Partitioned Signature File: Design Issues and Performance Evaluation[1]

Dik Lun Lee Chun-wu Leng

Department of Computer and Information Science
The Ohio State University
2036, Neil Ave
Columbus, OHIO 43210-1277

January 1989

Abstract

A signature file acts as a filtering mechanism to reduce the amount of text that needs to be searched for a query. Unfortunately, the signature file itself must be exhaustively searched, resulting in degraded performance for large file size. We propose to use a deterministic algorithm to divide a signature file into partitions, each of which contains signatures with the same "key." The signature keys in a partition can be extracted and represented as the partition's key. The search can then be confined to the subset of partitions whose keys match with the query key. The main concern of this paper is to study methods for obtaining the keys and their performance in terms of their ability to reduce the search space.

Due to the reduction of search space, partitioning a signature file has direct benefit in a sequential search (single-processor) environment. In a parallel environment, search can be conduct in parallel effectively by allocating one or more partitions to a processor. Partitioning the signature file with a deterministic method (as opposed to a random partitioning scheme) provides *intra-query* parallelism as well as *inter-query* parallelism.

In this paper, we outline the criteria for evaluating partitioning schemes. Three algorithms are described and studied. An analytical study of the performance of the algorithms is provided and the results verified with simulation.

Categories and Subject Descriptors: H.2.1 [**Database management**]**:** Physical Design - *access methods*; H.3.6 [**Information Storage and Retrieval**]**:** Library Automation; H4.1 [**Information Systems Applications**]**:** Office Automation; I.7.0 [**Text Processing**]**:** Text Editing

General Terms: Design, Performance

Additional Keywords and Phrases: Document retrieval, information retrieval, office automation, surrogate file, superimposed coding, text retrieval, access method, parallel search, performance evaluation

[1]This research has been supported by the Applied Information Technologies Research Center (AITRC), Columbus, Ohio, project no. EES 4509 and EES 4628.

1 Introduction

Although the application of signature files, in particular, the superimposed coding method [15], in information retrieval has been studied for a long time, it has recently received increasing attention in both the traditional and new application domains. Traditional library information systems have a constant demand to provide full text retrieval, faster response time, and better services to the users. Applications in office information systems and artificial intelligence require large quantity of information, most of them without a fixed format and organization, to be manipulated efficiently [2,19].

A signature file is basically an abstraction of a text file. It is searched in the first phase of a search to eliminate most of the unqualified documents. The second phase further verifies the documents which passed the first phase against the query. The main advantage of signature files is its simple file structure and low storage overheads. Unlike inverted files which may have more than 100% storage overhead, a signature file is typically 10-20% the size of the primary database and therefore very suitable for large databases [3].

A distinct characteristic of the signature file is that it produces false drops. That is, the signature file may indicate that a record contains a particular keyword but in fact it does not. A high false drop rate causes unnecessary disk accesses and lowers the overall system performance. Therefore, a substantial amount of efforts has been devoted to the study of optimal algorithms for generating signatures from the text and the estimation of false drop probability [6,7]. The goal of these efforts is to improve the overall system performance by reducing the false drop rate.

In this paper, we take another direction for improving the performance of the signature file technique. We examine ways to improve the speed of searching the signature file itself. Our approach is orthogonal to that of reducing false drops in the sense that they can be combined together to yield an even better performance.

The main purpose of a signature file is to reduce the search space in the primary database. However, searching the signature file itself may be slow. This is because the size of the signature file is directly proportional to the size of the database, and retrieving all records (documents or database records) satisfying a query requires exhaustively searching the whole signature file. Therefore, the time spent on searching the signature file is directly proportional to the database size, and the performance will become a concern for large databases.

Work has been done on improving the speed of searching the signature file [11]. A transposed file organization has been suggested for the storage of the signature file to avoid reading unnecessary bits from the storage [15]. Ahuja and Roberts proposed a word-serial, bit-parallel hardware processor to expedite the comparison of signatures against the query [1]. Lee proposed a word-serial, bit-parallel signature processor which combined the advantages of a transposed organization and a parallel hardware implementation [9]. The performance of a signature processor can be further enhanced by using more than one processor module operating in parallel, each of which holds and searches only a partition of the signature file. In Ahuja and Roberts' design, modules are connected to a bus and activated in parallel for searching when a query signature is broadcast to them. Thus, the time for searching the whole signature file is approximately the time to search one partition when each partition has about the same size. The same parallel search method can be applied to Lee's design. Commercial parallel systems have also been employed for parallel text search using signature files as a filtering mechanism [18].

These approaches still require examining every signature in the signature file to determine which signature qualifies. This is acceptable only when the database is small. For large database, these approaches become very inefficient, for precisely the same reasons as the inefficiency of an exhaustive search of the primary database itself.

