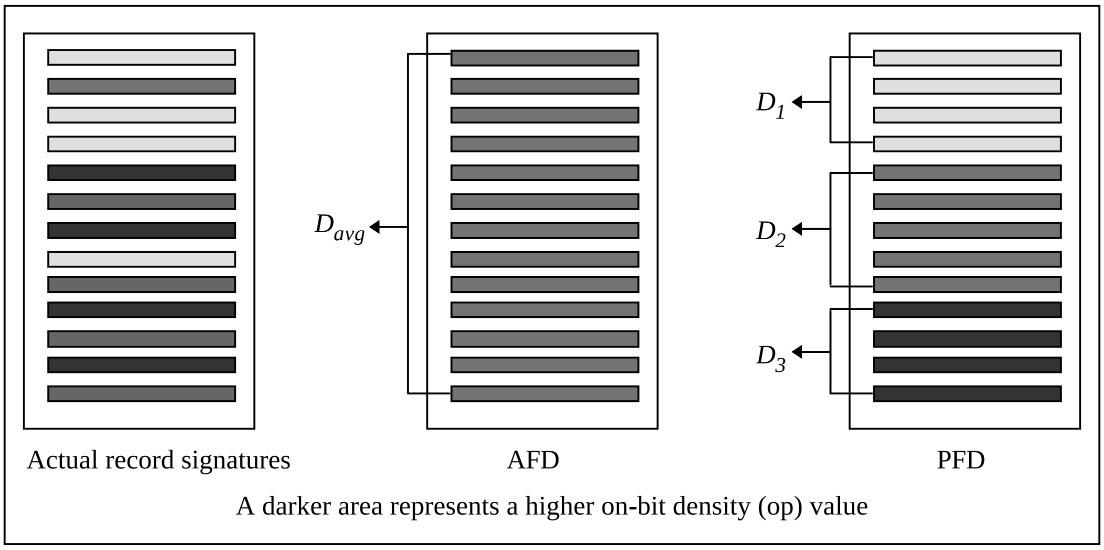


Figure 6.2. Graphical representations of estimating FD with AFD and PFD for SSF.

A database (DB) is a set of N records such that $DB = \{R_1, R_2, \cdots, R_N\}$. The domain of D values (Dom) of a database contains integers between 1 and the maximum of D values, D_{max}. Since records with no term never accessed, we excluded the case $D = 0$. Dom is divided into p disjoint sub-domains and each sub-domain is assigned to a partition. A sub-domain Dom_i is a subset of Dom and it is defined by a lower (L_i) and an upper bound (U_i) specifying the members of the sub-domain as follows.

$$Dom_i = \{D \mid L_i \le D \le U_i\} \quad for \;\; 1 \le i \le p \tag{6.8}$$

The lower and upper bounds of the sub-domains satisfy the following conditions.

$$L_1 = 1 \text{ and } U_p = D_{\max} \tag{6.9}$$

$$L_i \le U_i \quad for \quad 1 \le i \le p \tag{6.10}$$

$$L_i = U_{i-1} + 1 \quad for \;\; 1 < i \le p \tag{6.11}$$

A partition P_i is a subset of DB and it is defined as

$$P_i = \left\{ R_r \mid R_r \in DB \wedge D_r \in Dom_i \right\} \;\; for \;\; 1 \le i \le p\,. \tag{6.12}$$

The rth record with D_r distinct terms is assigned to the ith partition, P_i, if $L_i \le D_r \le U_i$. A record can be member of at most one partition, i.e., the partitions are mutually exclusive, and a record is always assigned to a partition.

Each partition is considered as a separate signature file with its own average number of distinct terms. The estimated number of false drops in ith partition, FD_i, is computed as follows.

$$FD_i = N_i \cdot (1 - (1 - S/F)^{AD_i})^{W(Q)_t} \quad for\; 1 \le i \le p \tag{6.13}$$

where N_i and AD_i are the number of records and the average number of distinct terms in the ith partition, respectively. If C_d is the number of records containing d distinct terms, then N_i and AD_i are computed as follows.

$$N_i = \sum_{d=L_i}^{U_i} C_d \tag{6.14}$$

$$AD_i = \frac{\sum_{d=L_i}^{U_i} C_d \cdot d}{N_i} \tag{6.15}$$

The estimated number of false drops for the whole database, *FD*, is computed by adding the estimated false drop values of the partitions.

$$FD = \sum_{i=1}^{p} FD_i \tag{6.16}$$

If there is only one partition, i.e., $p = 1$, $N = N_1$, and $D_{avg} = AD_1$, then Equation (6.16) reduces to Equation (6.6). Therefore, AFD is a special case of PFD. If the domain of each partition contains only one *D* value, i.e., $p = D_{max}$, then $L_i = U_i = AD_i$

= i for $1 \le i \le D_{max}$. Consequently, Equations (6.14), (6.15), and (6.16) can be combined and rewritten as follows.

$$FD = \sum_{d=1}^{D_{\max}} C_d \cdot (1-(1-\tfrac{S}{F})^d)^{W(Q)_t} \tag{6.17}$$

Note that, Equation (6.17) is a simpler and more efficient form of equation (6.7). It considers the number of terms in each record individually. Therefore, IFD is also a special case of PFD. Since AFD and IFD are two extreme cases in estimating FD, to show the performance increase obtained by PFD more clearly, we take $p = D_{max}$ in the rest of this chapter, i.e., we used IFD for FD estimation.

6.3 Using PFD in Sequential Signature Files

The sequential signature file (SSF) method requires retrieving the whole signature file for each query [CHR84b]. Although the signature file occupies less space than the original records, except for small databases, the response time of SSF is still very high. For example, without any seek requests, just reading a SSF for F = 1200 and N = 10^6 requires 1.76 minutes ($(5.77 \cdot (1200 \cdot 10^6)/(8192 \cdot 8))/(1000 \cdot 60)$ where 5.77 is the time required to read a disk block in ms and 8192 is the size of a disk block in bytes). However, small databases or small subsets of a database may be searched using SSF efficiently. For example, the two level access method [SAC83, SAC85] partitions a signature file horizontally such that the signatures of each partition fit into a disk block and the signatures are stored sequentially in the disk blocks. For query evaluation each qualifying disk block is searched sequentially. Therefore, we inspected the effect of using AFD and PFD with SSF on small databases.

To minimize the number of seek operations to access the actual records, the record pointers are stored along with the signatures. Each record pointer holds the position of the corresponding actual record and it occupies four bytes (P_{size}). We are not concerned marginal improvements which will be valid for both FD estimation methods. For example, other alternatives to store the record pointers, such as storing the offsets between the positions of the records (i.e., run lengths) instead of actual positions of the records, are not considered.

For a given F and D values, the value of S that satisfies the optimality condition is computed by using Equation (6.1). For varying record lengths, the value of S is determined by using D_{avg} instead of D in Equation (6.1). Note that the signatures of the records with $D_r > D_{avg}$ $(1 \leq r \leq N)$ will contain more on-bits than off-bits, i.e., the optimality condition may not hold for all record signatures. We will refer to this method as AFD sequential signature file (AFD-SSF) method.

For a given space overhead, since the whole signature file must be retrieved and processed for query evaluation, the time required to process the SSF will be approximately the same for all F and S values (note that F values will be adjusted to match the byte boundaries and therefore total signature file sizes may be different). Therefore, minimizing the observed FD will also minimize the response time for SSF. The false drop probability and FD are minimized when the optimality condition is satisfied [CHR84a, ROB79]. However, for the databases with varying record lengths the optimality condition will be violated by some records if the same F value is used for all records. We list three signature generation alternatives for SSF to obtain a lower FD with the same space overhead. To compare their performance, we adjusted the F values to obtain the same space overhead, OV bits, for all methods.

1. Fixed F-Fixed S (FFFS): The same F ($F = \frac{OV}{N}$) value is used for all records. A single S value is used for all partitions that minimizes FD estimated with PFD. The S value which provides the minimum FD is determined by a linear search in the domain of S. The lower bound of the search space is one. For given D and F values, the optimality condition defines an upper bound for S that is

$$\max(F \cdot \ln 2 / AD_i \;\; for \;\; 1 \leq i \leq p).$$

Since the value of S must be an integer, the number of possible S values will be small.

2. Fixed F-Varying S (FFVS): The same F ($F = \frac{OV}{N}$) value is used for all records. A different S_i ($1 \leq$ i $\leq p$) value is used for each partition. The value of S_i ($1 \leq$ i $\leq p$) for the ith partition is determined by using AD_i, i.e., average number of terms in per record in ith partition, in Equation (6.1). For large D_{max} values a second level partitioning may be defined for each first level partition and the values of S_i (1 ≤ i ≤

p) for the first level partitions may be determined by applying the FFFS method to the second level partitions. If p = 1, i.e., there is only one partition, this method converges to AFD-SSF. For query evaluation a different query signature must be generated for each partition.

3. Varying F-Varying S (VFVS): A different F_i ($1 \leq i \leq p$) value is used for all partitions. The F values of the partitions are determined as follows.

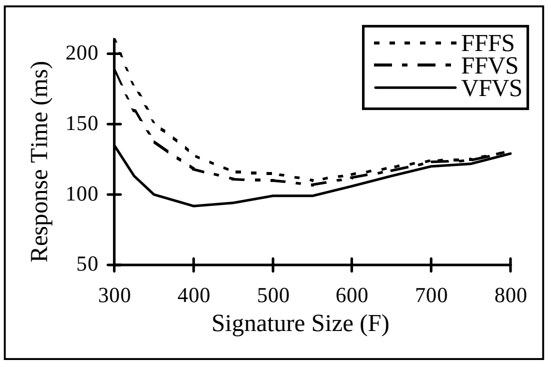
$$F_i = \frac{OV}{N \cdot D_{avg}} \cdot AD_i \qquad for \;\; 1 \leq i \leq p \tag{6.18}$$

where $\frac{OV}{N \cdot D_{avg}}$ is the number of bits used to represent a term in the signature file. A different S_i ($1 \leq i \leq p$) value is used for each partition. The value of S_i ($1 \leq i \leq p$) for the ith partition is determined by using AD_i, i.e., average number of terms in this partition, and F_i determined with Equation (6.18) in Equation (6.1). For large D_{max} values the second level partitioning method explained above can be used to obtain a lower FD. Note that there may be other strategies to determine F values of the partitions that may provide a lower FD than our method.

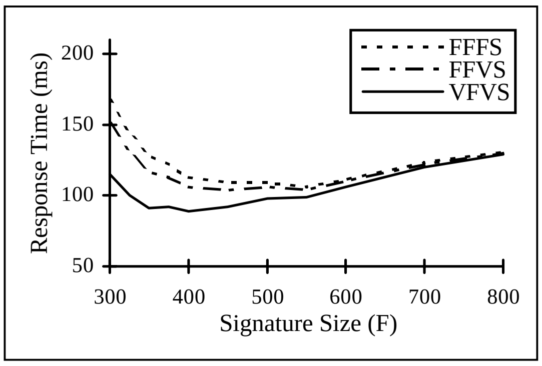
We exclude Varying F-Fixed S (VFFS) case since providing the same space overhead with other methods is difficult. Also, VFVS case covers VFFS.

The expected response time values of FFFS, FFVS, and VFVS methods obtained by simulation runs are plotted in Figure 6.3. The test database contains randomly selected 1000 records of the original test database and all methods estimate FD with PFD. To show the maximum performance improvement that can be obtained by VFVS we plotted the improvement percentage obtained by VFVS over FFFS in terms of response time in Figure 6.3.d.

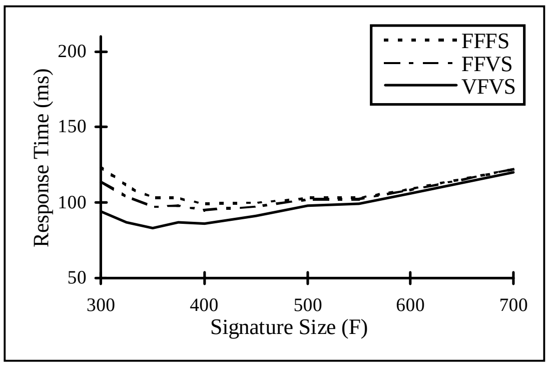
Time required to process the signature file is approximately the same for all methods since signature file sizes are practically the same (i.e., show negligible variations due to byte boundary alignment for signatures). Therefore, the differences among the response times are incurred due to the FD values. FFVS and VFVS perform better than FFFS since they guarantee the optimality condition for all record signatures and obtain lower false drop probability. VFVS outperforms FFVS since it uses the storage space more efficiently than FFVS by adjusting F according to the number of the terms in the records. FD values decrease for increasing signature size while the time required to process the signature file increases. Therefore, the time required to resolve these false drop records become insignificant compared to process the signature file and hence the IP values decrease for increasing *F* values.



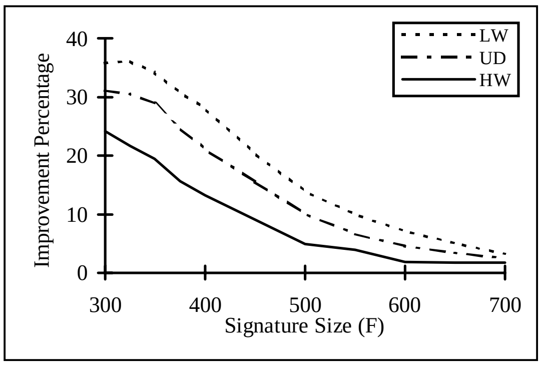
a. LW Query Case



b. UD Query Case



c. HW Query Case



d. IP(FFFS, VFVS)

(*SP* = 1.0, *N* = 1000).
Figure 6.3. Response time values of FFFS, FFVS, and VFVS methods obtained by simulation runs and IP(FFFS, VFVS) versus *F*.

