Multikey Access Methods Based on Term Discrimination and Signature Clustering

Jae W. Chang, Joon H. Lee and Yoon J. Lee

Department of Computer Science, KAIST
P.O. Box 150, Chongryang, Seoul, Korea 131
jwjang%csd.kaist.ac.kr@relay.cs.net

Abstract

In order to improve the two-level signature file method designed by Sacks-Davis et al. [20], we propose new multikey access methods based on term discrimination and signature clustering. By term discrimination, we create separate, efficient access methods for the terms frequently used in user queries. We in addition cluster similar signatures by means of these terms so that we may achieve good performance on retrieval. Meanwhile we provide the space-time analysis of the proposed methods and compare them with the two-level signature file method. We show that the proposed methods achieve 15-30% savings in retrieval time and require 3-9 % more storage overhead.

1. Introduction

Traditional database management systems are well suited for a variety of applications in the commercial world such as air line reservations, banking, etc. These applications typically use formatted data. Recently, there have been many attempts to extend conventional systems so that they can handle documents (or records) containing both formatted fields and free texts [4,5,12,15]. Among such applications, there are library systems, medical records systems, and office information systems [13].

Because it is essential to efficiently store and retrieve both formatted data and free texts, efficient access methods are required in such systems. An approach widely advocated for both data bases uses the signature file method [14]. Signature file methods, in fact, have been proposed for various applications, such as multikey retrieval [17,18,19], text retrieval [6,11], office systems [4,10], and prolog systems [1,3,16].

A signature (descriptor) in the signature file method is associated with each record, the signature being an encording of the index terms used to retrieve the record. When a query is processed, the file of signatures, rather than the data file, is examined for potential matches. This method provides good retrieval performance and efficient storage usage [6]. As well as forming record signatures for individual records, it is possible to maintain block signatures for blocks of records, thus forming the two-level signature file method [18,20]. In this method, the file of block signatures is viewed as a bit slice representation whereas the file of record signatures is built using a bit string one.

In this paper, we propose new multikey access methods which improve the above two-level method using term discrimination and record signature clustering. Faloutsos & Christodoulakis [9] considered it unrealistic to assume both uniform query and uniform occurrence frequency. They proposed an encording scheme to allow special treatment to terms with high discriminatory power, so as to improve the performance of the signature file method. The terms with high discriminatory power are the ones which are frequently used in user queries but are infrequently occurring in data file. In the proposed methods, we also distinguish terms with high discriminatory power from ones with low discriminatory power and specially treat the formers to acquire good retrieval performance. However, instead of using a special encording scheme we construct additional, efficient access

methods, e.g. the inverted file, for the terms with high discriminatory power.

Meanwhile we cluster similar record signatures so that we can achieve better performance on retrieval to records (documents). To make the clustering easy and effective, it is possible to combine term discrimination and record signature clustering. That is, we can form clusters by means of the similarity between terms with high discriminatory power, rather than all the terms. This still keeps clustering benefits to a high degree.

The remainder of the paper is organized as follows. First we present a brief overview of the signature file method. In section 3 we propose new, efficient multikey access methods based on term discrimination and record signature clustering. In section 4 we provide the space-time analysis of the proposed methods. In section 5 we present the performance results of the proposed methods and compare them with that of the two-level signature file method. The conclusions and further works are presented in section 6.

2. Signature Files

In a recent review of text retrieval methods, Faloutsos [6] has presented the signature file method as an access method applicable to both formatted and unformatted data. In fact, the signature file is an abstraction which contains the signatures of the records (documents) in data file. A signature is a bit string formed from the terms that are used to index a record. Indexing using signature files assigns a signature to every record in the data file. To answer a query, the signature file, rather than the data file, is examined first for immediately discarding many non-qualifying records.

Signature files typically use superimposed coding [14] to create the signature of a record. When a record consists of n terms, each term is converted into a bit string (term descriptor) using a hash function. A record descriptor is formed by superimposing (inclusive ORing) the n term descriptors. For example, we assume that a record consists of 2 terms, say Database and Date. The record signature of this record using superimposed coding is given below.

Database	0010 0100
Date	1000 1000
Record Signature	1010 1100

This method naturally supports variable numbers of terms per record since the number of terms does not affect the signature length. Thus records with multi-valued fields (fields in a record which can contain more than one value) and free text are handled easily because there is no distinction made between terms (values) belonging to the same field.