Multi-level signature files have been proposed to alleviate the problem [13,16,17]. The tree-like structure of the signature files is similar to that of a search tree in that higher level signature files perform coarse filtering before the signature files at the lower levels are consulted. This method eliminates the need of examining every signature in the signature files at lower levels. However, the additional levels of signature files require extra storage and processing overhead. For queries with a large number of qualified signatures, this approach may be even slower than a single-level signature file due to the overhead incurred in the multi-level file organization. By putting "similar" signatures in groups and organizing the groups in a B-tree, the s-tree technique [4] achieves a performance improvement but still shares the same weakness as the multi-level signature file method.

In this paper, we examine methods for dividing the signature file into partitions, each of which holds signatures sharing the same "key." As a result, we can determine if the signatures in a partition will satisfy a query by merely examining the key; partitions not matching the key need not be searched. This method is similar to hashing in the sense that signatures are hashed into partitions according to the keys computed algorithmically from the signatures. Unlike standard hashing, the search criteria used for a signature file is not an exact match. This introduces some unique and interesting problems for the partitioned signature file method.

A partitioning method has obvious advantages in a sequential, single-processor, environment, because not every partition needs to be searched always. In addition, unlike the multi-level signature file structure, the partitioning technique may use less storage than a single-level signature file (therefore much less than that of a multi-level signature file), because the keys of the signatures in a partition are identical and can be derived from the partition's key. Furthermore, the partitioning algorithms require a negligible amount of processing time while searching a multi-level signature file requires substantial processing and multiple disk accesses. Unlike a multi-level signature file, the time to search the partitions is strictly less than that time to search a single-level signature file because there is little or no overhead involved compared to a multi-level signature file organization.

A partitioned signature file can be implemented in a parallel environment by storing one partition in one processor module (with adequate internal and/or external storage). Without a suitable partitioning scheme, all partitions must be searched indiscriminately and simultaneously. Although this improves the speed of one query by introducing *intra*-query parallelism, only one query can be serviced at a time (unless the queries are batched) and no *inter*-query parallelism can be exploited. Since only a subset of the partitions are searched for a query in our scheme, only those processors containing partitions matching the key of the query need to be activated for searching. Inactive processors will then be available to service other queries. Allowing both *intra*- and *inter*- query parallelism is the major advantage of our parallel search method over others proposed in the literature [1,9].

The main problem in our method is to find a partition scheme which allows us to search as few partitions, and as few signatures, as possible. In this paper, we examine several methods of partitioning the signature file and evaluate their characteristics and performance. An overview of the superimposed coding method is given in Section 2. Readers not familiar with the subject are encouraged to further consult the literature [5,11,19,20]. The principle of the partitioned signature file is presented in Section 3. In Section 4, we outline the criteria according to which partitioning algorithms are evaluated. Three partitioning algorithms are presented in Section 5. The performance of the algorithms are studied both analytically and by simulation. Section 6 provides a conclusion of the study.

2 Superimposed Coding

There are a number of different methods for constructing a signature file from a database. Our partition methods are based on the superimposed coding method [15]. In this section, we provide a brief overview of the superimposed coding method. A summary of the symbols used in this paper can be found in Table 1.

A database is divided into a number of blocks. The length of the blocks are controlled so that each block contains about the same number of unique, non-trivial, words. A block B_i is associated with a signature S_i which is a fixed-length bit vector $S_i^1 S_i^2 ... S_i^m$, where m is the length of the signature and $S_i^j \in \{0, 1\}$. The signature is abstracted from the block by means of a hashing function and can be considered as a condensed representation of the block.

Figure 1 is an example illustrating the process of generating a block signature from a text block. Each non-trivial word in the text block is hashed into a bit vector, called a *word signature*, by selecting a fixed number of bit positions (with replacement) from the word signature and setting them to one. The *block signature* (or signature for short), S_i, is generated by superimposing all word signatures for that block and put in a signature file which stores all signatures generated from the database.

When a keyword is searched for, a *query signature*, S_Q, of the word is generated from the query, Q, in the same way as described above. S_Q is then compared with every signature from the signature file. A signature, S_i, is *qualified*, if and only if for all bit positions in

k	length of a key
K_{M_i}	key for the i^{th} partition
K_{S_i}	key obtained from the i^{th} block signature
$K_{S_i}^V$	key value, the bit string which forms the key
$K_{S_i}^P$	key position, the starting position of the key
K_{S_Q}	key obtained from the query signature
m	length of a signature
M_i	i^{th} partition
P_a	probability that a partition is activated
S_i	i^{th} signature
$\bar{S}_i$	non-key portion of the i^{th} signature
S_Q	query signature
$\bar{S}_Q$	non-key portion of a query signature
w_X	no. of ones (weight) of bit vector X
$\bar{w}_X$	no. of zeros of bit vector X
z	no. of zeros in a key

Table 1: List of Symbols.