The query evaluation with FFFS is similar to AFD-SSF. The only difference is that they determine the value of *S* differently and may use different S values in signature generation. Both methods generate a single signature for each query and use the same signature size for all records. Consequently, the implementation techniques of these methods will have minimal effect on the response time. Also, FFFS is the worst one among FFFS, FFVS, and VFVS. Therefore, we selected FFFS as the representative of partitioned FD estimation methods for SSF and we compared the performance of it with AFD-SSF. We will refer to FFFS as PFD-SSF. (Note that from the viewpoint AFD-SSF this is a more-than-fair comparison, since we are using the worst case of PFD as its representative.) We tested the performance of AFD-SSF and PFD-SSF on a database with 1000 records. The expected (denoted by Exp) and the observed (denoted by Obs) average false drop values of both methods for varying *F* values are given in Table 6.1.

Table 6.1. Expected (Exp) and Observed (Obs) Average False Drop Values for AFD-SSF and PFD-SSF

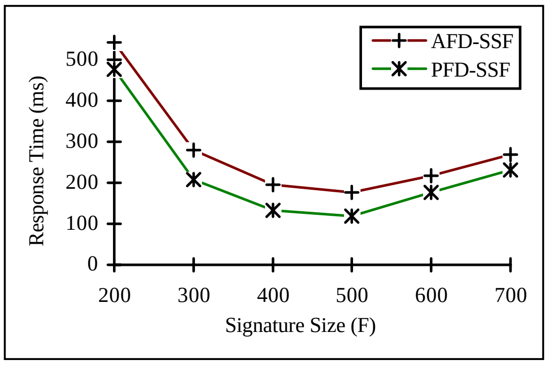
	LW Query Case				**UD Query Case**				**HW Query Case**			
	AFD-SSF		**PFD-SSF**		**AFD-SSF**		**PFD-SSF**		**AFD-SSF**		**PFD-SSF**	
F	**Exp**	**Obs**	**Exp**	**Obs**	**Exp**	**Obs**	**Exp**	**Obs**	**Exp**	**Obs**	**Exp**	**Obs**
200	8.10	12.76	12.49	11.98	5.42	8.62	8.61	8.13	2.75	4.76	4.73	4.33
300	1.29	4.07	3.38	3.28	0.86	2.80	2.31	2.19	0.43	1.62	1.24	1.18
400	0.21	1.59	1.11	1.04	0.14	1.13	0.76	0.65	0.07	0.67	0.40	0.34
500	0.03	0.96	0.44	0.43	0.02	0.64	0.30	0.28	0.01	0.43	0.15	0.13
600	0.01	0.54	0.19	0.18	0.00	0.38	0.13	0.11	0.00	0.23	0.07	0.05
700	0.00	0.40	0.09	0.09	0.00	0.27	0.06	0.05	0.00	0.16	0.03	0.03

Expected FD values of AFD is always less than the observed FD values of this method. Another important result is that the observed FD values of PFD-SSF is always less than the observed FD values of AFD-SSF. Finally, the expected and observed average FD values for PFD-SSF are very close for all query cases. This shows that PFD estimates FD more accurately than AFD.

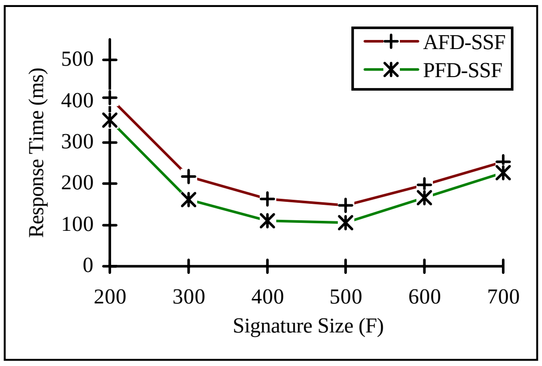
The observed response time values versus *F* are plotted in Figure 6.4. In (d) the improvement percentage obtained by PFD-SSF over AFD-SSF in terms of response time is plotted.

The observed FD values, hence the time required to resolve the false drop records, decreases for increasing *F* value. Since the size of the signature file increases for increasing *F* value, the time required to process the signature file also increases. The decrease in FD becomes negligible after a certain *F* value while the increase in the time required to process the signature file increases almost linearly. Therefore, the response time decreases for increasing *F* value for small *F* values and starts to increase after a certain *F* value (for example, *F* = 500 for LW query case). We call this point optimum *F* (space overhead) for a database instance. Since the observed FD diminishes more rapidly for increasing query weights, the optimum *F* value decreases as the query weight increases.

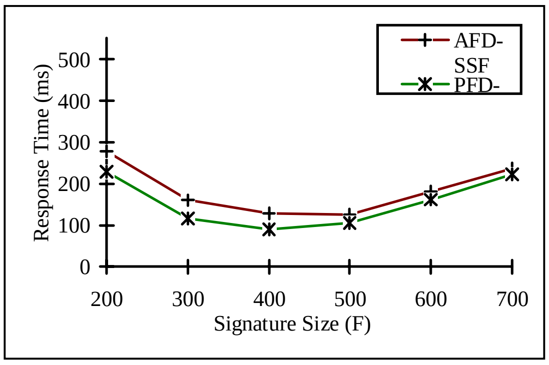
Using SSF with an *F* value greater than the optimum *F* value is meaningless. Also, the response time values for small *F* values are very high. Therefore, we can assume the IP values around the optimum *F* values as the performance increase obtained by using PFD for SSF. The IP values obtained at optimum *F* values are 33%, 32%, and 30% for LW, UD, and HW query cases, respectively.



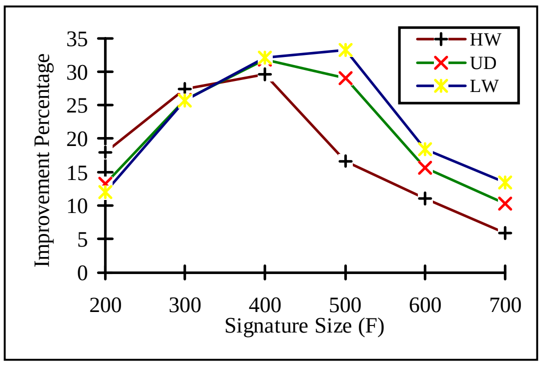
a. LW Query Case



b. UD Query Case



c. HW Query Case



d. IP(AFD-SSF, PFD-SSF)

(*SP* = 1.0, *N* = 1000)

Figure 6.4. Observed response time versus *F* for AFD-SSF and PFD-SSF.

6.4 Using PFD in Generalized Frame-Sliced Signature Files

In the GFSSF method, a signature is divided into k $(1 \le k \le F)$ equal sized frames. To obtain a term signature, n $(1 \le n \le k)$ frames are selected among k frames and m $(1 \le m \le F/k)$ bits are set to "1" in each selected frame [LIN92]. A heuristic search algorithm was provided in [LIN92] to obtain the values of parameters k, n, and m for given t (number of terms in a query), N, D_{avg}, and F values. The objective of the algorithm is to minimize the response time instead of the false drop probability. Therefore, the optimization algorithm adjust the signature file parameters such that FD is reduced to an optimum value with minimum signature file processing. Consequently, the expected response time can be achieved if the observed FD is close to the expected FD value.

A formula to estimate the false drop probability, *fd*, for single term queries for given *F*, *N*, D_{avg}, *n*, *k*, and *m* values was provided in [LIN92]. The false drop probability was approximated as $fd^{\,t}$ for the queries containing more than one term where *t* is the number of terms in the query. Since the average number of terms per record, D_{avg}, was used in FD estimation, we call this method AFD-GFSSF.

To estimate FD by using PFD, we use the *fd* estimation formula of [LIN92] *p* times as follows.

$$FD = \sum_{i=1}^{p} N_i \cdot GFSSF\_\ fd(F,k,n,m,AD_i) \tag{6.19}$$

where $GFSSF\_fd(F,k,n,m,AD_i)$ denotes the false drop estimation formula provided in [LIN92]. The ith usage of GFSSF_fd computes the false drop probability for the ith partition. Multiplying this false drop probability with the number of records in this partition gives the expected number of false drops in the ith partition. We will refer to this method as PFD-GFSSF.

The same heuristic search algorithm given in [LIN92] was used to obtain the optimized configurations of AFD-GFSSF and PFD-GFSSF. The algorithm starts with a random configuration and tests all neighbor configurations obtained by increasing or decreasing the values of the parameters *n*, *s* (*F/k*), *m* by one. Like in [LIN92], we also confined the values of *s* (*F*/*k*) to exact multiples of 8 for easy processing of the resultant signature file. A GFSSF configuration with *s* = 1 (each frame is a bit-slice) becomes a B'SSF [LIN92]. B'SSF is a special case of MFSF [KOÇ95b] and we will inspect the MFSF method in Section 6.5.

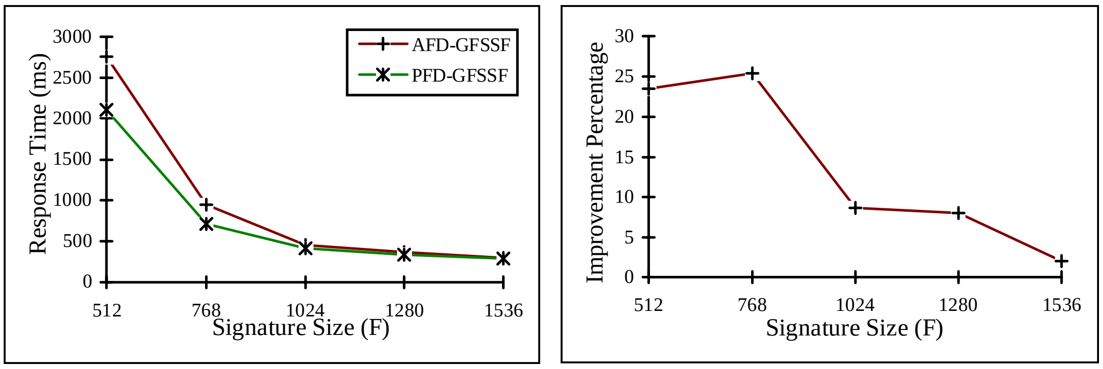
The optimization method defined for GFSSF expects a specific number of query terms. Also, the false drop probabilities of the queries containing more than one term is computed approximately. Therefore, for accuracy we used 1000 randomly generated zero hit single term queries to test the performance of AFD-GFSSF and PFD-GFSSF.

The expected (Exp) and the observed (Obs) FD values for *N* = 20,000 are given in Table 6.2 (In the experiments we used a part of our test database due to long time requirements of the GFSSF file structure generation and query evaluation. Furthermore, *N* = 20,000 provides us the necessary observations.). The optimized configurations of AFD-GFSSF and PFD-GFSSF may be different. Consequently, the time required to process the signature file may be different for these methods. For this reason the method that obtains a lower FD for a given database instance may obtain a higher response time than other method. Therefore, we also provide the corresponding observed response time values in Figure 6.5.

Table 6.2. Expected (Exp) and Observed (Obs) Average False Drop (FD) Values for AFD-GFSSF and PFD-GFSSF (N = 20,000)

	AFD-GFSSF		**PFD-GFSSF**	
F	**Exp**	**Obs**	**Exp**	**Obs**
512	6.12	32.54	31.56	26.66
768	0.61	6.61	5.56	4.15
1024	0.40	1.70	1.23	1.28
1280	0.13	1.14	0.75	0.84
1536	0.22	0.86	0.74	0.82

The observed FD values of PFD-GFSSF are always less than the observed FD values of AFD-GFSSF. Since these lower FD values are obtained with less processing time, PFD-GFSSF performs better than AFD-GFSSF. The differences between the observed FD values of AFD-MFSF and PFD-MFSF for the same F value decrease for increasing F and diminish for $F \geq 1024$. As a result, the time required to resolve these false drop records decreases for increasing F. This causes a decrease in improvement percentage for increasing F.



a. Response Time b. IP(AFD-GFSSF, PFD-GFSSF)

(SP = 1, N = 20,000)

Figure 6.5. Observed response time versus F for AFD-GFSSF and PFD-GFSSF.

For F < 768, the observed FD values and the response time values are very high. Therefore, for N = 20,000, using a signature file with F < 768 is impractical. Depending on the space overhead, PFD-GFSSF obtains up to 25.73% response time improvements over AFD-GFSSF. Experiments with other *N* values show that FD diminishes at higher *F* values for increasing *N* values. Therefore, we obtained better IP values for the same *F* value for increasing *N* values. This shows that the space overhead of GFSSF must be increased to obtain an acceptable response time for increasing *N*.

6.5 Using PFD in Multi-Fragmented Signature Files

The MFSF method is defined in Chapter 5. The FD computation method proposed in Chapter 5 corresponds the AFD method. Therefore, we call the original method AFD-MFSF. To use PFD in MFSF, we rewrite the stopping condition given in Equation (5.14) as follows.

$$T_{slice} = (TFD_i - TFD_{i+1}) \cdot T_{resolve} \tag{6.20}$$

where T_{slice} is the time required to read and process a bit slice, TFD_i is the number of expected false drops after processing *i* bit slices (where TFD stands for Total number of False Drops), and $T_{resolve}$ is the time required to resolve a false drop record by accessing to the actual record. In Equation (6.20), $TFD_i - TFD_{i+1}$ gives the number of expected false drops which will be eliminated if we process the i+1st bit slice after processing *i* bit slices. At the stopping step the time required to process a bit slice becomes greater than or equal to the time required to resolve these false drops by accessing the actual records. Therefore, the signature file processing stops at this step.