To check if a signature matches a query, the query terms are also hashed to form a query descriptor using exactly the same way used to generate the record signatures. If every bit set in the query descriptor is also set in the record signature, then the record signature is a potential match. Superimposed coding can result in a case where the query descriptor qualifies the record signature, but the corresponding record does not actually contain the query terms. This is called a false match. The probability of a false match occurred is a function of the number of bits in the query descriptor and the record signature. False matches place a practical restriction on the number of terms that a record may contain for a given signature size. Detailed formulas for calculating false match probability and values for k and b are presented in [4], where b is a signature size and k is the number of bits set to 1.

Meanwhile many works have focused on minimizing the search time for signature files. To efficiently access signature files, four strategies have been adopted as follows [13]:

(i) bit slice representations
(ii) multilevel indexes.
(iii) special encoding schemes based on term discrimination
(iv) compression techniques.

Roberts [17] proposed a bit slice representation to improve a bit string one. A bit slice approach reduces the amount of signature file that must be retrieved on query. Instead of viewing the signature file as consisting of N strings of b bits in length (where N is the number of records and b is a signature size), the bit slice approach stores the signature file as b strings, each of length N. For example, we assume that a data file contains 5 records, and

their signatures are 001, 101, 110, 011, and 111 respectively. Both representations of this example are seen as below.

Bit string Representation	Bit slice Representation
001	01101
101	00111
110	11011
011	
111	

If a signature have b bits in size and a query contains w 1's, then with the bit slice representation, it is possible to check the relevant w bits of every record signature without fetching any of the other bits. Thus only wN rather than bN bits of the signature file need be examined on a query. This approach can contribute considerable savings, since typically w << b.

The second strategy for reducing the amount of signature file accessed is based on multilevel signature files, rather than single-level files. For very large data files, queries using the bit slice approach can still be expensive. For example, we suppose there is k=4 slices, each containing n = 500,000 bits and each page in secondary store has a capacity of p=8192 bits. As many as 248 disk accesses are required to determine the matching records, since the number of disk accesses is at least k*[n/p]. This shows that one-level method is somewhat expensive even if using a bit slice representation. By Sacks-Davis et al. [20], a two-level method is proposed for which a data file of N records is viewed as consisting of Ns blocks, each containing Nr records where N = Ns*Nr. Thus block signatures as well as record signatures must be stored in this method. A block signature is formed analogously to a record signature using all the terms of all the records contained in that block. The block signature is first examined on a query to identify which blocks of records match, and then the record signatures from the matching blocks are examined to identify the individual matching records. The block signature file is viewed as the bit slice representation to facilitate efficient query processing, whereas the record signatures are stored as the bit string one.

The third strategy uses special encoding schemes based on term discrimination. In general, many works [4,17,18] assume that the probability of any term in a user query is uniform (uniform query frequency assumption), as well as that the terms appear with equal frequency in the text (uniform occurrence frequency assumption). But Faloutsos & Christodoulakis [9] argued that both assumptions should be unrealistic. They improved the performance of the signature file method by allowing special treatment to terms with high discriminatory power (high query frequency and low occurrency frequency). And they demonstrated the 50% savings in false drops in case the 80-20 rule holds. If the number of false drops is kept constant, the savings in false drops result in the decrease in signature length. This after all reduces the search time for signature files.

The fourth strategy uses compression techniques on signatures. Although the bit slice representation is much faster than the bit string one, there may be room for improvement. On searching, each search term requires the retrieval of k bit slices. where k is the number of bits set to 1. The retrieval time can be improved if k is forced to be one. In that case, the signature size has to be increased in order to maintain the same false drop probability. But the corresponding bit slices will be sparse and they can be compressed. As a result, the small amount of space in signature files can still maintain the same false match. We finally classify the works concerning signature files from the viewpoint of above four strategies as shown in Table 1.