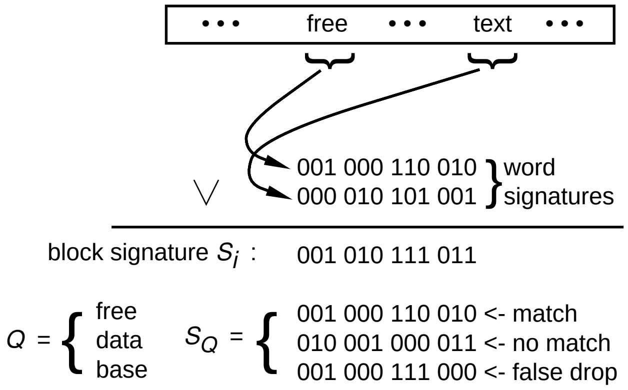


Figure 1: Generating signatures from a text block.

S_Q which are set to 1, the corresponding bit positions in S_i are also set to 1 (known as the *inclusion* condition). Formally, the set of qualified signatures is defined by:

$$\{S_i \mid (S_i \wedge S_Q) = S_Q\} \tag{1}$$

In this formula bitwise conjunction is used. A disqualified signature indicates that the associated text block is guaranteed not to contain the word specified in the query. However, a qualified signature only means that the corresponding text block is very likely to satisfy the query. That is, there is a small chance that the text block may not contain the word being searched for (i.e., a false drop). Therefore, the text block has to be searched to eliminate false drops. The three queries in Figure 1 (i.e., "free," "data," and "base") illustrate these three different cases.

3 Signature File Partitioning

As can be observed in the above discussion, the size of the signature file and therefore the search time involved are linear with the size of the text file. Dividing the signature file into random partitions and searching the partitions in parallel have been proposed to improve the speed [1,9]. However, a random partitioning scheme still requires every signature in every partition to be examined.

In this paper, we investigate the partitioning of a signature file with a deterministic algorithm which relieves us from searching every partition in order to answer a query and thus reducing the search space involved. In addition, since a subset of the partitions is searched for each query, a parallel implementation will be able to accommodate multiple queries simultaneously, resulting in both *intra-* and *inter*-query parallelism.

The main problem in signature file partitioning is to determine the partitioning scheme to map signatures into partitions. The process is similar to hashing but the retrieval operation is quite different from that of hashing. Standard hashing looks for an exact match of a key in the hash table, whereas the inclusion condition in Equation (1) must be satisfied in a signature file. Alternately, if we view a signature of m bits as a tuple of m 1-bit attributes, the retrieval of signatures is similar to partial match retrieval [8,14]. However, due to the large value of m, methods for allocating tuples to buckets in partial match retrieval are impractical for allocating signatures into partitions.

To simplify our discussion and analysis, we assume that S_i's and S_Q are random bit vectors uniformly distributed in the range of 0 to 2^{m-1}; each bit position is equally likely to have a 0 or 1 on it. Since S_i's are random bit vectors, substrings taken from S_i's are themselves random bit vectors and hence can be used as keys. Let K_{S_i} and K_{S_Q} be the keys for a block signature, S_i, and a query signature, S_Q, respectively; and $\bar{S}_i$ and $\bar{S}_Q$ denote the non-key portions of S_i and S_Q respectively. Signatures having the same key are grouped into the same partition. Since a partition contains signatures of the same key, we use K_{M_i}

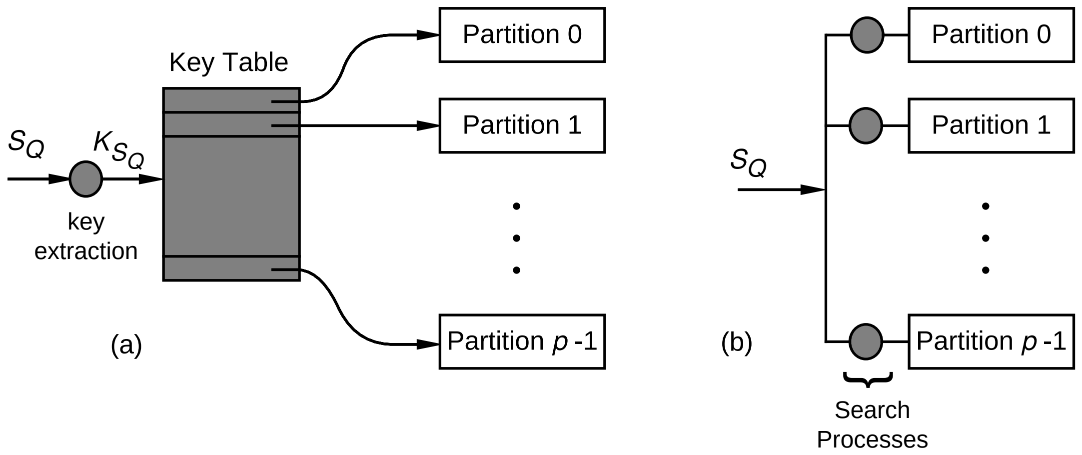


Figure 2: The partitioned signature file: (a) sequential and (b) parallel implementation.