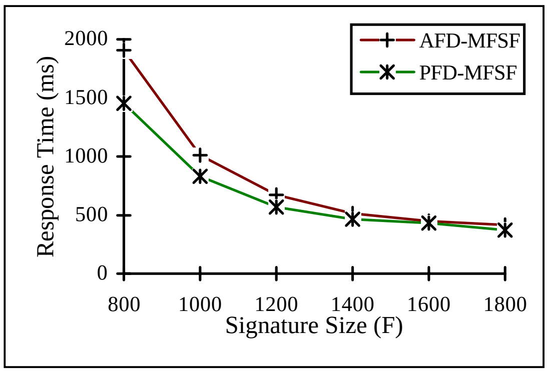
Each partition can be considered as a separate MFSF file and the number of expected false drops can be computed in each partition by using the formulas provided in Chapter 5. We compute TFD_i as follows.

$$TFD_i = \sum_{r=1}^{p} MFSF\_FD(i, f, F, S, N_r, AD_r) \tag{6.21}$$

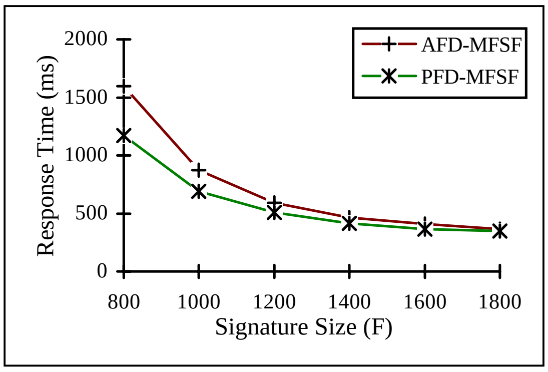
where N_r is the number of records in the kth partition and AD_r is the average number of terms in a record of the kth partition. MFSF_FD denotes the FD computation method defined in Chapter 5 and it computes the number of expected false drops after processing *i* bit slices (see Equation 5.8). Note that, for *p* = 1, the proposed FD computation method converges to the FD computation method of MFSF given in Chapter 5.

We used the heuristic search algorithm given in Figure 5.2 to search the optimum PFD-MFSF configuration by using Equation (6.21) for FD estimation.

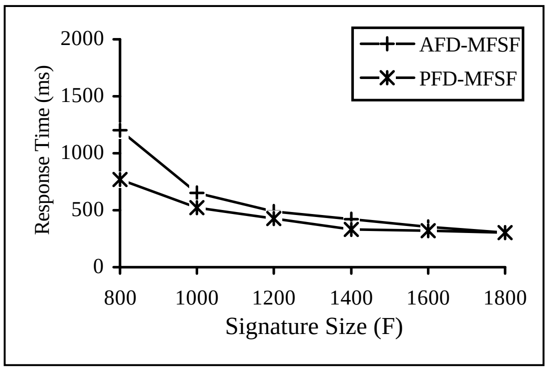
The expected (Exp) and the observed (Obs) FD values for N = 152,850 are given in Table 6.3. Since FD values are estimated differently, the stopping conditions of AFD-MFSF and PFD-MFSF may require processing different number of bit slices for each method. Consequently, signature file processing times may be different. Therefore, we also provide the corresponding observed response time values in Figure 6.6.



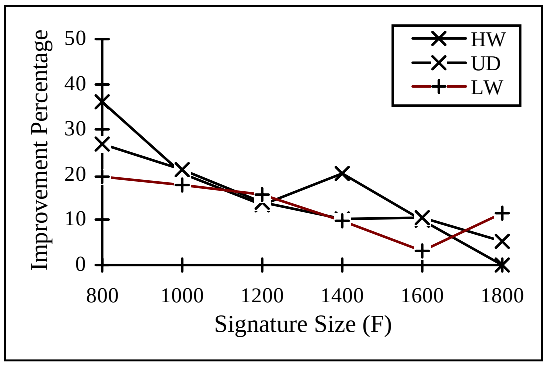
a. LW Query Case



b. UD Query Case



c. HW Query Case



d. IP(AFD-MFSF, PFD-MFSF)

(SP = 1, N = 152,850)
Figure 6.6. Observed response time versus F for AFD-MFSF and PFD-MFSF.

Like PFD-SSF and PFD-GFSSF, the observed FD values of PFD-MFSF are always less than the observed FD values of AFD-MFSF. Additionally, PFD-MFSF estimates FD more precisely and obtains these smaller observed FD values with less response times. Therefore, the PFD-MFSF method outperforms the AFD-MFSF method. Depending on the space overhead, PFD-MFSF obtains up to 20.24%

response time improvements over AFD-MFSF (since the observed FD values are very high, we considered the space overheads with *F* < 1000 as practically unusable).

Table 6.3. Expected (Exp) and Observed (Obs) Average False Drop Values for AFD-MFSF and PFD-MFSF

	LW Query Case				UD Query Case				HW Query Case			
	AFD-MFSF		PFD-MFSF		AFD-MFSF		PFD-MFSF		AFD-MFSF		PFD-MFSF	
F	Exp	Obs	Exp	Obs	Exp	Obs	Exp	Obs	Exp	Obs	Exp	Obs
800	0.80	12.48	8.90	8.78	0.89	10.45	6.75	6.25	0.61	7.53	4.32	3.50
1000	0.60	5.37	3.43	3.40	0.64	4.82	2.79	2.45	0.47	3.11	2.26	2.59
1200	0.54	2.92	2.00	1.92	0.43	2.57	1.64	1.71	0.29	1.37	1.31	1.22
1400	0.43	2.15	1.02	1.19	0.30	1.50	1.07	1.03	0.35	1.45	0.72	0.55
1600	0.43	1.51	0.87	1.17	0.35	1.26	0.68	0.75	0.27	0.97	0.62	0.56
1800	0.37	1.31	0.66	0.72	0.30	1.02	0.64	0.72	0.21	0.60	0.49	0.53

Mathematical model of GFSSF shows that the frame size of GFSSF decreases for increasing database size and GFSSF converges to B’SSF which is a special case of GFSSF. For increasing number of query terms, GFSSF converges to B’SSF at smaller database sizes. Since B’SSF is also a special case of MFSF and MFSF obtains better response times in multi-term query environments, we will use only the MFSF method in the following analysis.

6.6 The Effect of Distribution of Record Lengths

The example FD computations given in Figure 6.1 show that the difference between FD values estimated with AFD and IFD increases if the difference between the number of terms increases. This shows that the performance increase obtained by PFD is affected by the distribution of the record lengths. Therefore, we inspected the effect of record length distributions experimentally and analytically.

To test the effect of the distribution of the record lengths experimentally, we obtained four test databases with different record length distributions by selecting 100,000 records from the test database. (See Figure 3.1 for the record length distribution of the original test database.) To obtain a lower STD value, we deleted some long and some short records. Similarly, to obtain a higher STD value, we deleted some records near the peak point of the normal distribution (we flattened the peak point).

The IP values for all query cases are nearly equal at F = 1200 (see Figure 6.6.d). Therefore, we optimized the test databases at F = 1200 and for the UD query case. The results of the experiments with these test databases are given in Table 6.4. The last column, IP, of Table 6.4 represents the improvement performance values, IP(AFD-MFSF, PFD-MFSF), computed using the observed response time values.

Table 6.4. FD and Response Time Values for Changing STD Values

	AFD-MFSF				**PFD-MFSF**				
	FD		**RT (ms)**		**FD**		**RT (ms)**		
STD	**Exp**	**Obs**	**Exp**	**Obs**	**Exp**	**Obs**	**Exp**	**Obs**	**IP**
10.93	0.36	0.97	250	274	0.64	0.51	289	243	11.31
11.33	0.36	1.91	250	430	1.25	1.14	344	359	16.51
12.57	0.30	2.35	254	467	1.33	1.20	368	380	18.63
12.69	0.30	2.40	255	473	1.36	1.24	371	385	18.60

The optimization algorithm of AFD-MFSF expects similar FD values for all cases. This is the objective of the optimization algorithm that tries to reduce the estimated FD value to the optimum value. However, the observed FD values increases for increasing STD value. This shows that the error in the estimation of FD with AFD increases for increasing STD values.

All of the expected and observed FD values for PFD-MFSF are very close. The optimization algorithm adjusts the parameters of the MFSF file properly by using this accurate FD estimation. This provides obtaining lower observed FD values with the PFD-MFSF method than the AFD-MFSF method. This reduction in FD is obtained without increasing the observed response time obtained with the PFD-MFSF method. Also, the differences between the observed response time values of AFD-MFSF and PFD-MFSF increase for increasing STD. Consequently, the IP values increases for increasing STD value. The above experiments performed with real data show that PFD is effective for normal distributions of record lengths.

To test the performance of the approach in another extreme we consider uniform record length distributions. The parameters that affect the performance of PFD-MFSF in a uniform distribution of record lengths are the minimum and the maximum number of terms in the records. We performed five experiments with simulation runs on conceptual databases having different maximum D values. To make results comparable, we adjusted the F value for each case to obtain the same space overhead (1200/25.7 = 46.7 bits/term, where 25.7 is the D_{avg} value for the test database). The results of these simulation runs are given in Table 6.5. We give the results obtained by PFD-MFSF since PFD estimates FD accurately.

Table 6.5. Results of the Simulation Runs for PFD-MFSF with Uniform Distributions of Record Lengths (N = 100,000, UD Query Case)

Minimum D	**Maximum D**	**Average D**	**STD**	**F**	**Expected FD**	**Expected RT (ms)**
1	10	5.50	2.87	257	0.75	316
1	25	13.00	7.21	607	0.83	321
1	50	25.50	14.43	1191	0.85	323
1	100	50.50	28.87	2358	0.74	324
1	200	100.50	57.73	4693	0.75	324

The estimated response time for increasing maximum D values increases slightly. On the other hand, since the STD values increases as the number of possible D values increases, the performance improvement obtained by the PFD-MFSF with respect to AFD-MFSF also increases.

A possible cause of a performance degradation may be storing records from diverse sources in the same database. For example, storing the records that contain only abstracts with the records that contain full text of a document in the same database. In this case the distribution of the record lengths will have more than one peak value. For such record length distributions, we propose dividing the record file and the signature file physically with respect to the record lengths. This can be considered as another implementation of the conceptual partitioning of records proposed in PFD.

6.7 Dynamic Databases

In a dynamic environment N_i, probably AD_i, values ($1 \le i \le p$) will change as new records are added to a database. Therefore, the performance of the system may degrade and a reorganization for the signature file may become necessary. During reorganization, if required, the signature size may be increased to obtain an acceptable performance. For simplicity we only consider the addition of records and inspect the reorganization frequency for PFD-MFSF and measure possible performance degradation experimentally if a necessary reorganization is delayed.

An increase in the number of records will cause an increase in FD if no additional bit slices are used during query evaluation. We store the number of records in the partitions in main memory and we update them during record insertion with negligible space and processing overheads (the number of partitions will be small). Therefore, we always use the current database instance in determining the number of bit slices to be used for query processing. By this way the query processor may use additional bit slices to keep FD near the optimum value for increasing *N*.

The optimization algorithm adjusts the parameters of MFSF such that the minimum response time is obtained by allowing a small FD. Using the same MFSF configuration after adding a few records to the database, i.e., increasing *N* slightly, will cause a small increase in FD. Until this increase in FD causes a noticeable increase in the response time, the optimization algorithm still finds approximately the same configuration as the optimum one. Therefore, using the same MFSF configuration for increasing *N* will cause no performance degradation until the optimization algorithm finds a different MFSF configuration. A reorganization becomes necessary to obtain better response time when the optimization algorithm changes the signature file configuration.

We always observed more FD than the estimated value with AFD. As a result of this, the optimization algorithms that use AFD also estimate the response time less accurately and find sub-optimal configurations. For this reason, deciding a reorganization analytically will be misleading for AFD-MFSF. However, PFD-MFSF estimates FD accurately and we can determine necessary reorganization points with simulation runs as the database grows.

The value of *N* that requires a reorganization can be determined by using the current instance. Usually, the records added to a database expose similar characteristics with the records already in the database. Therefore, if the database is sufficiently large, adding new records will cause insignificant variations in the percent of records in the partitions and AD_i values. As the difference between L_i and U_i values of a partition decreases, the change in AD_i value due to new records added to that partition will also decrease. In the extreme case where $L_i = U_i$, the AD_i value never changes ($L_i = U_i = AD_i$). We assume the percent of the records in the partitions will not change as database grows and project the number of records in the partitions for a target instance by using current instance as follows.

$$N'_i = N' \cdot \frac{N_i}{N} \quad for\ 1 \le i \le p \tag{6.22}$$

where N is the number of record in the current database instance and N' is the number of records in the target database instance.

Starting from N = 10,000, we inspected the configurations obtained for LW, UD, and HW query cases for increasing N values up to N = 300,000. The target database instances were obtained by projecting the database instance that contains the first 100,000 records (see Equation 6.22). For F = 1200, the values of N where a reorganization is needed are given in Table 6.6.

Table 6.6. *N* Values that Requires a Reorganization for LW, UD, and HW Query Cases

Query Case	***N* Values (*1000) Where a Reorganization is Needed**
LW	10 - 20 - 40 - 50 - 60 - 150 - 220
UD	10 - 30 - 80 - 110 - 160
HW	10 - 60 - 150 - 280

The numbers in the second column of Table 6.6 indicates the N values where a reorganization is needed. For example, “10 - 20” for LW query case should be interpreted as the configuration obtained for N = 10,000 can be used with no performance degradation until N reaches 20,000. When N becomes 20,000 a fresh signature file must be created with new parameters. The frequency of a reorganization increases for decreasing query weight. The reason of this is that FD increases more rapidly for low weight queries as N increases. In all query cases, for N > 100,000 a reorganization is not needed at least for the next 50,000 records. The same is true for higher N values. Consequently, we can say that the reorganization need for PFD-MFSF is rare.

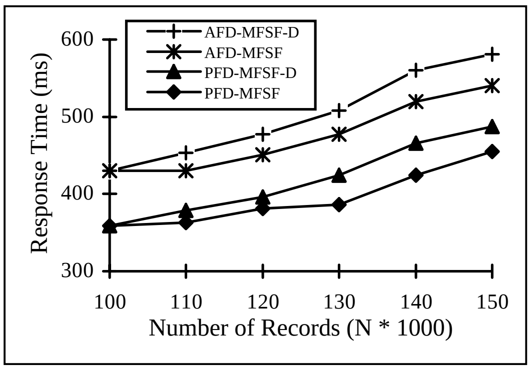
To measure the performance degradation incurred due to a delayed reorganization for AFD-MFSF and PFD-MFSF, we optimized the signature file parameters for F = 1200 and N = 100,000 and we continued to use the same configuration for increasing N values for UD query case. Note that, a reorganization is needed at N = 110,000 for UD query case and this configuration stays optimum until N becomes 160,000.