(N/A : Not Applicable)

		Bit String		Bit Slice	
		No Special Encoding	Special Encoding	No Special Encoding	Special Encoding
One-level	No Comp.	Files & Huskey [11]	Faloutsos & Christodoulakis [9]	Roberts [17]	N/A
	Comp.	Faloutsos [7]	N/A	Faloutsos & Chan [8]	N/A
Multi-level	No Comp.	Sacks-Davis & Ramamohanarao [18]	Sacks-Davis et al. [20]	Sacks-Davis & Ramamohanarao [18]	Sacks-Davis et al. [20]
	Comp.	N/A	N/A	Faloutsos & Chan [8]	N/A

Table 1. Classification of Related Works

3. New Multikey Access Methods

3.1 Framework

As described in the previous section, the two-level signature file method is an encouraging one to combine two strategies for efficiently accessing signature files. But there can be rooms for improvement. Faloutsos & Christodoulakis [9] distinguished terms with high discriminatory power from terms with low discriminatory power, which is called term discrimination, and made use of a special encoding scheme for the former terms to improve the performance of the signature file method. Here the terms with high discriminatory power are the ones which are frequently used in user queries but are infrequently occurring in data file, while the terms with low discriminatory power are opposite. Besides Sacks-Davis et al. [20] improved the two-level signature file method by identifying the terms in the database which occur most frequently (referred to as common terms) and by using a special encoding scheme for these terms in creating the block signature file. Therefore we take term discrimination into account to improve the two-level signature file method. However instead of using a special encoding scheme (i.e. the third strategy in section 2), we construct separate, efficient access methods, e.g. the inverted file, for terms with high discriminatory power.

In addition to four strategies described in section 2, the use of clustering method is another promising strategy to efficiently access signature files. In fact the clustering method is a dominating access method in library science [21]. Similar documents in the method are grouped together to form clusters. Usually they are stored physically together. However it seems that the clustering method can not handle insertion easily, and moreover may fail to retrieve some documents even though they qualify. We use the clustering method to cluster similar record signatures rather than similar records (documents), and therefore avoid above two problems to some extent. To make the clustering easy and effective, it is required to combine term discrimination and signature clustering. That is, we construct clusters by means of the similarity between only terms with high discriminatory power, rather than all the terms. This leads to fast retrieval to the records required by a user query.

Based on term discrimination and signature clustering, we present the framework of a multikey access scheme [2] as shown in Figure 1. In the following, we will call primary terms the terms with high discriminatory power and secondary terms the ones with low discriminatory power. Also we will call primary block descriptors the block descriptors for primary terms and secondary block descriptors the ones for secondary terms.

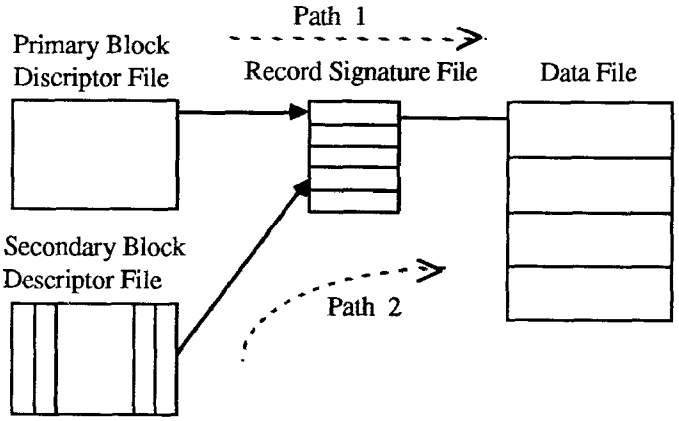

Figure 1. Framework of a New Multikey Access Scheme

In this scheme there are two block descriptor files, primary and secondary block descriptor files. For a query with a primary term, we must access the primary block descriptor file following path 1, while for a secondary term, we must follow path 2. Since there exist two paths to retrieve records in data file, it is necessary to select paths effectively in the case of queries with multiple terms. However for simplicity, queries with a single term are considered in this paper.

Depending on a way to construct the primary block descriptor file, we propose three different access methods, primary-signature-based two-level signature file Method, inversion-based two-level signature file method, and hash-table-based two-level signature file method. The proposed methods are described in detail next. On the other hand, we adopt the signature file method to construct the secondary block descriptor file. Record signatures are stored as a bit string representation, while the secondary block descriptor file is viewed as a bit slice one. Thus the path 2 is exactly the same as the two-level signature file method. If we use the compression technique for the primary block descriptor

file, we can combine four strategies in section 2 as well as the signature clustering strategy into the proposed methods. As a result, they become very efficient access methods for retrieving both formatted data and free texts. Table 2 gives a list of the names of the methods and their abbreviations.

TSM	Two-level Signature file Method
PTSM	Primary-signature-based Two-level Signature file Method
ITSM	Inversion-based Two-level Signature file Method
HTSM	Hash-table-based Two-level Signature file Method

Table 2. List of Methods and Abbreviations

3.2 PTSM

PTSM is an access method which is directly constructed using above scheme. As a primary block descriptor file, PTSM creates a separate signature file called primary block signature file and store it in a bit slice representation, like the secondary block signature file. Figure 2 illustrates this method.