to refer to the key of the partition, M_i, and k to denote the length of a key. More precisely,

$$M_i = \{\bar{S}_i \mid K_{S_i} = K_{M_i}\} \tag{2}$$

Note that only $\bar{S}_i$ are stored in the partition. The original signature can be constructed from $\bar{S}_i$ and K_{M_i}. Therefore, a partitioned signature file may take less storage than the original signature file. When no ambiguity arises, we simple call $\bar{S}_i$ a signature.

When a query signature arrives, the query key is obtained by extracting the substring from the query signature at the same position at which the key of the partition is obtained. The query key is then compared to the partition keys. A partition M_i is searched when K_{M_i} includes K_{S_Q}:

$$(K_{S_Q} \wedge K_{M_i}) = K_{S_Q} \tag{3}$$

Condition (3) is checked first. Partitions satisfying (3) are further searched for signatures with $\bar{S}_i$ satisfying $\bar{S}_Q$. The set of partitions searched by a query is called the *working set* of the query.

Figure 2 illustrates two possible system configurations. In the sequential implementation, the keys of the partitions are stored in a key table and matched sequentially with the query key. The qualified partitions are further searched for qualified signatures. We find that implementing the key table as a trie provides better speed and flexibility than a straight table [10]. The key table is usually small enough to fit in the main memory (given the size of main memory available in contemporary workstations) and therefore contributes little to the response time which is dominated by disk accesses. In a parallel environment, each partition (or a small group of partitions) is searched with a processor. The extraction and comparison of keys can be done within each processor. Therefore, the key table is not needed (or it can be considered distributed into the processors and searched in parallel).

4 Criteria for Evaluation

Two system configurations are to be considered. The first is a sequential environment in which all partitions are stored on disk and searched sequentially with one processor. The other is a parallel implementation in which partitions are searched simultaneously. We consider the following criteria for signature file partitioning, bearing in mind that these criteria are conflicting to each other and that compromise must be made according to the objectives of a particular implementation.

• *reduction of search space*: reduction of search space is the primary motivation for using a partition scheme. The performance can be evaluated by two ratios: (1) the ratio of the number of signatures searched to the maximum number of signatures (*the signature reduction ratio*); and (2) the ratio of the number of partitions searched to the maximum number of partitions (*the partition reduction ratio*). The former is an important measure for the reduction of search space. When a partition is retrieved in one disk access, the latter indicates the number of disk accesses required. It is also an important measure in a parallel environment, since a higher degree of concurrency can be achieved if the number of partitions searched for a query is small.

• *uniform distribution of workload*: in a parallel environment in which partitions are searched in parallel with multiple processors, it is important to make the workload on each processor as uniform as possible. In systems where one partition is assigned to one processor and synchronization must be kept to a minimum (e.g., in a connection machine [18]), this requirement means that the probability that a partition is activated by a query should be as uniform as possible. Partitions which are more likely to be searched than others would cause a higher load on their corresponding processors. When more than one partition can be assigned to one processor, a variation of activation probability can be balanced by chosing an appropriate processor assignment strategy. However, for simplicity, we assume a one-to-one mapping between partitions and processors. A uniform workload is desirable but is less significant in a sequential system.

• *uniformity of partition size*: it is desirable to have partitions of uniform size for implementation as well as performance reasons. This means that the probability that a signature is distributed into a partition is about the same for all partitions. This criterion is particular important in a parallel implementation in which the search time is the time to search the largest partition. This criterion also helps us to allocate storage resources to a partition.

• *number of partitions required*: the number of partitions required should be within the limit imposed by the implementation. Given the number of files that is typically found in large computer installation and the current technology of parallel systems (e.g., connection machines), we can assume that a fairly large number of partitions can be accommodated.

Among these criteria, the reduction of search space is our primary concern. Uniformity of activation affects the workload distribution and therefore the degree of parallelism in a parallel implementation. The number of partitions required and the uniformity of partition size are implementation issues which we consider secondary to the first two criteria.