To simulate a dynamic database environment, we used the first 100,000 records of the test database as the current instance. For insertion of new records the remaining

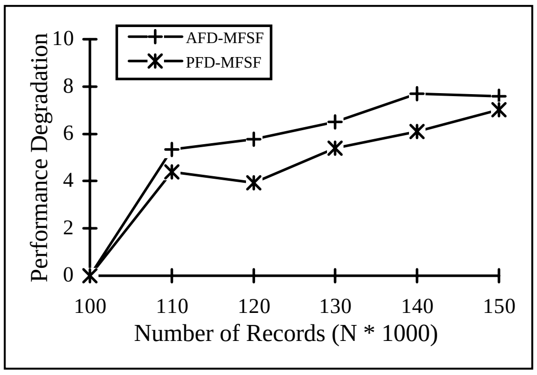
50,000 records were used in the order they were recorded to the original record file. For example, the database after 10,000 insertions, which corresponds a 10% increase in the number of records, contains the first 110,000 records of the test database. Also, we carried out the same experiments after reorganization. The observed response time values are plotted in Figure 6.7.a. A "D" at the end of a method indicates that the results are for the delayed reorganization of the method. In Figure 6.7.b the performance degradation values for increasing N values are plotted. The performance degradation of PFD-MFSF is computed as follows.

$$100 \cdot \frac{TR(PFD{-}MFSF{-}D) \; - \; TR(PFD{-}MFSF)}{TR(PFD{-}MFSF)}$$

The performance degradation values of AFD-MFSF are computed similarly.



a. Observed Response Time



b. Performance Degradation

(*SP* = 1, *F* = 1200, UD Query Case)

Figure 6.7. Observed response time values for delayed reorganizations.

The results given in Figure 6.7 show that delaying a reorganization will cause small performance degradation for both AFD-MFSF and PFD-MFSF methods. The performance degradation observed in PFD-MFSF are slightly less than the AFD-MFSF methods.

7. THE COMPRESSED MULTI-FRAGMENTED SIGNATURE FILE

In the record signatures of a BSSF the number of “1”s is equal to the number of “0”s since the optimality condition is satisfied. By assuming the on-bits are randomly distributed in the signature file, we can also assume that half of the bits in a bit slice of BSSF are also “1”. In this case, storing a bit slice as a bit string is the optimum storage method. However, usually, the number of “1”s in the bit slices of MFSF is less than the number of “0,”s since MFSF obtains better response times by allowing on-bit densities less than 0.5.

Lower on-bit density provides rapid elimination of false drops and therefore the optimal number of expected false drops is obtained in fewer bit slice processing. Reducing on-bit density while providing sufficient on-bits in query signatures is possible by increasing *F* (the number of hashing locations). However, increasing *F* also increases the space overhead if the bit slices are stored as they are without compression. We propose the Compressed Multi-Fragmented Signature File (C-MFSF) method that stores the bit slices of MFSF in a compressed form. The space overhead of C-MFSF with a larger *F* value is less than the space overhead of MFSF with a smaller *F* value. For example, the signature file sizes are 17.11 MBytes and 32.80 MBytes for C-MFSF with *F* = 15,000 and MFSF with *F* = 1800, respectively.

Data compression can be used to compress the records of full text databases [KLE89, BEL90, BEL93, MOF95b, ZOB95b]. Compressing the records in full text databases reduces the disk space used to store records and provides retrieval of actual records with fewer disk accesses. Therefore, record compression can also be used in addition to compressing the signature file to further improve the performance.

The organization of this chapter is as follows. Previous work that use compression in indexing is summarized in Section 7.1. In Section 7.2, the compression methods used in IR systems are given. We propose a new method to code the positions of “1”s in the bit slices of MFSF in Section 7.3. Description of C-MFSF and the results obtained with simulation runs are given in Section 7.4. In Section 7.5, the results of the experiments with real data are given. Section 7.6 contains the projection results obtained for very large databases. Finally, Section 7.6 contains a theoretical comparison of C-MFSF and the inverted file method.

7.1 Related Work

Data compression involves with transforming a string of bits in some representation into a new string that contains the same information (we are not concerned with lossy compression) but whose length is as small as possible [LEL87]. Using data compression in IR systems increases the performance by reducing the space overhead and the amount of data to be retrieved. However, additional processing time may be required to decode the encoded data.

Compressing the record signatures of sequential signature files is inspected in [FAL85b]. In this study, to obtain a lower false drop probability, record signatures are produced using large *F* and small *S* values. The resulting sparse record signatures are compressed. To compress record signatures the Run Length encoding (RL), bit-Block Compression (BC), and Variable bit-Block Compression (VBC) methods are used. It is observed that RL obtains the lowest false drop probability followed by BC [FAL85b]. However, Faloutsos prefers BC since it has good features of all the other methods.

Moffat and Zobel inspect a variety of index compression methods [MOF92]. Posting lists are compressed to reduce the space overhead and improve the performance of the inverted file method [MOF92, ZOB92]. It is shown that the space overhead of inverted files can be reduced to less than 10% of the space used by the original records [ZOB92]. Decoding long posting list may cause a bottleneck [ZOB92]. To solve this problem skips, an index on the entries of a posting list, are added to the compressed posting lists [MOF95]. Skips provide substantial time savings in searching a specific record number in the compressed inverted file entries with an additional small space overhead [MOF95].

In [MOF95a] the γ (gamma) code[ELI75], δ (delta) code [ELI75], and *Golomb* code [GOL66] are considered and their performances are compared. In [MOF95a], the authors reported that the improvement provided by other methods is relatively small and they prefer to use the Golomb code to compress posting lists of the inverted files. Since the on-bit densities of the bit slices of MFSF are higher than the on-bit densities of the posting lists of IF, the performances of these methods may change. Therefore, we tested the performances of these methods with the bit slices of MFSF. In the following section we briefly summarize these methods and in Section 7.3 propose a new coding method.

7.2 Compression Methods

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

A record number for a database containing *N* records can be represented with $\lceil \log N \rceil$ bits where log denotes the base 2 logarithm ($\lceil x \rceil$ indicates the smallest integer greater than or equal to *x*). Generally, the differences (*gap* or *run length encoding*) between the record numbers are smaller than the record numbers and they may be represented with fewer number of bits than $\lceil \log N \rceil$ bits. Therefore, instead of the record numbers the gaps are compressed [GAL75]. For example, the ascending sequence of record numbers "1, 7, 15, 23, 27" is represented as "1, 6, 8, 8, 4."

Elias defines a sequence of coding schemes that maps positive integers to binary *codewords* (compressed representation) [ELI75]. The γ code represents integer *x* in two parts. The first part contains $\lfloor \log x \rfloor$ ($\lfloor x \rfloor$ indicates the greatest integer less than or equal to *x*) "0"s followed by a "1" which represents $2^{\lfloor \log x \rfloor}$ (the number obtained by setting all bits except the higher order bit of *x* to "0"). For example, the first part for $x = 19$ is "00001" since $\lfloor \log 19 \rfloor = 4$. The second part contains the binary representation of $x - 2^{\lfloor \log x \rfloor}$ (the remaining part of *x* obtained by setting the high order bit of x to “0”). The second part for x = 19 is “0011” (note that binary representation of 19 is “10011”). Therefore, γ code represents integer x with $2 \cdot \lfloor \log x \rfloor + 1$ bits.

The δ code represents the number of bits in the first part of an γ code, $\lfloor \log x \rfloor + 1$, with an γ code, i.e., the number of bits in the first part of γ code is compressed. For example, there are five bits in the first part of the γ code of x = 19. Therefore, the first part of x = 19 for the δ code is “00101” (see Table 7.1). The second part of the δ code is the same with the second part of γ code. Therefore, the δ code represents integer x is with $\lfloor \log x \rfloor + 2 \cdot \lfloor \log(1 + \lfloor \log x \rfloor) \rfloor + 1$ bits.

Golomb [GOL66] divides an integer x into two parts, q and r, by using a parameter, b, as follows.

$$q = \left\lfloor \frac{x-1}{b} \right\rfloor \qquad r = x - q \cdot b - 1 \tag{7.1}$$

The first part, q, is represented with q “0”s followed by a “1”. The second part, r, is represented with either $\lceil \log b \rceil$ or $\lfloor \log b \rfloor$ bits depending on r. The parameter b is determined according to the on-bit density in the bit map representation of the posting lists as follows [WIT94].

$$b = \left\lceil \frac{\log(2 - op)}{-\log(1 - op)} \right\rceil \tag{7.2}$$

where op is the probability of a particular bit in the bit string being an on-bit. Some sample values of the γ, δ, and Golomb codes are given in Table 7.1 (the first and the second parts of the codewords are separated with space).

Table 7.1. Example γ, δ, and Golomb Codes

Gap	γ	δ	**Golomb (*b* = 6)**
1	1	1	1 00
2	01 0	010 0	1 01
3	01 1	010 1	1 100
4	001 00	011 00	1 101
5	001 01	011 01	1 110
15	0001 111	00100 111	001 100
19	00001 0011	00101 0011	0001 00
47	000001 01111	00110 01111	00000001 110
257	000000001 00000001	0001001 00000001	(42 “0”s)1 110

The performances of the aforementioned compression methods are affected by the distribution of the gaps and the value of *b* for the Golomb code. For example, the compressed representation of 257 with a Golomb code that use *b* = 6 requires 46 bits. Therefore, for a better compression a higher *b* value must be used if such gap values are possible. The distributions of gaps in the bit slices of MFSF generated with BLISS-1 for *op* = 0.011 and *op* = 0.042 are plotted in Figure 7.1. The y axis, "% of Covered Gaps," represents the percent of the gaps that have a gap length less than or equal to the maximum gap value plotted in the x axis. For example, 95.1% and 76.1% of the gaps have a length of 96 or less for *op* = 0.042 and *op* = 0.011, respectively.

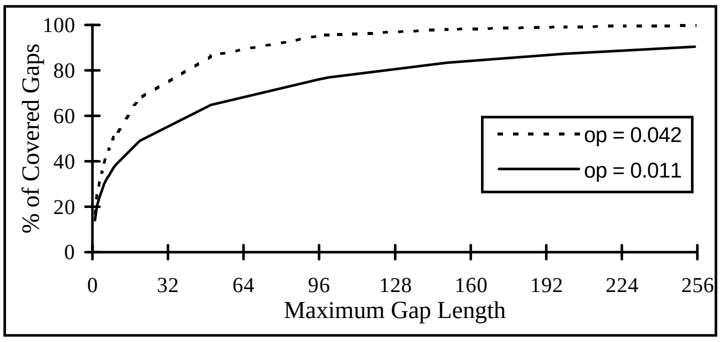


Figure 7.1. Distribution of the gaps in the bit slices of MFSF for on-bit densities 0.011 and 0.042.

The bitwise ANDing of bit slices is one of the fundamental operations in the BSSF query evaluation method. It requires obtaining the record numbers (or positions of on-bits) in the processed bit slice, i.e., decoding a compressed bit slice. The gaps, hence the record numbers, can be reconstructed from the compressed representation by scanning the compressed bit string from the left to the right. If codewords are in varying lengths, many shift and housekeeping operations may be required to decode a codeword. Therefore, in the framework of query processing with bit sliced signature file method, the best compression method may obtain a higher response time than other compression methods if decompression requires complex operations.

7.3 Fixed Code Compression Method

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

$$k = \left\lceil \log \frac{N}{OB} \right\rceil \tag{7.3}$$

where *N* is the number of records (number of bits in a uncompressed bit slice) in the database and *OB* is the average number of on-bits in a slice. Since $op = \frac{N}{OB}$, we rewrite Equation (7.3) by using *op* as follows.

$$k = \left\lceil \log \frac{1}{op} \right\rceil \tag{7.4}$$

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

FC can be explained with gaps as follows. A gap is represented with *k* bits if the gap is less than or equal $2^k - 1$. Otherwise, a codeword of length *k* with all “0” is used and $2^k - 1$ is subtracted from the gap value. The remaining part is coded with FC. Thus, a gap may be represented with more than one codeword. For example, the bitmap of the term T_4, “10100,” given in Figure 2.3 is represented in FC with *k* = 4 as “0001 0010” (the gaps for this bitmap are 1 and 2). Some sample gap values coded in FC with *k* = 4 and *k* = 8 are given in Table 7.2 (the codewords are divided with spaces).

Table 7.2. Example FC Codes with *k* = 4 and *k* = 8

Gap	**k = 4**	**k = 8**
1	0001	00000001
4	0100	00000100
5	0101	00000101
15	1111	00001111
16	0000 0001	00010000
47*	0000 0000 0000 0010	00101111
255*	(16 “0000”) 1111	11111111
257*	(17 “0000”) 0010	00000000 00000010

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

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

$$F \cdot k \cdot \left(\left\lfloor \frac{N \cdot (1-op)}{2^k - 1} \right\rfloor + N \cdot op \right) \tag{7.5}$$

bits. Since there are $F \cdot N \cdot op$ on-bits in the signature file, on the average, each on-bit is represented with

$$WC_k = k + \frac{k \cdot \left\lfloor \frac{N \cdot (1-op)}{2^k - 1} \right\rfloor}{N \cdot op} \tag{7.6}$$

bits in the worst case.