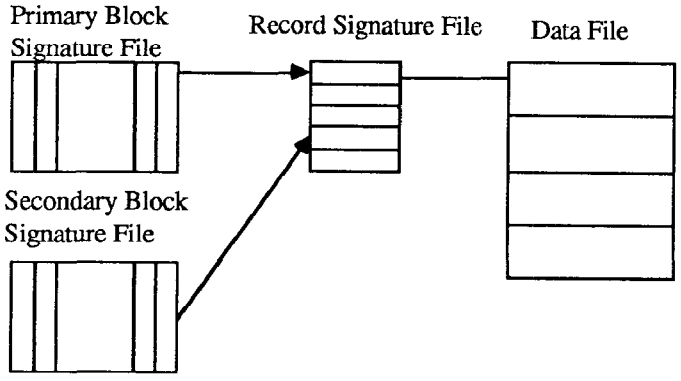

Figure 2. Illustration of PTSM

In this method there exists primary and secondary block signature files. By two separate block signature files, the method can allows as few false drops as the special encoding techniques described in section 2. That is, since primary terms yield their own block signature in their separate signature file, there is no false drop occurring when primary terms and secondary terms are mixed to create a block signature. In addition, as primary terms are frequently used in user queries, it is necessary to increase the size of primary block descriptors. This results in smaller false drops, thus providing good retrieval performance.

3.3 ITSM

Analogously to PTSM, ITSM also creates a separate block descriptor file for primary terms. But it adopts the inversion method, instead of the signature file method. Figure 3 illustrates this method.

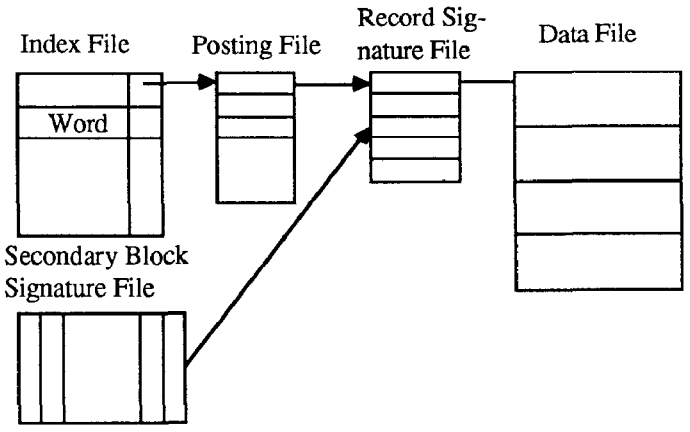

Figure 3. Illustration of ITSM

The motivation behind the method is to avoid false drops completely by storing actual terms in the primary block descriptor file. In addition ITSM produces great clustering benefits since it can contain much information needed for clustering. Thus ITSM achieves fast retrieval on user queries. However this method suffers from two problems of the inversion method: slow insertion and much storage overhead. But this is not critical since we construct the inverted file for only primary terms. A different point from a conventional inversion method is that a posting file points a block containing record signatures, rather than actual records.

3.4 HTSM

PTSM is one extreme which allows false drops but provides small storage overhead. On the contrary, ITSM is the other extreme which concentrates on fast retrieval by suppressing false drops completely but gives rise to much storage overhead. HTSM can be regarded as a bridge between above two extreme. This method

makes use of hash table to suppress false drops to a small degree. But it contains only pointers without storing actual terms, thus leading to relatively small storage overhead. Moreover HTSM is nearly as good on clustering effects as ITSM since it can store the information needed for clustering in the posting file. As a result, this method is considered very suitable because it provides good performance on both sides: retrieval side and storage overhead side. Figure 4 illustrates the method.