5 Algorithms and Performance Evaluation

The performance of a partitioning algorithm depends on how the key is selected. Since the signatures are themselves random bit vectors, the signature key can simply be a substring selected from the signature. Essentially, two parameters must be determined: the key's starting position in the signature and its length. This observation leads us to four different partition algorithms: (1) keys with a fixed length and fixed starting position, (2) keys with variable lengths but fixed starting position, (3) keys with a fixed length but variable starting positions, and (4) keys with variable lengths and variable starting positions. For the first two methods, we may arbitrarily fix the starting position to the leftmost bit so that the keys are prefixes of the signatures. Therefore, we call these methods *fixed prefix* and *extended prefix* methods, respectively. The third method uses fixed-length keys but the starting positions of the keys are chosen so that the key contains the largest possible number of zeros. We call this the *floating key* method. Both the lengths and the starting positions of a key may vary in the last method. This method generates a very large number of choices for obtaining a key. This will result in an excessively large number of partitions and thus a large amount of overhead in managing the partitions. Therefore, we will not consider this method any further.

In this section, we describe and study in detail the first three methods: fixed prefix, extended prefix, and floating key method. The algorithms are analyzed mathematically according to the criteria outlined in the last section. A simulation is performed and the results compared to the numerical results from the analytical formula. In the simulation, 100,000 signatures were generated randomly and partitioned using the three algorithms. Query signatures are also generated randomly with different query weights, and the number of signatures and the number of partitions searched are collected. The signatures are generated by setting each bit in the signature to 0 or 1 with equal probability using a random number generator. The number of one's in the signatures follows a binomial distribution, as shown in earlier studies [3,12].

In the analysis of the algorithms, we need to define the following function:

$$\chi(i,j) = \begin{cases} 1 & P(i,j)\cdot n \geq 1 \\ P(i,j)\cdot n & \text{otherwise} \end{cases}$$

$P(i,j)$ is the probability that a signature generated randomly will be hashed into a partition whose key has a weight of i and a starting position of $j \cdot k + 1$, where k is the length of the key. For example, if there are 2^k possible partitions and each partition has an equal chance to receive an arbitrary signature, then $P(i,j)$ will be 2^{-k} for every partition.

The purpose of using the auxiliary function χ is that not every possible partition is generated when a certain number of signatures is partitioned. This is particularly true when

n is small compared to the number of all possible partitions generated by a partitioning scheme. As shown later, the floating key method is one such example. When the expected number of signatures hashed into a partition is greater than or equal to one when n signatures are partitioned, the partition is assumed to be in existence; otherwise, the partition only has a chance of χ ($\chi < 1$) to exist and therefore only has a fractional contribution to the final result being computed. From the definition of $P(i,j)$, $\chi(i,j)$ is an approximation of the probability that a partition whose key weight is i and whose key's starting position is $j \cdot k + 1$ will contain at least one signature (and thus generated) when n signatures are partitioned.[2]

The value of j depends on the partitioning algorithm. For instance, $j = 0$ for the fixed and extended prefix method, and j is equal to 0, 1, 2, ..., $\lfloor m/k \rfloor$. The value of $P(i,j)$ for a partition is equal to the maximum size of the partition divided by the number of all possible signatures. We will obtain its expression for each method below.

5.1 Fixed Prefix Partitioning

The simplest way to select a key is to take the first k bits of a signature as the key. Signatures with the same k-prefix are grouped in the same partition. Likewise, the first k bits from S_Q can be taken as K_{S_Q}. Only partitions whose keys include the query's key are searched further.

Lemma 1 $P(i,j) = P(i,0) = 2^{-k}$.

Proof: The maximum partition size is 2^{m-k} when the key length is k, and the maximum number of signatures is 2^m. $\square$

A corollary immediate from this lemma is that every partition has the same expected size. Further, $\chi(i,j) = \chi(i,0)$ is independent of i.

Theorem 1 *Activation probability: the probability that a partition is searched for, when the partition's key has weight $w_{K_{Mi}}$ and the query key has weight w_{S_Q}, is:*

$$P_a = \frac{\binom{m - \bar{w}_{K_{Mi}}}{w_{S_Q}}}{\binom{m}{w_{S_Q}}}$$

where $\bar{w}_{K_{Mi}} = (k - w_{K_{Mi}})$.

Proof: From the inclusion condition, it can be observed that partition M_i is activated if and only if *no* zero-bit position in its key is specified in S_Q. That is, all w_{S_Q} bits are set on

[2] the probability that a partition contains at least one signature is $1 - (1 - P(i,j))^n \approx P(i,j) \cdot n$.

positions other than the $\bar{w}_{K_{Mi}}$ zero positions in the key. The numerator is the number of such possibility. The denominator is the number of possible query signatures with a fixed weight. $\square$

Theorem 2 *Average size of a working set: The average number of partitions searched per query is* $\sum_{i=0}^{k} \binom{k}{i} P_i \cdot \chi(i,j)$ *where* $P_i = \binom{m-k+i}{w_{S_Q}} / \binom{m}{w_{S_Q}}$.