We compare the performance of γ, δ, Golomb, and FC on the bit slices of MFSF produced with BLISS-1. We prefer to use actual signature files rather than using artificially generated bit strings since the distribution of the on-bits in the signature file affects the result. The number of bits required to represent each on-bit for various *op* values for γ, δ, Golomb, and FC ("Obs "denotes the observed, "Best" denotes the best case, and "Worst" denotes the worst case behavior of FC) are given Table 7.3. For the Golomb code, an appropriate *b* value is computed for each bit slice by using the on-bit density of the slice in Equation (7.2). Similarly, for FC, a different *k* value is determined using Equation (7.4) for each slice. FC outperforms γ and δ codes and uses approximately one bit more than the Golomb code for small *op* values. Note that the observed number of bits required per on-bit for FC is approximately equal to the average of the best and the worst cases of FC.

Table 7.3. Average Number of Bits Required to Represent an On-Bit for γ, δ, Golomb and FC for Various op Values for BLISS-1

op	**γ**	**δ**	**Golomb**	**FC (Obs)**	**FC (Best)**	**FC (Worst)**
0.011	8.84	8.34	6.96	8.15	5.85	10.06
0.014	8.42	8.02	6.70	7.79	5.63	9.63
0.028	7.33	7.17	6.00	6.85	4.98	8.47
0.042	6.63	6.62	5.54	6.26	4.55	7.72
0.069	5.76	5.94	4.94	5.51	4.00	6.78

For low *op* values (such as the ones used in Table 7.3), the number of bits required to represent an on-bit in FC is very close to the number of bits in a byte. Using a fixed size codeword that fits a byte provides very efficient processing of compressed bit slices since one byte is used to represent a character and the computers contain operations to manipulate them efficiently. The percent of the gaps that are less than or equal to 255 are given in Table 7.4. Note that these gaps can be represented with only one 8 bit long codeword and the majority of gaps can be represented with one byte.

Table 7.4. Percent of the Gaps that are Less than or Equal to 255

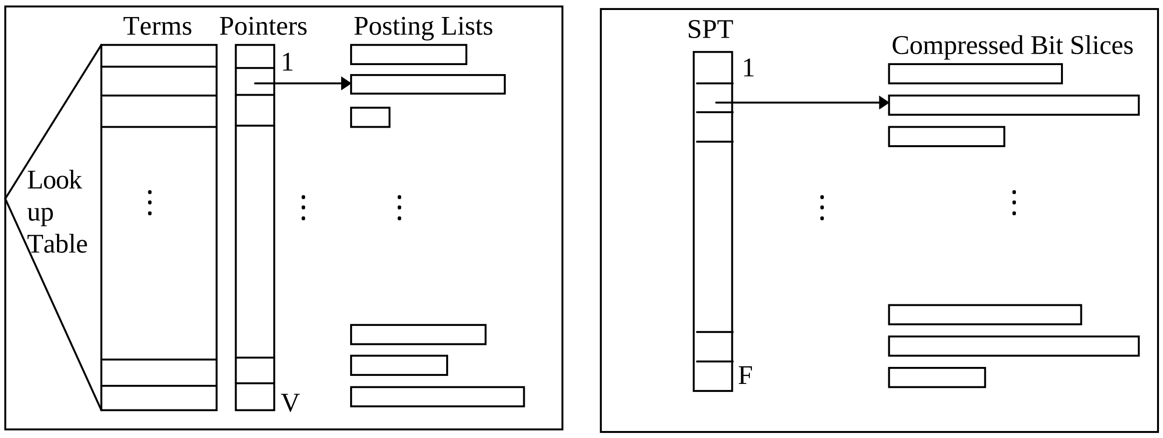
op	**Total Number of Gaps**	**Number of Gaps ≤ 255**	**% of Gaps ≤ 255**
0.011	3,904,897	3,524,697	90.3
0.014	3,893,434	3,633,099	93.3
0.028	7,728,445	7,609,289	98.5
0.042	11,476,059	11,430,734	99.6
0.069	18,866,579	18,861,873	≅100.0

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

7.4 Description and Analysis of C-MFSF for Very Large Signature Sizes

The PFD-MFSF method is inspected for LW, UD, and HW query cases in Section 6.5. The results of the experiments show that the observed false drop values diminish as the signature size increases and they are very close to the expected values. Since the space overhead without compression is relatively high for $F \geq 1800$ (8.75 bytes per distinct term in each record for $F = 1800$), we stop the analysis at $F = 1800$. However, we show that approximately one byte will be sufficient to represent each on-bit if the sparse bit slices are compressed.

Since the compressed bit slices are in varying lengths, a Slice Pointer Table (SPT) with F locations is used. SPT is stored in memory and to read a bit slice first the address of the bit slice is obtained by accessing SPT. To illustrate the difference between C-MFSF and the inverted file method the storage structures of these methods are shown in Figure 7.2. Since the values of F will be about 30,000, the memory required to store SPT will be small. Also, a high percent of this additional memory requirement will be compensated by the decrease in the buffer sizes since the compressed bit slices require less memory.



a. Inverted File method. b. C-MFSF method.
V: Number of unique terms in the vocabulary, F: Number of hashing positions (signature size)
Usually F << V

Figure 7.2. Storage structures of C-MFSF and the inverted file methods.

Equation (3.3) that is used to estimate the time required to process a bit slice of MFSF assumes each bit slice is N bits long. In MFSF each fragment may have a different op value and hence the numbers of on-bits in the bit slices of MFSF may vary. The observed number of bits required to store an on-bit with FC is approximately equal to the average of the best and the worst case (see Table 7.3), i.e., each on-bit is represented with $(k + WC_k) / 2$ bits. Therefore, we estimate the number of disk block accesses to retrieve a bit slice of C-MFSF, sl, as follows.

$$sl_i = \left\lceil \left(N \cdot op_i \cdot \left(\frac{k_i + WC_{k_i}}{2} \right) \right) \Big/ \left(8 \cdot B_{size} \right) \right\rceil \quad for\ 1 \le i \le f \qquad (7.7)$$

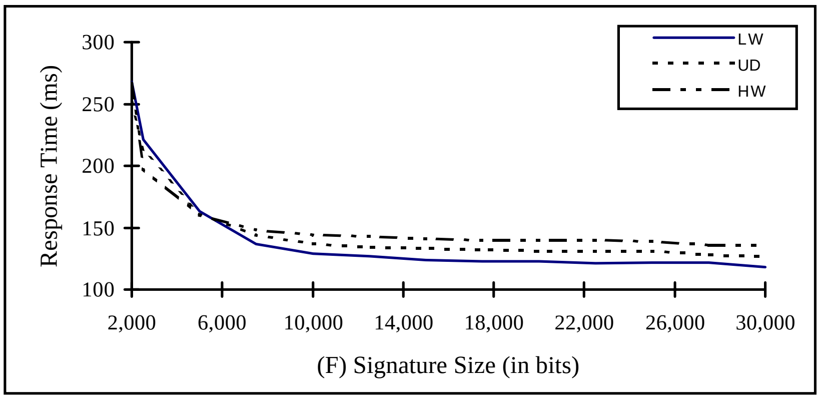
bits where op_i is the expected on bit density in ith fragment, k_i is the codeword length used in this fragment, and WC_{k_i} is the number of bits required to store an on-bit of the ith fragment in the worst case (see Equation 7.6). Since a whole disk block is retrieved to access a part of it, a possible under estimation of the slice lengths can be tolerated. We estimate the time required to process a compressed bit slice of ith partition as follows.

$$T_{slice-i} = Read(sl_i) + T_{byteop} \cdot \left\lceil \left(N \cdot op_i \cdot \frac{k_i + WC_{k_i}}{2} \right) \Big/ 8 \right\rceil \quad (7.7)$$

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

We plot the estimated response time values of C-MFSF for increasing *F* values in Figure 7.3. In this analysis we estimate FD with PFD. (In Chapter 6 we show that PFD estimates FD accurately for MFSF.) The estimated FD values with the expected response time values for LW, UD, and HW query cases are given in Table 7.5. To show the relation between the value of *F* and *S* (total number of bits set by each term in all fragments) we also include *S* values in Table 7.5.

The total number of bits set by each term (*S*) decreases for increasing signature size and becomes equal to two for F > 35,000. At the same time, the number of bit slice evaluations required to reach the stopping condition decreases for increasing *F*. To provide the contribution of each query term to the query evaluation we forced to use at least one on-bit from each term.



(*SP* = 1.0, *N* = 152,850)
Figure 7.3. 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. However, there is a lower bound for the value of *S* that is one. If a sufficiently large *F* value is used, *S* will become equal to one and single term queries can be evaluated with only one seek operation. This idea is inspected by Faloutsos and Chan in [FAL88b]. Since storing SPT will require enormous amount of memory they use smaller *F* values and propose additional data structures to reduce the false drop probability. We included *F* = 100,000 to show that *S* is still equal to two for such a large *F* value.

Table 7.5. Expected FD and TR Values for *N* = 152,850 with Total Number of Bits Set by Each Term (*S*) for LW, UD, and HW Query Cases

	LW			**UD**			**HW**		
F	**S**	**FD**	**TR**	**S**	**FD**	**TR**	**S**	**FD**	**TR**
1,000	6	4.30	795	6	3.69	687	6	2.44	555
2,000	5	0.41	268	5	0.68	252	5	0.46	227
5,000	4	0.04	163	4	0.19	161	4	0.14	160
7,500	3	0.16	137	3	0.12	144	3	0.07	148
30,000	3	0.08	118	3	0.06	127	3	0.07	136
35,000	3	0.19	115	3	0.03	125	2	0.08	135
100,000	2	0.02	102	2	0.02	116	2	0.01	130

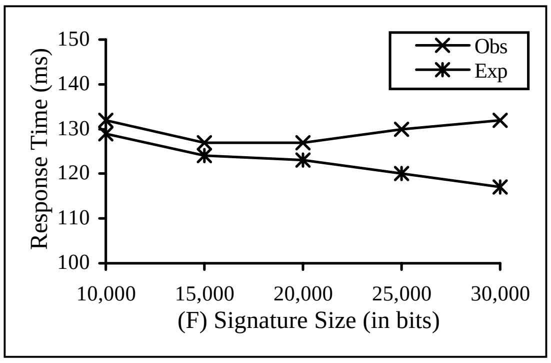
7.5 Experiments with Real Data

The analysis given in the previous section shows that a response time less than 150 milliseconds is possible if large *F* values are used. We tested the optimized C-MFSF configurations with BLISS-1. The expected (denoted by Exp) and the observed (denoted by Obs) response time values are plotted in Figure 7.4 (for easy comparison the observed response time values for LW, UD, and HW repeated in Figure 7.4.d). The expected (denoted by Exp) and the observed (denoted by Obs) average false drop values of these experiments for LW, UD, and HW are given in Table 7.6.

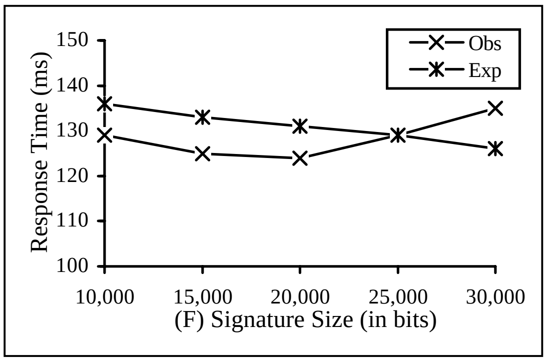
Table 7.6. Expected and Observed Average False Drop Values of C-MFSF for LW, UD, and HW

	LW		**UD**		**HW**	
F	**Exp**	**Obs**	**Exp**	**Obs**	**Exp**	**Obs**
10,000	0.06	0.46	0.05	0.32	0.03	0.14
15,000	0.02	0.60	0.01	0.32	0.01	0.14
20,000	0.01	0.37	0.01	0.28	0.00	0.13
25,000	0.09	0.38	0.07	0.35	0.05	0.14
30,000	0.07	0.40	0.05	0.41	0.03	0.22

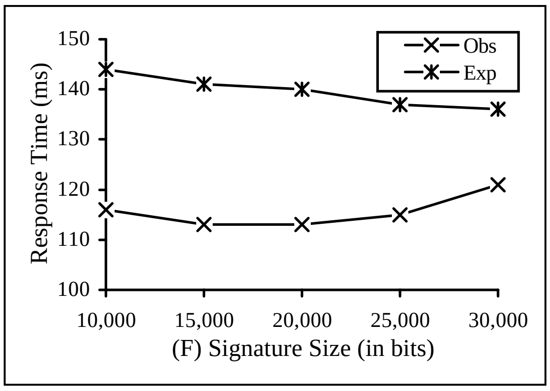
The queries with more than two terms obtain almost no false drops and the query evaluation is completed by accessing only the signature file without any actual record accesses for false drop resolution. Since the compressed signature files are relatively smaller than the uncompressed signature files and the record file, the average seek time for compressed signature files are smaller than the average seek time used in the analysis (30 ms). Therefore, the observed response time for UD and HW query cases are less than the estimated values. The observed response time values for LW query case are greater than the estimated values since LW obtains more false drops than UD and HW.



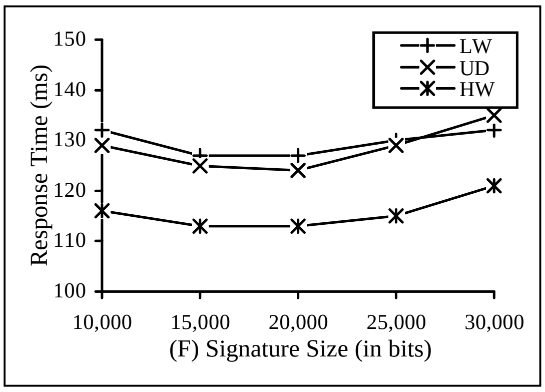
a. LW query case.



b. UD query case.



c. HW query case.



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

Figure 7.4. Expected and observed response time of C-MFSF versus *F* for LW, UD and HW (*SP* = 1).

For all query cases, the observed response time increases for $F > 20{,}000$. Most inspected C-MFSF configurations require setting three bits for each term. Consequently, the number of on-bits in the signature files are approximately the same for all configurations. Therefore, gap sizes, and hence the size of the compressed signature file, increase for increasing signature size. This causes a small increase in the response time (approximately 3 ms per processed bit slice).