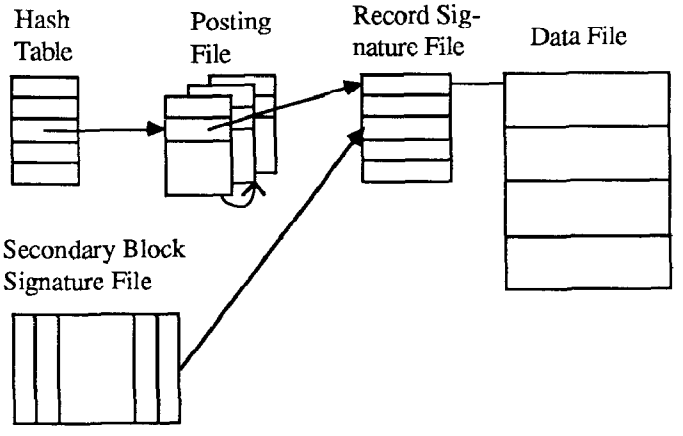

Figure 4. Illustration of HTSM

4. Analysis

Here we will provide the space-time analysis of proposed methods. In the following, we will require an estimate of the expected number of blocks containing r randomly chosen tokens, given that m is the number of tokens per block and n is the number of blocks. We define e(r,m,n) as the expected number of blocks containing the r tokens. Similarly, we define c(r,m,n) as the expected number of blocks containing the r tokens, when m*n tokens are clustered in n blocks according to their similarity. The input and design parameters for the analysis are given in Table 3.

Now we will examine the performance measures for given input and design parameters. The measures we are interested in are listed below:

- o R : number of disk accesses on retrieval. They include the disk accesses to search the descriptors, as well as to retrieve the qualifying records.
- o R_1 : number of disk accesses through path 1 (including the primary block descriptor)
- o R_2 : number of disk accesses through path 2 (including the secondary block descriptor)

Symbol	Definition
N	Total number of records
P	Page size in bits
A	Number of records that satisfy query
$\overline{A}$	Average Number of records that satisfy query
N_s	Size of the record signature file in blocks
N_r	Number of record signatures in a block(where $N = N_s * N_r$)
b_s, b_r, b_{s_1}, b_{s_2}	Size of a signature in bits, (block, record, primary block, secondary block respectively)
k_s, k_r, k_{s_1}, k_{s_2}	Number of bits set to '1' by a term (block, record, primary block, secondary block respectively)
F_s, F_r, F_{s_1}, F_{s_2}	False drop probability of a signature (block, record, primary block, secondary block respectively)
α	Probability that primary terms are used in user queries on the average
s	Average number of all the terms in a record
r	Average number of primary terms in a record
V_p	Total number of distinct primary words in all the collection of records
H^p	Hash table size
t	Pointer size in bytes
$\overline{w}$	Average word length in bytes
h	Height of inverted file (B-tree)
$\overline{c}$	Average chain length in posting file

Table 3. Input and Design Parameters

o O_v : disk storage overhead in pages
o I : number of disk accesses on insertion

4.1 TSM

o Retrieval

$$R = k_s * \left\lceil \frac{N_s}{P} \right\rceil + (e(A,N_r,N_s) + N_s * F_s)(1 + N_r * F_r) + A$$

o Storage Overhead

$$O_v = \left\lceil \frac{b_s * N_s}{P} \right\rceil + N_s$$

o Insertion

$$I = s * k_s + 2$$

4.2 PTSM

o Retrieval

$$R = \alpha * R_1 + (1 - \alpha) * R_2$$

$$R_1 = k_{s_1} * \left\lceil \frac{N_s}{P} \right\rceil + (c(A,N_r,N_s) + N_s * F_{s_1})(1 + N_r * F_r) + A$$

$$R_2 = k_{s_2} * \left\lceil \frac{N_s}{P} \right\rceil + (e(A,N_r,N_s) + N_s * F_{s_2})(1 + N_r * F_r) + A$$

o Storage Overhead

$$O_v = \left\lceil \frac{b_{s_1} * N_s}{P} \right\rceil + \left\lceil \frac{b_{s_2} * N_s}{P} \right\rceil + N_s$$

o Insertion

$$I = r * k_{s_1} + (s - r) * k_{s_2} + 2$$

4.3 ITSM

o Retrieval

$$R = \alpha * R_1 + (1 - \alpha) * R_2$$

$$R_1 = h + \bar{c} + c(A,N_r,N_s)(1 + N_r * F_r) + A,$$

where $\bar{c} = \left\lceil \frac{c(\bar{A},N_r,N_s) * t}{P} \right\rceil$ and

$h = \lceil \log_d V_p \rceil$, where $d = \left\lfloor \frac{P - t}{\bar{w} + t} \right\rfloor$ is the branching degree of B-tree. (Here B-tree is considered as the index file of the inversion method).