Proof: From Theorem 1, we can see that P_i is the probability that a partition with i ones in its key is activated when the query key has w_{S_Q} ones in it. Since the number of partitions with i ones in the partition keys is $\binom{k}{i}$ and the activation of each partition is independent to each other, the average number of activated partitions is:

$$\binom{k}{0} P_0 \cdot \chi(0,0) + \binom{k}{1} P_1 \cdot \chi(1,0) + ... + \binom{k}{k} P_k \cdot \chi(k,0) = \sum_{i=0}^{k} \binom{k}{i} P_i \cdot \chi(i,0)$$

$\square$

Theorem 3 *The ratio of the number of partitions searched to the number of partitions generated is* $\sum_{i=0}^{k} \binom{k}{i} P_i \cdot 2^{-k}$ *where* $P_i = \binom{m-k+i}{w_{S_Q}} / \binom{m}{w_{S_Q}}$.

Proof: The average number of partitions searched is obtained in Theorem 2, and the number of partitions generated is:

$$\binom{k}{0} \chi(0,0) + \binom{k}{1} \chi(1,0) + ... + \binom{k}{k} \chi(k,0) = \sum_{i=0}^{k} \binom{k}{i} \chi(i,0)$$

The partition reduction ratio is:

$$\frac{\sum_{i=0}^{k} \binom{k}{i} P_i \chi(i,0)}{\sum_{i=0}^{k} \binom{k}{i} \chi(i,0)} = 2^{-k} \sum_{i=0}^{k} \binom{k}{i} P_i$$

$\square$

Note that χ can be canceled out, since, from lemma 1, χ is independent of i and has the same value for a fixed value of n and k. In the next theorem, χ is canceled out in the same manner.

Theorem 4 *The ratio of the number of signatures searched per query to the number of signatures in the file is* $\sum_{i=0}^{k} \binom{k}{i} P_i \cdot 2^{-k}$ *where* $P_i = \binom{m-k+i}{w_{S_Q}} / \binom{m}{w_{S_Q}}$.

Proof: The number of signatures searched is the average number of partitions searched, obtained in Theorem 2, times the number of signatures in a partition. Since there are n signatures and every partition has the same expected size, the number of signatures in a partition is n divided by the number of partitions generated, which has been obtained in Theorem 3. Therefore, the ratio is:

$$\frac{1}{n}\sum_{i=0}^{k}\binom{k}{i}P_i\cdot\chi(i,0)\cdot\frac{n}{\sum_{i=0}^{k}\binom{k}{i}\chi(i,0)}=\sum_{i=0}^{k}\binom{k}{i}P_i\cdot 2^{-k}$$

□

5.1.1 Performance and Discussion

As suggested by the analysis, the two reduction ratios are identical. Our simulation result agrees with this (see Figure 3). Data are obtained using signature lengths of 512, 1,024, 1,536, and 2,048. It is noticed that the performance improves as the key length increases, as the signature length decreases, and as the weight of the query signature increases. For clarity, we only show the data for $m = 512$ and $m = 2,048$ to depict the range of performance gain one can expect. The performance for $m = 1,024$ and $m = 1,536$ is within the bounds set by $m = 512$ and $m = 2,048$. For the set of data we used, the search space is reduced by 5% to 80%. This improvement is significant considering that little or no overhead is involved.[3] The graph also shows that the analytical result approximates the simulation result accurately.

For a sequential system, this method is very attractive since it offers significant advantage in speed with little overhead in the partitioning algorithm. The problem with this method occurs in a parallel environment. Theorem (1) shows that P_a is dependent on $\bar{w}_{K_{Mi}}$ and w_{S_Q}. Thus, some partitions are activated more frequently than others. Most importantly, P_a equals to 1 when $\bar{w}_{K_{Mi}}$ equals to zero. That is, $M_{11\ldots1}$ satisfies condition (3) regardless of the query and thus will be searched for every query. This would make $M_{11\ldots1}$ the bottleneck of the system. To alleviate the problem would require putting more buffer and/or using faster disks for that partition. One of the goals for the next two partitioning algorithms is to make accesses to the partitions as uniform as possible so that the workload is evenly distributed across the partitions.

Since the maximum number of partitions is 2^k, a key length of 10 to 15 bits seems to be a reasonable value. With the current technology in parallel systems (e.g., a connection machine can have 2^{16} processors[18]), the number of partitions generated by this method is considered manageable. Further, the expected size of each partition under our assumption is the same, making storage management more convenient.

[3] Storage is actually reduced since keys need not be stored explicitly.