The difference between estimated and observed false drops decreases for increasing number of query terms. Most false drops are generated by single term queries. Single term queries have only three (or two) on-bits in their query signature and if one of them shares the same bit slice with a high frequency term, more false drops are produced than the expected number. To obtain better performance with low weight queries with $S = 2$, the frequency of the terms must be considered in the signature file optimization [AKT93b].

7.6 Projection for Large Databases

We performed a series of experiments to test the change in the observed response time for increasing database sizes (N value). The results of the experiments are plotted in Figure 7.5. The test databases for the experiments were obtained by considering only the first N records of the original database. The signature file parameters f, F_r, and S_r ($1 \leq r \leq f$) were optimized for each run by considering the tested N value for $SP = 1$ and $F = 15{,}000$.

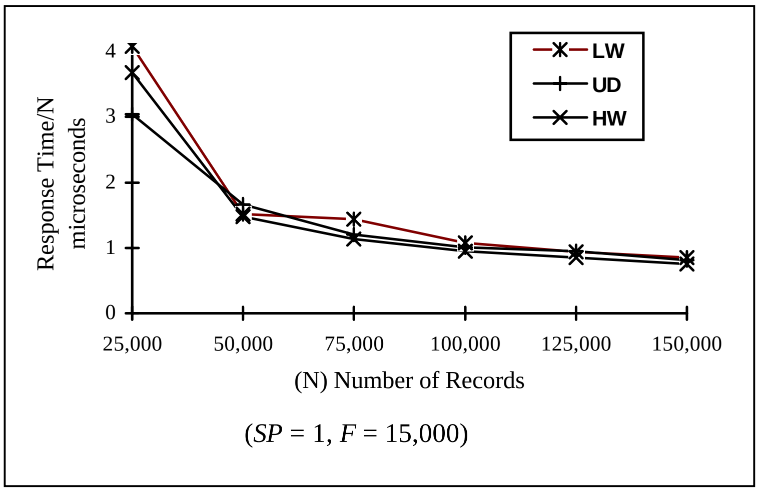


Figure 7.5. Response time per record versus N for LW, UD and HW.

Simulation runs show that approximately the same number of bit slices will be processed for $N = 10^6$ and $N = 150{,}000$. Consequently, the number of seek operations will be the same for increasing N and the number of seek requests per record will decrease for increasing N. Therefore, in the first phase of a query evaluation the time spend for each record of the database decreases for increasing N value.

We can project the result of this experiment to predict the observed response time for larger databases by assuming TR/N ratio will not be greater than 0.85 micro seconds for larger databases. Note that this value is the maximum TR/N figure observed for LW, UD, and HW query cases for $N = 150{,}000$. By assuming TR/N = 0.85 micro seconds, we project the observed response time for $N = 10^6$ as 0.85

seconds. Note that this is a pessimistic assumption since the *TR/N* ratio (response time/record) decreases for increasing *N*.

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

7.7 Theoretical Comparison of C-MFSF and the Inverted Files

Inverted file (IF) methods and signature file methods are efficient search indices. There are theoretical [ZOB92] and experimental comparisons [COU94, ZOB95a] of these methods. However, the performance of IF and signature file methods in terms of efficiency depend on many parameters such as the database instance, the computer used in the experiments, disk space allocation methods, and the amount of available main memory. Due to the absence of well defined fair comparison environments the results of the comparisons become questionable. Another difficulty is that both methods have configuration parameters providing fine tuning of the performance of the methods. Especially signature files have many configuration parameters which provide adaptation of the method to various environments.

In the rest of this section, we provide a brief theoretical comparison of IF and C-MFSF in terms of space overhead and the number of disk accesses required to respond a query. Our aim is to show that C-MFSF opens new promising research directions rather than proving that C-MFSF performs better than IF. In the following discussion, we assume that RPT (record pointer table, see Figure 2.2 and Figure 4.1) is stored in main memory and both methods apply the best compression method for them.

In the IF method at least one disk access is required per query term to read the posting list of the term (we ignore chained long posting lists and compressed bit slices). Also, to obtain the locations of the posting lists, a lookup table must be maintained and it should be searched for query processing. If we assume only one disk access will be required to obtain the location of the posting list of a query term, each query term will require two disk accesses [ZOB95a]. Therefore, in IF, a t term query will require $2 \cdot t$ disk accesses. (Since both methods may trade query evaluation with false drop resolution we ignore this possibility.)

In C-MFSF no lookup table is needed. For F = 30,000, reaching the stopping condition requires processing only three bit slices even for very large databases ($N \geq 10^6$). For the queries containing one or two terms C-MFSF requires three disk accesses plus false drop resolution. Therefore, even without any false drops IF outperforms C-MFSF for single term queries. Both methods will obtain similar results for queries with two terms. IF will require one more disk access but C-MFSF may obtain false drops for t = 2. Therefore, the number of false drop records determines the result of the comparison.

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

Since each term sets more than one bit in C-MFSF, the number of on-bits in a MFSF will be greater than the number of on-bits in the posting lists of an IF constructed for the same database instance. The number of bits required to store each on-bit of a bit string in a compressed form decreases as the number of on-bits in the bit string increases (see Table 7.3). Since, on the average, a posting list of an IF is more sparse than a bit slice of a C-MFSF, an on-bit of C-MFSF requires less space than an on-bit of IF. Additionally, IF requires storing a lookup table containing an entry for each term of the vocabulary. Therefore, the space overhead comparison of C-MFSF and IF depends on the number of terms in the vocabulary. Usually, records contain unique terms such as names, id numbers, or dates. Consequently, the number of terms in the vocabulary will increase as the number of records in the database increases. Note that this will also increase the space overhead of IF.

The performance of IF can be increased if the lookup table can be stored in main memory [ZOB92]. In this case, still one disk access for each query term is required to read the posting list of the query term. If such a large memory will be available, we can store the compressed form of a C-MFSF fragment (or a part of it) in main memory. For example, a fragment of MFSF for BLISS-1 with op = 0.011 (S = 1 and F = 2400) will require 3.31 MBytes ($7.07 \cdot 25.7 \cdot 152850$ $bits$) of memory (see Table 7.3). The value of *op* can be adjusted to fit the fragment to the available memory. Since the bit slices with many on-bits are seldom used in query evaluation, to reduce the memory requirement we can store only short bit slices in memory.

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

8. SUMMARY AND CONTRIBUTIONS OF THE THESIS AND DIRECTIONS FOR FUTURE RESEARCH

8.1 Summary

The thesis, firstly presents basic file structures for information retrieval and summarizes the previous work on inverted files and signature files. We also discuss the distinguishing features of the vertically partitioned signature files and inverted files and clarify the features that can be used for the distinction of these two methods.

To estimate and test the performance of the proposed methods, a simulation and test environment is designed. The experimental environment used in the thesis reflects a real computing environment and uses records of a real application. This provides the validation of our mathematical models with the observed results of the real data experiments and robust projections for very large databases.

The objective of a physical information retrieval method is to provide prompt response to user queries. Therefore, the performance of the inspected signature file methods are measured in terms of response time. To estimate the response time of the inspected signature file methods, the operations involved in query processing with signature files are modeled. Our model is versatile, i.e., it can be used in all operating system environments and is applicable to both dedicated and multi-user IR systems. This is due to the sequentiality probability (SP) concept incorporated into its development.

Generally, search queries of real information retrieval applications contain variable number of terms. Therefore, the access method of such environments should provide acceptable response times for queries ranging from one to several number of terms at the same time. In BSSF the time required to complete the first phase of the query evaluation increases for increasing number of query terms. We propose the Partially evaluated Bit-Sliced Signature File (P-BSSF) method that solves this problem. P-

BSSF employs a stopping condition that tries to complete the first phase of query evaluation without using all on-bits of the query signature, i.e., by partial evaluation. The aim of the stopping condition is to reduce the number of expected false drops to the optimum level that will also provide the lowest response time within the framework of the bit-sliced signature file environment.

The parameters of P-BSSF are optimized in a multi-term query environment by considering the submission probabilities of the queries with different number of terms. Therefore, P-BSSF obtains desirable response times for a wide range of number of query terms.

The response time of P-BSSF decreases for increasing signature size. However, the response time of BSSF first decreases for increasing signature size and then starts to increase. To provide a fair comparison between BSSF and P-BSSF we derive a formula that finds the optimum signature size by minimizing the response time of BSSF. In the comparison of BSSF and P-BSSF, the signature size of BSSF is fixed at the optimum value and the best response time obtained at this optimum signature size is assumed for larger signature sizes. The experiments show that P-BSSF with F = 1200 provides a 85% improvement in response time over BSSF with F = 530 for the UD query case.

Low on-bit density (the probability of a particular bit of a bit slice being on-bit) provides rapid reduction in the expected number of false drops. Thus, the stopping condition defined for P-BSSF is reached by processing fewer number of bit slices with low on-bit density. We propose a new signature generation and query evaluation method, Multi-Fragmented Signature File (MFSF), which improves the performance of P-BSSF without increasing the space overhead (F value). MFSF decreases the response time in multi-term query environments by dividing the signature file into variable sized sub-signature files, fragments. Each fragment is a separate BSSF with its own F and S (the number of bits set to “1” by each term) parameters and the optimality condition is relaxed. Therefore, in MFSF each fragment may have a different on-bit density as opposed to the uniform on-bit densities of the BSSF, B'SSF, GFSSF, and P-BSSF methods.

In MFSF for queries containing more than one term, the bit slices of the fragments with lower on-bit density are used first. The number of bit slices used from the fragments with lower on-bit density increases for increasing number of terms. Therefore, the false drop records are eliminated more rapidly and the performance of MFSF increases for increasing number of query terms. The analysis shows that MFSF obtains up to 17% and 85% performance improvement in response time over P-BSSF and GFSSF, respectively.

We propose a more accurate false drop estimation method, the partitioned false drop estimation method (PFD), for the databases with varying record lengths. In PFD, the records of a database are conceptually divided into disjoint partitions according to the number of distinct terms in the records. Each conceptual partition is considered as a separate signature file and average number of distinct terms in a partition is used to estimate FD in this partition. PFD decreases the differences among the numbers of distinct terms in the records of a partition. Therefore, FD is estimated more accurately.

The sequential, generalized frame-sliced, and multi-fragmented signature file methods are extended to use PFD in FD estimation. The PFD approach provides up to 33%, 25%, and 20% improvements in response time for the sequential, generalized frame-sliced, and MFSF methods, respectively. The experimentally observed FD values and response time values with PFD are very close to the expected values with PFD. Therefore, the signature file methods that use PFD can be compared analytically. Also, the results obtained in the experiments can be safely projected for larger databases.

In MFSF, the results of a single term query can be obtained by processing a few bit slices. However, for very large databases even a bit slice may be too large to obtain a desirable response time. Especially for multi-user environments reading each disk block may require a seek operation that also increases the response time. Fragments of MFSF have varying on-bit densities and the bit slices of a MFSF are sparse. Therefore we propose the Compressed Multi-Fragmented Signature File (C-MFSF) method that extends MFSF by compressing the sparse bit slices of MFSF. Compressing the bit slices of MFSF reduces both the space overhead and the time required to retrieve a bit slice while it requires extra time to decode a compressed bit slice. To reduce the decoding time we propose a simple compression method, Fixed Code (FC), that also provides merging compressed bit slices without decompressing it. Also, in dynamic databases, where record additions are frequent, compressed bit slices can be extended incrementally without completely decompressing them.

Experiments with C-MFSF show that retrieving only two bit slices is sufficient to answer a single term query. For queries containing more than one term at most one bit slice is retrieved for each query term. Inverted file methods require at least two disk accesses (one disk access for searching the term in the lookup table and one disk access for retrieving the posting list of the term) for each query term. Most of the compressed bit slices of a C-MFSF with $N = 10^6$ fit an 8KBytes disk block (the effective block size of disks can be increased using the bucket concept [SALZ88] if required) and usually retrieving a bit slice of C-MFSF from the disk requires only one disk access. Therefore, for single term queries both methods require two disk accesses. For the queries containing more than one term, C-MFSF requires fewer number of disk accesses than the inverted file method.

8.2 Contributions of the Thesis

The major contributions of the thesis can be summarized as follows.

a. The storage structures of bit-sliced signature files and inverted files have some common properties but they are different methods. However, the differences have not been defined clearly. We provide a clarification of this.

b. The response time of BSSF depends on the signature size. Using an improper signature size for BSSF may unnecessarily result in increased response time and higher space overhead. We derive an exact formula that finds the optimum signature size for a given database instance by minimizing the response time.

c. For BSSF, high weight queries unnecessarily require processing many bit slices in multi-term query environments. We define a partial evaluation strategy and derive an exact formula to find the optimum number of bit slice evaluations. The partial evaluation strategy uses a subset of the on-bits of a query signature.

d. We propose a new signature file optimization method, Partially evaluated Bit-Sliced Signature File (P-BSSF). P-BSSF combines optimal selection of *S* value for a given signature size with the partial evaluation strategy in a multi-term query environment. During the selection of the optimal *S* value, we consider the submission probabilities of the queries with various number of terms. Therefore, P-BSSF balances the trade off between fewer slice processing and resolving more false drops; therefore, increases the performance.

e. In experiments with real data we observe bit-slices with too low or too high on-bit densities. Since the partial evaluation approach may use a subset of the on-bits of a query signature, the selection method of the on-bits used in the query evaluation affects the results of the experiments. Therefore, we tested three different query on-bit selection methods (SS, MF, and RR) in experiments with real data and we show that the RR method obtains similar results with the MF method. Since RR maximizes equal contribution of each query term to the query evaluation we prefer to use it.

f. The stopping condition defined for P-BSSF improves the system performance by processing a limited number of bit slices. To further improve the performance of P-BSSF we propose a new signature file organization and query evaluation method, Multi-Fragmented Signature File (MFSF). A MFSF contains vertical fragments (sub bit-sliced signature files) with variable on-bit densities that provides better optimization of multi-term queries. MFSF provides decreasing response time for increasing number of query terms.

g. Usually, records of unformatted databases contain varying number of terms. False drop estimation formulas used so far assume the existence of the same (average, D_{avg}) number of terms per record and this causes some error in the estimation of the number of false drop records (FD). We propose a more accurate false drop estimation method, the Partitioned False Drop estimation method (PFD), for the databases with varying number of distinct terms in the records. In PFD, the records of a database are conceptually divided into disjoint partitions according to the number of distinct terms in the records. The conceptual partitioning of records decreases the differences among the numbers of distinct terms in the records of a partition and FD is estimated more accurately.

h. For the databases with varying number of terms adjusting the value of *S* according to the optimality condition may produce poor performance for the sequential signature file method. We list other optimization methods for the sequential signature file method and compared them.

i. PFD is applied to GFSSF, P-BSSF, and MFSF methods. The performance increase obtained by estimating FD with PFD is measured in experiments with real data.

j. Most of the bit slices of a MFSF are sparse. We propose the Compressed Multi-Fragmented Signature File (C-MFSF) method that extends MFSF by compressing the sparse bit slices of MFSF. The C-MFSF approach increases the performance of MFSF in terms of response time and space overhead. We also propose a simple compression method that provides efficient merging of compressed bit slices.