$$R_2 = k_{s_2} * \left\lceil \frac{N_s}{P} \right\rceil + (e(A,N_r,N_s) + N_s * F_{s_2})(1 + N_r * F_r) + A$$

o Storage Overhead

$$O_v = \left\lceil \frac{(\bar{w} + t) * V_p}{P} \right\rceil + \bar{c} * \left\lceil \frac{V_p}{\bar{n}} \right\rceil + \left\lceil \frac{b_{s_2} * N_s}{P} \right\rceil + N_s,$$

where $\bar{n}$ is the average number of chains in a page,

$$\bar{n} = \left\lfloor \frac{P}{c(\bar{A},N_r,N_s) * t} \right\rfloor \text{ if } c(\bar{A},N_r,N_s) \leqslant P$$
$$= 1 \text{ otherwise.}$$

o Insertion

$$I = r * (h + \bar{c}) + (s - r) * k_{s_2} + 2$$

4.4. HTSM

o Retrieval

$$R = \alpha * R_1 + (1 - \alpha) * R_2$$

$$R_1 = 1 + \bar{c} + (c(A,N_r,N_s) + N_s * F_h)(1 + N_r * F_r) + A,$$

where $F_h \simeq \frac{r}{H}$ is the false drop probability in hash table.

$$R_2 = k_{s_2} * \left\lceil \frac{N_s}{P} \right\rceil + (e(A,N_r,N_s) + N_s * F_{s_2})(1 + N_r * F_r) + A$$

o Storage Overhead

$$O_v = \left\lceil \frac{H * t}{P} \right\rceil + \bar{c} * \left\lceil \frac{V_P}{\bar{n}} \right\rceil + N_s,$$

where $\bar{c}$ and $\bar{n}$ are the same as those in ITSM.

o Insertion

$$I = r * (1 + \bar{c}) + (s - r) * k_{s_2} + 2.$$

5. Comparison

In order to show the efficiency of the proposed methods, we will compare TSM with PTSM, ITSM, and HTSM. For the comparison, we consider an example database shown in Table 4. It is formed based on Sacks-Davis' database [19,20] and the 80-20 rule [9]. Using this database, we obtain the performance results as presented in table 5.

N = 1,048,576 (number of records)		
s = 20 (number of attributes)		
P = 32768 (page size (4K bytes))		
L = 1024 bytes (average record size)		
t = 4 bytes		
$\overline{w}$ = 6 bytes		
[Two-level Method]		
$N_S = 16384$		
$N_r = 90$		
$k_s = 5$	$k_r = 11$	
$b_s = 58066$	$b_r = 332$	
$F_S = 1/N_S$	$F_r = 1 / 29\ N_r$	
[Proposed Methods]		
$k_{s_1} = 3$	$k_{s_2} = 5$	
$b_{s_1} = 26888$	$b_{s_2} = 4645$	
$F_{s_1} = 1 / N$	$F_{s_2} = 1 / N_S$	
$V_p = S = 150{,}000$	$\alpha = 0.8$	(80-20 rule)
$\overline{A} = 200$	$r = 4$	(80-20 rule)

Table 4. Example Database

Measures / Methods	Retrieval Time (# of disk access)	Storage Overhead (bytes/record)	Insertion Time (# of disk access)
TSM	5.03+1.03B+A	177.4	102
PTSM	4.43+0.82C_1 +0.21B+A	207.2	94
ITSM	3.61+0.82C_2 +0.21B+A	273.4	94
HTSM	3.02+0.82C3 +0.21B+A	272.5	90

A : Number of qualifying records

B : e(A, N_r , N_s) in all methods

C_1, C_2, C_3 : C(A, N_r, N_s) in PTSM, ITSM, and HTSM, respectively

Table 5. Performance Results

It is shown from the results that the main difference of the proposed methods from TSM comes from the signature clustering benefits described as C_1, C_2, and C_3. Here c(A, N_r, N_s), the common expression of C_1, C_2, and C_3, is the expected number of blocks containing about A matching record signatures, when N record signatures are clustered in N_s blocks according to the similarity of primary terms. Therefore e(A, N_r, N_s) $\geq$ c(A, N_r, N_s), that is B $\geq C_1$, C_2, and C_3. We let c(A, N_r, N_s) = A/β, where β is a clustering factor which indicates the degree of signature clustering. As β increases, more record signatures are clustered in a single block, thus resulting in better performance on retrieval. On the contrary, as A increases, the retrieval performance decreases since B becomes relatively small when A is large.