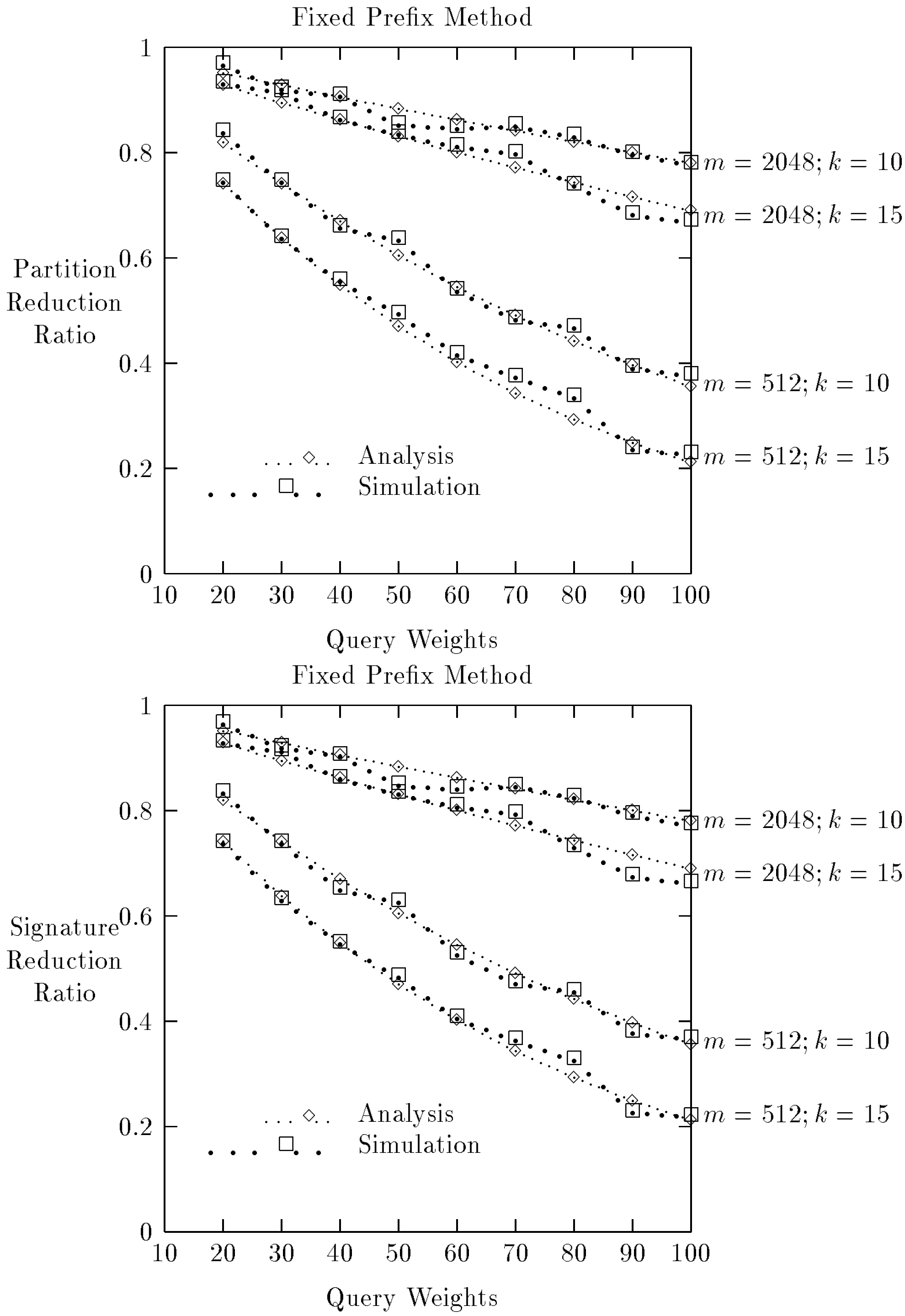


Figure 3: Performance of the fixed prefix method.

partition 1	$\underline{00}$ 0111
	$\underline{00}$ 1011
	$\underline{00}$ 1101
	$\underline{00}$ 1110
partition 2	$\underline{010}$ 011
	$\underline{010}$ 101
	$\underline{010}$ 110
partition 3	$\underline{0110}$ 01
	$\underline{0110}$ 10
partition 4	$\underline{01110}$ 0
partition 5	$\underline{100}$ 011
	$\underline{100}$ 101
	$\underline{100}$ 110
partition 6	$\underline{1010}$ 01
	$\underline{1010}$ 10
partition 7	$\underline{10110}$ 0
partition 8	$\underline{1100}$ 01
	$\underline{1100}$ 10
partition 9	$\underline{11010}$ 1
partition 10	$\underline{11100}$ 0