8.3 Directions for Future Research

C-MFSF provides efficient processing of conjunctive queries with many terms while it also provides desirable response times for the queries containing a few terms. The performance of MFSF increases for increasing number of query terms. Therefore, C-MFSF opens new research directions. We list some of them below.

a. In the vertical partitioning methods the same signature size (fixed *F*) is used for all records irrespective of the number of terms in the records. The signature file optimization algorithms that use the partitioned false drop estimation method (PFD) adjust the value of *S* such that minimum response time is obtained with the fixed *F* constraint. However, In Section 6.3 we show that adjusting the value of *F* and *S* with respect to the distinct number of terms of records obtains up to 35% performance improvement in terms of response time for BLISS-1. Similar performance improvements can be obtained with alternative file organization methods for vertical partitioned signature files. One such method for *SP* = 0, is dividing the record file and the signature file physically with respect to the record lengths. This can be considered as the physical counterpart of the conceptual partitioning of records proposed by the PFD approach.

b. Searching with partially specified query terms, (i.e., using a wildcard character, *, that matches any sequence of letters) may require accessing many vocabulary entries in the IF method. To provide searching with wildcard character each term may be decomposed into *n-grams* and each n-gram is indexed [WIT94]. Faloutsos proposes to divide a term into successive overlapping triplets. A term signature is obtained by superimposing the signatures of the triplets extracted from that term [FAL85b]. Since the partial query evaluation is an inexact match method, C-MFSF can be used to evaluate queries containing wildcard characters efficiently.

c. Ishikawa et al. use BSSF as set access facilities in object oriented database systems [ISH93]. An instance of a multi-valued (or set valued) attribute of relational database systems can be considered as an unformatted record. Since C-MFSF provides desirable response times for the queries with many terms, it can be used to search a multi-valued attribute efficiently.

d. Usually, IR systems are accessed simultaneously by many users [COU94]. Therefore, an IR system serves many users in a time sharing approach and evaluates hundreds of queries at the same time. Since the partial evaluation approach of C-MFSF uses a subset of the query signature on-bits, selecting a bit slice for query evaluation that can also be used for other queries decreases the number of disk accesses and improves the performance of the IR system. This needs further investigation for C-MFSF.

e. The superimposed signature file approach represents each record with a fixed size bit string which facilitates parallel processing of search requests [COU94, GRA92, POG87]. Parallel processing of vertically partitioned signature files is also studied in the literature [GRA92, PAN94]. C-MFSF can be adapted for parallel processing environments.

f. In C-MFSF, most false drops are generated by single term queries. Single term queries have only *S* 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 effect of high frequency terms increases for decreasing *S* values. To obtain better performance with low weight queries, the frequency of the terms must be considered in the signature file optimization [AKT93b].

REFERENCES

[AKT93a] Aktug, D., Can, F. 1993. Signature files: an integrated access method for formatted and unformatted databases. Submitted to *ACM Comp. Surveys* (under revision).

[AKT93b] Aktug, D., Can, F. 1993. Analysis of multiterm queries in a dynamic signature file organization. In *Proceedings of the 16th International ACM-SIGIR Conference on Research and Development in Information Retrieval* (Pitsburgh, Penn., June). ACM, N.Y. 96-105.

[BEL90] Bell, T. C, Cleary, J. G., and Witten, I. H. 1990. *Text Compression.* Prentice Hall, Englewood Cliffs, N.J.

[BEL93] Bell, T. C., Moffat, A., and Zobel, J. 1993. Data compression in full-text retrieval systems. *Journal of the American Society for Information Science.* 41, 9. 508-531.

[BLA90] Blair, D. C. 1990. *Language and Representation in Information Retrieval.* Elsevier, Netherlands.

[CAN85] Can, F. 1985. A new clustering scheme and its use in an information retrieval system incorporating the support of a database machine. Ph.D. dissertation, Dept. of Computer Engineering. Middle Each Technical Univ., Ankara, Turkey.

[CAN90] 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.

[CAN93] Can, F. 1993. Incremental clustering for dynamic information processing. *ACM Transactions on Information Systems.* 11, 2 (Apr.). 143-164.

[CHA93] 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.). 138-144.

[CHR84a] 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.

[CHR84b] Christodoulakis, S., Faloutsos, C. 1984. Design considerations for a message file server. *IEEE Trans. on Software Engineering.* 10, 2 (March). 201-210.

[COU94] Couvreur, T. R., Benzel, R. N., Miller, S. F., Zeitler, D. N., Lee, D. L., Singhal, M., Shivaratri, N., and Wong, W. Y. P. 1994. An analysis of performance and cost factors in searching large text databases using parallel search systems. *Journal of American Society for Information Science.* 45, 7. 443-464.

[DAT90] Date, C. J. 1990. *An Introduction to Database Systems, 5th edition.* Addison Wesley, Reading, MA.

[DEP86] 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, Sep.). ACM, N.Y. 77-87.

[DOU89] Douglas, P. M., Stephanie, W. H. 1989. The constituent object parser: syntactic structure matching for information retrieval. *ACM Transactions on Information Systems.* 7, 3 (July). 292-316.

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

[FAL85a] Faloutsos, C. 1985. Access methods for text. *Computing Surveys.* 17, 1 (March). 49-74.

[FAL85b] 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.

[FAL85c] Faloutsos, C., 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* (Endowment, Aug.). 165-170.

[FAL87a] Faloutsos, C., Christodoulakis, S. 1987. Description and performance analysis of signature file methods for office filing. *ACM Transactions on Information Systems.* 5, 3 (July). 237-257.

[FAL87b] Faloutsos, C., Christodoulakis, S. 1987. Optimal signature extraction and information loss. *ACM Transactions on Database Systems.* 12, 3 (Sep.). 395-428.

[FAL88a] Faloutsos, C. 1988. Signature files: an integrated access method for text and attributes, suitable for optical disk storage. *BIT.* 28, 4. 736-754.

[FAL88b] 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.

[FAL91] Faloutsos, C., Jagadish, H. V. 1991. Hybrid index organizations for text databases. *CS-TR-2621, Computer Science, Univ. of Maryland.*

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

[FOX91] Fox, E. A., Chen, Q. F., Daoud, A. M., and Heath, L. S. 1991. Order-preserving minimal perfect hash functions and information retrieval. *ACM Transactions on Information Systems.* 9, 3 (July). 281-308.

[FOX93] Fox, E. A., Robert, K., Eskinder, S., Daoud, A. M., and Cline, B. E. 1993. Development of a modern OPAC: From REVTOLC to MARIAN. In *Proceedings of the 16th International ACM-SIGIR Conference on Research and Development in Information Retrieval* (Pitsburgh, Penn., June). ACM, N.Y. 248-259.

[GRA92] Grandi, F., Tiberio, P., and Zezula, P. 1992. Frame-sliced partitioned parallel signature files. In *Proceedings of 15th International ACM-SIGIR Conference* (Copenhagen, June). 286-297.

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

[HAR92] Harman, D., Fox, E., Baeza-Yates, R., and Lee, W. 1992. Inverted files. In *Information Retrieval Data Structures and Algorithms.* Edited by W. B. Frakes and R. Baeza-Yates. Prentice Hall, Englewood Cliffs, N.J. 28-43.

[HAS81] Haskin, R. L. 1981. Special-purpose processors for text retrieval. *Database Engineering.* 4, 1 (Sep.). 16-29.

[ISH93] Ishikawa, Y., Kitagawa, H., and Ohbo, N. 1993. Evaluation of signature files as set access facilities in OODBs. In *Proceedings of the ACM SIGMOD'93 Conference* (Washington, D.C., May). 247-256.

[KEN90] Kent, A., Sacks-Davis, R., and Ramamohanarao, K. 1990. A signature file scheme based on multiple organizations for indexing very large text databases. *Journal of the American Society for Information Science.* 41 (7). 508-534.

[KLE89] Klein, S. T., Bookstein, A., and Deerwester, S. 1989. Storing text retrieval systems on CD-ROM: comparison and encryption considerations. *ACM Transactions on Information Systems.* 7, 3 (July). 230-245.

[KNU75] Knuth, D. E. 1975. *The Art of Computer Programming. vol. 3: Sorting and Searching.* Addison-Wesley, Reading, MA.

[KÖK79] Köksal, A. 1979. Bilgi erişim sorunu ve bir belge dizinleme ve erişim dizgesi tasarımı ve gerçekleştirimi. Doçentlik tezi. Hacettepe Üniv., Ankara, Turkey.

[KOÇ95a] Koçberber, S., Can, F. 1995. Generalized vertical partitioning of signature files. Tech. Rept. BU-CEIS-9513, Dept. of Computer Eng. and Information Science, Bilkent University (ftp://ftp.cs.bilkent.edu.tr/ pub/tech-reports/1995/BU-CEIS-9513.ps.z).

[KOÇ95b] Koçberber, S., Can, F. 1995. Optimization of bit-sliced signature files in multi-term query environments. In the *Proceedings of 10th International Symposium on Computer and Information Sciences.* Ephesus, Turkey. 161-168.

[KOÇ95c] Koçberber, S., Can, F. 1995. Partial evaluation of queries for bit-sliced signature files. *Information Processing Letters* (to appear).

[KOÇ95d] Koçberber, S., Can, F. 1995. Vertical fragmentation of superimposed signature files using partial evaluation of queries. Submitted to *Information Processing and Management.*

[LEE89] Lee, D. L., Leng, C. W. 1989. Partitioned signature files: design issues and performance evaluation. *ACM Transactions on Information Systems.* 7, 2 (Apr.). 158-180.

[LEE90] Lee, D. L., Leng, C. W. 1989. A partitioned signature file structure for multi-attribute and text retrieval. In *Proceedings of the 6th International Conference on Data Engineering,* (Los Angeles, Calif.). 389-397.

[LEE95] Lee, D. L., Kim, Y. M., and Patel, G. 1995. Efficient signature file methods for text retrieval. *IEEE Transactions on Knowledge and Data Engineering.* 7, (3). 423-435.

[LEL87] Lelewer, D. A. and Hirschberg, D. S. 1987. Data compression. *ACM Computing Surveys*. 19, 3 (Sep.). 261-296.

[LEN92] Leng, C. W. R., Lee, D. L. 1992. Optimal weight assignment for signature generation. *ACM Transactions on Database Systems*. 17, 2 (June). 346-373.

[LIN88] Lin, Z., Faloutsos, C. 1988. Frame-sliced signature files. Tech. Rept. CS2146 and UMIACS-TR-88-88, Comput. Sci. Dept. University of Maryland.

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

[MOF92] Moffat, A., Zobel, J. 1992. Parameterised compression for sparse bitmaps. In *Proceedings of the 15th International ACM-SIGIR Conference on Research and Development in Information Retrieval* (Copenhagen, June). ACM, N.Y. 274-285.

[MOF95a] Moffat, A., Zobel, J. 1995. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems* (to appear).

[MOF95b] Moffat, A., Zobel, J., and Sharman, N. 1995. Text compression for dynamic document databases. *IEEE Transactions on Knowledge and Data Engineering* (to appear).

[OMI90] Omiecinski, E., Scheuermann, P. 1990. A parallel algorithm for record clustering. *ACM Transactions on Database Systems*. 15, 4 (Dec.). 599-624.

[PAN94] Panagopoulos, G., Faloutsos, C. 1994. Bit-sliced signature files for very large text databases on a parallel machine architecture. In *Proceedings of the EDBT'94 Conference*, (Cambridge, Mass., March). 379-392.

[PFA80] Pfaltz, J. L., Berman, W. J., and Cagley, E. M. 1980. Partial-match retrieval using indexed descriptor files. *Communications of the ACM*. 23, 9 (Sep.). 522-528.

[POG87] Pogue, C.A., Willett, P. 1987. Use of text signatures for document retrieval in a highly parallel environment, *Parallel Computing*. 10. 259-268.

[RAB91] Rabbiti, F., Savino, P. 1991. Image query processing on multilevel signatures. In *Proceedings of the 14th Annual International ACM-SIGIR Conference* (Chicago, Illinois, Sep.), ACM, N.Y. 305-314.

[RAM83] Ramamohanarao, K., Lloyd, J. W., and James, A. T. 1983. Partial-match retrieval using hashing and descriptors. *ACM Transactions on Database Systems*. 8, 4 (Dec.). 552-576.

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

[SAC83] Sacks-Davis, and Ramamohanarao, K. 1983. A two level superimposed coding scheme for partial match retrieval. *Information Systems*. 8 (4). 273-280.

[SAC85] Sacks-Davis, R. 1985. Performance of multikey access method based on descriptors superimposed coding techniques. *Information Systems*. 10 (4). 391-403.