On the other hand, there also exists the difference among the proposed methods on retrieval performance. The difference is mainly made by the value of C_1, C_2, and C_3. C_1, C_2, and C_3 are the values dependent on the clustering degree of PTSM, ITSM, and HTSM respectively. In general, ITSM produces the greatest clustering benefits because it can contain much information required for clustering as well as can avoid false drop completely. And HTSM is nearly as good on clustering effects as ITSM. Thus we can state that $C_2 \leq C_3 \leq C_1$. As a result, ITSM requires the smallest disk accesses to retrieve the qualifying records for a given query.

Here we will compare ITSM with TSM to show the superiority of the proposed methods. Table 6 presents the ratio of the decreased disk accesses of ITSM over TSM, with respect to β and A. In addition Figure 5 describes the disk accesses on retrieval with respect to A when β = 2 and 3 respectively [19]. It is seen from the comparison that ITSM achieves 15-30 % faster retrieval than TSM when A = 1-10000 and β = 2-3. Since the comparison is mainly focused on the clustering benefits, PTSM and HTSM will show similar results when C_1 and C_3 are the same as C_2. Meanwhile the proposed methods require more storage overhead than TSM. Here the average record size is 1024 bytes as presented in our example database in Table 4. Therefore the ratios of the increased storage overhead of PTSM, ITSM, and HTSM over TSM are 3.9, 9.4, and 9.3 %

respectively. In terms of insertion time, the proposed methods require about 10 % smaller disk accesses than TSM.

(Unit : %)

A \ β	1.5	2	2.5	3	3.5	4
1	30.0	30.0	30.0	30.0	30.0	30.0
10	19.6	24.8	27.9	30.0	31.4	32.5
100	14.1	20.7	24.6	27.2	29.1	30.5
1000	12.5	19.4	23.5	26.2	28.1	29.6
10000	3.9	11.6	16.2	19.3	21.5	23.1

Table 6. Ratio of Decreased Disk Accesses of ITSM over TSM

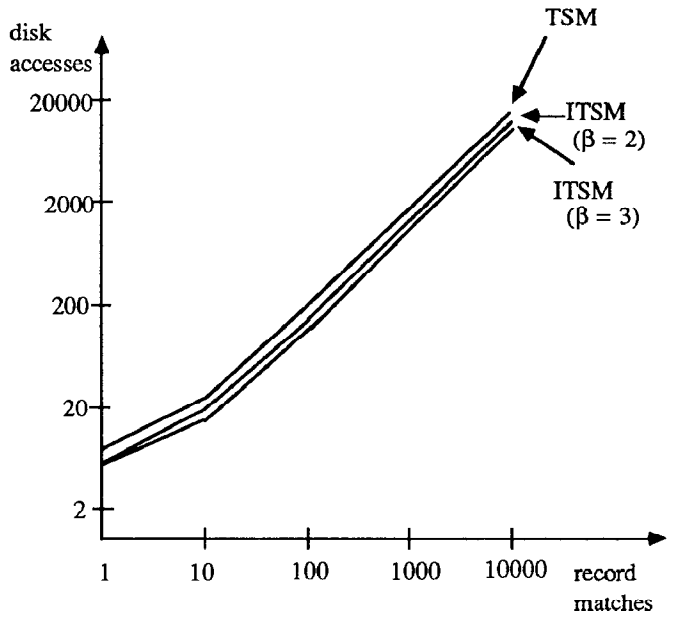

Figure 5. Disk Accesses on Retrieval when β = 2 and 3

6. Conclusions and Further Works

In order to improve the two-level signature file method, we proposed new multikey access methods using term discrimination and signature clustering. By term discrimination, we can separate primary terms from all the terms used for indexing records. Thus it is possible to create new, efficient access methods for primary terms, so that we can achieve good performance on retrieval. In addition, we can cluster similar record signatures to provide better performance. To make the clustering easy and effective, we can form clusters by means of the similarity between primary terms, rather than all the terms.

Using above two concepts, we proposed three different access methods depending on a way to construct the primary block descriptor file: PTSM, ITSM, and HTSM. PTSM gives small storage overhead while ITSM concentrates on fast retrieval. HTSM is a bridge between both extremes which can be considered as a promising method to provide good performance on both sides: retrieval side and storage overhead side. We in addition provided the space-time analysis of the proposed methods and compared them with the two-level signature file method. We showed from the comparisons that the proposed methods achieved 15-30 % gains on retrieval performance when the clustering factor is 2 and 3, and required 3-9 % more storage overhead.