[SAC87] Sacks-Davis, R., Kent, A., and Ramamohanarao, K. 1987. Multikey access method based on descriptors superimposed coding techniques. *ACM Transactions on Database Systems*. 12, 4 (Dec.). 655-696.

[SAL75] Salton, G. 1975. A Theory of Indexing. *Regional Conference Series in Applied Mathematics*. Philadelphia, PA.

[SAL83a] Salton, G., Fox, E. A., and Wu, H. 1983. Extended Boolean information retrieval. *Communications of the ACM*. 26, 11 (Nov.). 1022-1036

[SAL83b] Salton, G., McGill, M. J. 1983. *Introduction to Modern Information Retrieval*. McGraw-Hill.

[SAL88] Salton, G., Buckley 1988. Term-weight approaches in automatic text retrieval. *Information Processing and Management*. 24, 5 (May). 513-523.

[SAL89] Salton, G. 1989. *Automatic Text Processing: The Transformation Analysis, and Retrieval of Information by Computer*. Addison Wesley, Reading, MA.

[SALZ88] Salzberg, B. J. 1988. *File Structures: An Analytical Approach*. Prentice Hall, Englewood Cliffs, N.J.

[THA88] Tharp, A. L. 1988. *File Organization and Processing*. John Wiley and Sons, N.Y.

[TSI83] Tsichritzis, D., Christodoulakis, S. 1983. Message files. *ACM Trans. on Office Information Systems*. 1, 1 (Jan.). 88-98.

[ULL88] Ullman, J. D. 1988. *Principles of Database and Knowledge-Base Systems.* Computer Science Press, Rockville, Maryland.

[VAN79] van Rijsbergen, C. J. 1979. *Information Retrieval, 2nd edition.* Butterwortths, London.

[WIL88] Willett, P. 1988. Recent trends in hierarchic document clustering: a critical review. *Information Processing and Management.* 24, 5. 577-597.

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

[ZEZ91] Zezula, P., Rabitti, F., and Tiberio, P. 1991. Dynamic partitioning of signature files. *ACM Transaction on Information Systems.* 9, 4 (Oct.). 336-367.

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

[ZOB95a] Zobel, J., Moffat, A., and Ramamohanarao, K. 1995. Inverted files versus signature files for text indexing. Tech. Rept. CITRI/TR-95-5, Dept. of Computer Science, The University of Melbourne.

[ZOB95b] Zobel, J., Moffat, A. 1995. Adding compression to a full-text retrieval system. *Software-Practice and Experience.* 25, 8 (Aug.). 891-903.

APPENDICES

A. Definition of More Frequently Used Acronyms

Acronym	**Full Name**	**Defined in Section**
AFD-GFSSF	Average False Drop estimated Generalized Frame Sliced Signature File	6.4
AFD-MFSF	Average False Drop estimated Multi Fragmented Signature File	6.5
AFD-SSF	Average False Drop estimated Sequential Signature File	6.3
BC	bit Block Compression	2.2.2
BRTM	Binary Record Term Matrix	2.1.2
BSSF	Bit-Sliced Signature File	1.2
B’SSF	extended Bit-Sliced Signature File	4.3
CBS	Compressed Bit Slices	2.3
C-MFSF	Compressed Multi-Fragmented Signature File	1.3
DCBS	Doubly Compressed Bit Slices	2.4
FC	Fixed Code	7.2
FD	number of False Drop records	1.3
FFFS	Fixed F Fixed S	6.3
FFFS	Fixed F Varying S	6.3
FSB	Fixed Size Block	2.2.1.3
FSSF	Frame-Sliced Signature File	2.2.3.1.2
FWB	Fixed Weight Block	2.2.1.3
GFSSF	Generalized Frame-Sliced Signature File	1.2

HW	High Weight (query case)	3.4
IF	Inverted File	2.
IFD	Individual False Drop estimation method	6.2
IR	Information Retrieval	1.
IP	Improvement Percentage	3.1
LHSS	Linear Hashing with Superimposed Signatures	2.2.3.2.1
LW	Low Weight (query case)	3.4
MARC	MAchine Readable Cataloging	3.2
MF	Minimum *op* First query signature on-bit Selection	4.7
MFSF	Multi-Fragmented Signature File	1.3
NFD	No False Drop	2.4
P-BSSF	Partially evaluated Bit-Sliced Signature File	1.3
PFD	Partitioned False Drop estimation method	1.3
PFD-GFSSF	Partitioned False Drop estimated Generalized Frame Sliced Signature File	6.4
PFD-MFSF	Partitioned False Drop estimated Multi Fragmented Signature File	6.5
PFD-SSF	Partitioned False Drop estimated Sequential Signature File	6.3
RL	Run Length encoding	2.2.2
RPT	Record Pointer Table	2.1
RR	Round Robin query signature on-bit Selection	4.7
SC	Superimposed Coding	2.2.1.2
SF	Signature File	2.
SS	Sequential query signature on-bit Selection	4.7
SPT	Slice Pointer Table	7.4
SSF	Sequential Signature File	1.2

STD	Standard Deviation of D	3.2
TFD	Total number of False Drops	6.5
UD	Uniform Distribution (query case)	3.4
VBC	Variable bit-Block Compression	2.2.2
VFVS	Varying F Varying S	6.3
WS	Word Signature	2.2.1.1

B. Definition of More Frequently Used Symbols (used in calculations)

Symbol	**Definition**	**Defined in Section**
b_s	on-bit density of sth bit slice used in query evaluation	5.2
f	number of fragments	5.1
fd	false drop probability	2.2
fd_i	false drop probability after processing *i* bit slices	4.4
$fd_{W(Q)_t}$	false drop probability for a *t* term query	4.1
k	number of frames in a GFSSF	4.3.2
m	number of bits to be set by each term in a frame	4.3.2
n	number of frames selected to set bits	4.3.2
op	average on-bit density	4.4
op_r	average on-bit density in rth fragment	5.1
s	size of a frame	4.3.2
t	number of query terms	1.3
t_{max}	maximum number of terms in a query	3.4
w_t	total number of on-bits in all fragments of a *t* term query signature	5.1
AD_i	average number of distinct terms in ith partition	6.2
B_{size}	size of a disk block (bytes)	3.5
D	number of distinct terms in a block	2.2.1.1
D_{avg}	average number of distinct terms in a record	1.3
D_{max}	maximum number of distinct terms in a record	3.2

DOM	domain of D values	6.2
DOM_i	sub domain	6.2
F	size of a signature (bits)	1.2
F_r	size of rth fragment of F (bits)	5.1
FD	expected number of false drops	6.1
FD_i	expected number of false drops after processing *i* bit slices	5.9
$FD_{w(Q)_t}$	expected number of false drops after processing all on-bits of a *t* term query	4.1
IP	improvement percentage	3.1
L_i	lower bound of domain i	6.2
N	number of records in database	1.2
N_i	number of records in ith partition	6.2
P_i	partition i	6.2
P_t	probability of submission of a t term query	3.4
P_{size}	size of a record pointer (bytes)	3.5
PB	number of record pointers in record pointer buffer	3.5
RB	average number of disk block accesses required to retrieve a record	3.2
RFD_{i+1}	number of reduced false drops	5.3
RT(i)	response time as a function of number of bit slices used in the query evaluation	4.4
RT(F)	response time as a function of *F*	4.2
RT(S,t)	response time as a function of *S* and *t*	4.5
RT_t	response time for a t term query	5.4
S	total umber of bits set by each term	1.2

S_r	number of bits set by each term in rth fragment	5.1
S_{max}	maximum number of distinct term signatures	2.2.1.1
SP	sequentiality probability of logically consecutive disk blocks	3.5
T_{byteop}	time required to perform bit operations between two bytes	3.3
T_{read}	time required to read a disk block	3.5
$T_{resolve}$	false drop resolution time for one record	3.5
T_{scan}	time required to scan a record to test it with query	3.5
T_{seek}	time required to position read head of disk	3.5
T_{slice}	time required to process a bit slice	3.5
$T_{slice-i}$	time required to process a compressed bit slice of ith partition	7.3
T_{wordop}	time required to perform a bitwise AND operation between two memory words and store the result in one of the words	3.5
TR	expected response time	4.5
U_i	upper bound of domain i	6.2
V	number of distinct terms in the database	2.1.2
V(t)	variance of t	
$W(Q)_t$	query weight for a *t* term query	4.1
W(Q)(r,t)	number of on-bits in rth fragment for a *t* term query	
W_{size}	size of a memory word (in bytes)	3.5

C. Hashing Function and On-Bit Position Generator

```
/* Basic functions for signature generation.
   FindRandSeed : Obtains random number seeds for a given term
   FillRandom   : Obtains bit positions to be set to “1” using the
   random number seed generated by FindRandSeed

   Compatibility: Borland C & gcc


*/

#include <string.h>
#include <stdlib.h>
#include <values.h>

/* Size of a local array. The array can be dynamic. However,
   frequent calls to obtain on-bit positions may fragment available
   memory.
*/
#define MAX_RAND              500

/* FindRandSeed

   Borland C uses unsigned int seeds which are too small. Therefore,
   two different seeds are generated for each term.

   Input Parameters:
      kw    : pointer to the term
      kw_len: number of characters in the term

Output Parameters:
      seed1 : first random number seed
      seed2 : second random number seed
*/
void FindRandSeed(char *kw, int kw_len, unsigned *seed1, unsigned
*seed2)
{ unsigned rand_seed, temp;
  int i;
  char *ptr;

  rand_seed = 0U;
  for ( i = kw_len, ptr = kw; i >= sizeof(unsigned);
             i -= sizeof(unsigned), ptr += sizeof(unsigned))
      { memcpy(&temp, ptr, sizeof(unsigned));
        rand_seed += temp;
        rand_seed >>= 1;
      }
  for ( ; i; --i, ++ptr)
    rand_seed += (((unsigned) *ptr) << i );

  *seed1 = rand_seed;

  rand_seed = (unsigned) *kw;
  for ( i = kw_len - 1, ptr = kw + 1; i >= sizeof(unsigned);
             i -= sizeof(unsigned), ptr += sizeof(unsigned))
      { memcpy(&temp, ptr, sizeof(unsigned));
        rand_seed += temp;
        rand_seed >>= 1;
      }
  for ( ; i; --i, ++ptr)
    rand_seed += (unsigned) *ptr;

/* If the seeds are equal, make them different */
  *seed2 = rand_seed;
  if ( *seed1 == *seed2 )
     *seed2 += (int) *kw;
  return;

} /* end FindRandSeed */


/* FillRandom

   Borland C uses unsigned int seeds which are too small. Therefore,
   two different seeds are used for each term. A random number The
random numbers obtained

   Input Parameters:
      rand_seed1 : the first random number seed
      rand_seed2 : the second random number seed
      max_val    : random number(s) are generated between 0 and
                   max_val - 1
      set_num    : number of bit positions to be generated
      distinct   : TRUE  --> generate set_num distinct bit positions
                   FALSE --> generate not necesarily distinct bit
                             positions (number of generated bit
                             positions may be less than set_num)

Output Parameters:
      set_pos    : pointer to an integer array. At least set_num
                   locations should be allocated before calling
                   FillRandom.
Returns: number of bit positions generated or error
*/

int FillRandom(unsigned *rand_seed1, unsigned *rand_seed2,
               int max_val, int set_num, int *set_pos, int distinct)
{ int try_cnt, pos, i, k, set_cnt, rand_pos;
  unsigned rand1[MAX_RAND], rand2[MAX_RAND];
  unsigned long randval;

if ( distinct && set_num >= max_val + 1)
      return -1; /* Error: not enough bit positions */

  try_cnt = 0;
  set_cnt = 0;
  rand_pos = set_num + 1;
  rand1[set_num] = *rand_seed1;
  rand2[set_num] = *rand_seed2;

  for ( try_cnt = 0; try_cnt < set_num; ++rand_pos )
    { /* If all random numbers are used produce new numbers */
      if ( rand_pos > set_num )
       { rand_pos = 0;
       #ifdef __TURBOC__
         srand(rand1[set_num]);
       #else
```

```
         srandom(rand1[set_num]);
       #endif
         for (i = 0; i <= set_num; ++i)
         #ifdef __TURBOC__
             rand1[i] = random(MAXINT);
         #else
             rand1[i] = random();
         #endif

       #ifdef __TURBOC__
         srand(rand2[set_num]);
       #else
         srandom(rand2[set_num]);
       #endif
         for (i = 0; i <= set_num; ++i)
         #ifdef __TURBOC__
            rand2[i] = random(MAXINT);
         #else
           rand2[i] = random();
         #endif

       }
      randval = (unsigned long) rand1[rand_pos] + (unsigned long)
                                                    rand2[rand_pos];
      pos = randval % max_val;

for ( k = 0; k < set_cnt; ++k)
        if ( *(set_pos +k) == pos )
           break;
      /* if produced position is already in the list */
      if ( k < set_cnt )
       { if ( ! distinct )
            ++try_cnt;
         continue;
       }
      *(set_pos + set_cnt) = pos;
      ++set_cnt;
      ++try_cnt;
    }
  *rand_seed1 = rand1[set_num];
  *rand_seed2 = rand2[set_num];

  return set_cnt;
} /* FillRandom */
```

D. List of Stop Words

The terms listed in decreasing word length are articles of different languages. In MARC records the Library of Congress subject headings are used. Therefore, most of them do not contain noisy words. Title and other fields are indexed without any stemming.

HENAS HENOS HINAR HINIR

EENE EINE EYNE FROM HEIS HENA KATA SINA UPON VEYA

AND ANJ BIR BUT DAS DEI DEN DER DET DIE EEN EGY EIN EIT ELS ETT EYN FOR GLI

HAI HEN HET HIN HOI ILE ISA LAS LES LOS MIA NJE NJI NOT THE UNA UNE UNO UNS

AL AM AN AS AT AZ BY DE DI EI EL EN ET GL HA HE HI HO IN KA LA LE HI IS LO

LU NA NY OF ON OR OS SI TA TO UM UN US VE YE YN YR

All single letters (A-Z)