Now in order to validate the space-time analysis of the proposed methods, we are implementing the PTSM, ITSM, and HTSM method. Further works can deal with the following issues:

- Performance analysis on multi-term queries
- Efficient path selection for multi-term queries
- Performance comparison with the conventional inversion method.

Reference

[1] P.B. Berra et al., "Computer Architecture for a Surrogate File to a Very Large Data/Knowledge Base," IEEE Computer, Vol. 20, No. 3, Mar. 1987, 25-32.

[2] J.W. Chang and Y.J. Lee,"Multikey Access Scheme based on Term Discrimination and Signature Clustering," International Symposium on Database Systems for Advanced Applications, Apr. 1989, to be appeared.

[3] R.M. Colomb and J. Jayasooriah, "A Clause Indexing System for Prolog Based on Superimposed Coding," The Australian Computer J., Vol. 18, No. 1, Feb. 1986, 18-25.

[4] S. Christodoulakis and C. Faloutsos, "Design Considerations for a Message File Server," IEEE Trans. Soft. Eng., SE-10, No. 2, 1984, 201-210.

[5] B.C. Desai, T. Goyal and F. Sadri, "A Data Model for Use with Formatted and Textual Data," J. Am. Soc. Inf. Sci.(JASIS), Vol. 37, No. 3, May 1986, 158-165.

[6] C. Faloutsos, "Access Methods for Text," Computing Survey, Vol. 17, No. 1, Mar. 1985, 49-74.

[7] C. Faloutsos, "Signature Files: Design and Performance Comparison of Some Signature Extraction Methods," ACM SIGMOD, May 1985, 63-82.

[8] C. Faloutsos and R. Chan, "Fast Text Access Methods for Optical and Large Magnetic Disks: Design and Performance Comparison," VLDB, Aug. 1988, 280-293.

[9] C. Faloutsos and S. Christodoulakis, "Design of a Signature File Method That Accounts for Non-Uniform Occurrence and Query Frequencies," VLDB, Aug. 1985, 165-170.

[10] C. Faloutsos and S. Christodoulakis, "Description and Performance Analysis of signature File Methods," ACM TOOIS, Vol. 5, No.3, 1987, 237-257.

[11] J.R. Files and H.D. Huskey, "An Information Retrieval System Based on Superimposed Coding," Proceedings of the Fall Joint Computer Conference, AFIPS Press, 1969, 423-432.

[12] R.L. Haskin and R.A. Lorie, "On Extending the Functions of a Relational Database System," ACM SIGMOD, Jun. 1982, 207-212.

[13] A. Kent, R. Sacks-Davis, and K. Ramamohanarao, "A Superimposed Coding Scheme Based on Multiple Block Descriptor Files for Indexing Very Large Data Bases," VLDB, Aug. 1988, 351-359.

[14] D.E. Knuth, "The Art of Computer Programming, vol 3: Sorting and Searching," Addison-Wesley, 1973.

[15] I. A. Macleod and A. R. Reuber, "The Array Model: A Conceptual Modeling Approach to Document Retrieval," J. Am. Soc. Inf. Sci.(JASIS), Vol. 38, No. 3, May 1987, 162-170.

[16] K. Ramamohanarao, and J. Shepherd. "A Superimposed Codeword Indexing Scheme for Very Large Prolog Databases," Proceedings of International Symposium on Logic Programming, 1986, 569-576.

[17] C.S. Roberts, "Partial-Match Retrieval via the Method of Superimposed Codes," Proc. IEEE 67, Dec. 1979, 1624-1642.

[18] R. Sacks-Davis and K. Ramamohanarao, "A Two Level Superimposed Coding Scheme for Partial Match Retrieval," Information Systems, Vol. 8, No. 4, 1983, 273-280.

[19] R. Sacks-Davis and K. Ramamohanarao, "Performance of a Multi-Key Access Method Based on Descriptors and Superimposed Coding Techniques," Information Systems, Vol. 10, No. 4, 1985, 391-403.

[20] R. Sacks-Davis et al., "Multikey Access Methods Based on Superimposed Coding Techniques," ACM Tran. Database Systems, Vol. 12, No. 4, Dec. 1987, 655-696.

[21] G. Salton and M.J. McGill, "Introduction to Modern Information Retrieval," McGraw-Hill, 1983.