(a)

partition 1	$\underline{00}$ 0111
	$\underline{00}$ 1011
	$\underline{00}$ 1101
	$\underline{00}$ 1110
partition 2	01 $\underline{00}$ 11
	10 $\underline{00}$ 11
	11 $\underline{00}$ 01
	11 $\underline{00}$ 10
partition 3	0111 $\underline{00}$
	1011 $\underline{00}$
	1101 $\underline{00}$
	1110 $\underline{00}$
partition 4	$\underline{01}$ 0101
	$\underline{01}$ 0110
	$\underline{01}$ 1001
	$\underline{01}$ 1010
partition 5	$\underline{10}$ 0101
	$\underline{10}$ 0110
	$\underline{10}$ 1001
	$\underline{10}$ 1010

(b)

Figure 4: Distribution of signatures based on (a) Extended Prefix, and (b) Floating Key.

5.2 Extended Prefix Partitioning

In order to achieve a uniform workload distribution on the partitions, we can try to make the activation probability of each partition as close to each other as possible. In particular, to ensure no partition will be activated for every query, every key must contain at least one zero in it so that the probability that a partition is searched is sure to be less than 1.

In the extended prefix method, the key of a signature is chosen to be the shortest prefix which contains a pre-defined number, z, of zeros. Figure 4(a) is an example for partitioning 20 signatures with m equal 6 and z equal 2. The keys of the signatures are underlined. The keys are shown only for clarity; they need not be stored explicitly since they can be obtained from the partition keys. We assume that a signature must contain at least z zeros. As shown later, the value of z is very small to be practical. Therefore, the assumption would not introduce inconsistency when the algorithms are compared.

Lemma 2 $P(i,j) = P(i,0) = 2^{-k}$.

Proof: The proof is the same as Lemma 1. $\square$

Since $k = z + i$ in this method, both $P(i,j)$ and $\chi(i,j)$ are dependent on the weight of the key.

Theorem 5 *Activation probability: partitions are activated uniformly and the probability that a partition is activated is:*

$$P_a = \frac{\binom{m-z}{w_{S_Q}}}{\binom{m}{w_{S_Q}}}$$

Proof: Applying the argument in Theorem 1, we have

$$P_a = \frac{\binom{m-\bar{w}_{K_{Mi}}}{w_{S_Q}}}{\binom{m}{w_{S_Q}}} = \frac{\binom{m-(k-w_{K_{Mi}})}{w_{S_Q}}}{\binom{m}{w_{S_Q}}} = \frac{\binom{m-z}{w_{S_Q}}}{\binom{m}{w_{S_Q}}}$$

For a given query, P_a will be identical for all partitions since m and z are constant in extended prefix, and w_{S_Q} is constant. $\square$

Theorem 6 *Average size of a working set: The average number of partitions activated per query is $P_a \sum_{i=0}^{m-z} \binom{z+i-1}{i} \chi(i,0)$.*

Proof: Let's consider partitions with i ones in their keys. There are $\binom{z+i-1}{i}$ such keys, since the key length is $z+i$ (sum of zeros and ones) and the last bit of a key must be a zero (recall that the key always expands towards the right until it meets the z^{th} zero and that there are exactly z zeros in a partition's key). Therefore, all i ones appear in the first $z+i-1$ bit positions.

According to the last theorem, every partition has a equal probability of being activated (i.e., P_a constant). Therefore, the average number of activated partitions is $\sum_i \binom{z+i-1}{i} P_a \cdot \chi(i,0)$, where i ranges from 0 to $m-z$. $\square$

Theorem 7 *The ratio of the number of partitions searched to the number of partitions generated is P_a.*

Proof: The number of partitions generated is $\sum_i \binom{z+i-1}{i}\chi(i,0)$, where i, the number of ones in the key, ranges from 0 to $m-z$. The result can then be obtained from Theorem 6. □

Theorem 8 *The ratio of the number of signatures searched per query to the number of signatures in the signature file is P_a.*

Proof: The proof is similar to that of Theorem 4. However, since the expected size of the partitions is not uniform and not all possible partitions exists, it is more difficult to find the number of signatures in a partition when n signatures are partitioned. When all partitions do exist, the number of signatures hashed into a partition is $n \cdot 2^{-k}$. However, we know that only $\sum_{i=0}^{m-z} \binom{z+i-1}{i}\chi(i,0)$ partitions are generated, the number of signatures in a *generated* partition is obtained by dividing $n \cdot 2^{-k}$ with the ratio of the total capacity of the generated partitions and the capacity of all possible partitions:

$$\frac{n \cdot 2^{-k} \cdot 2^m}{\sum_{i=0}^{m-z} \binom{z+i-1}{i}\chi(i,0)2^{m-k}}$$

Therefore, the reduction ratio is:

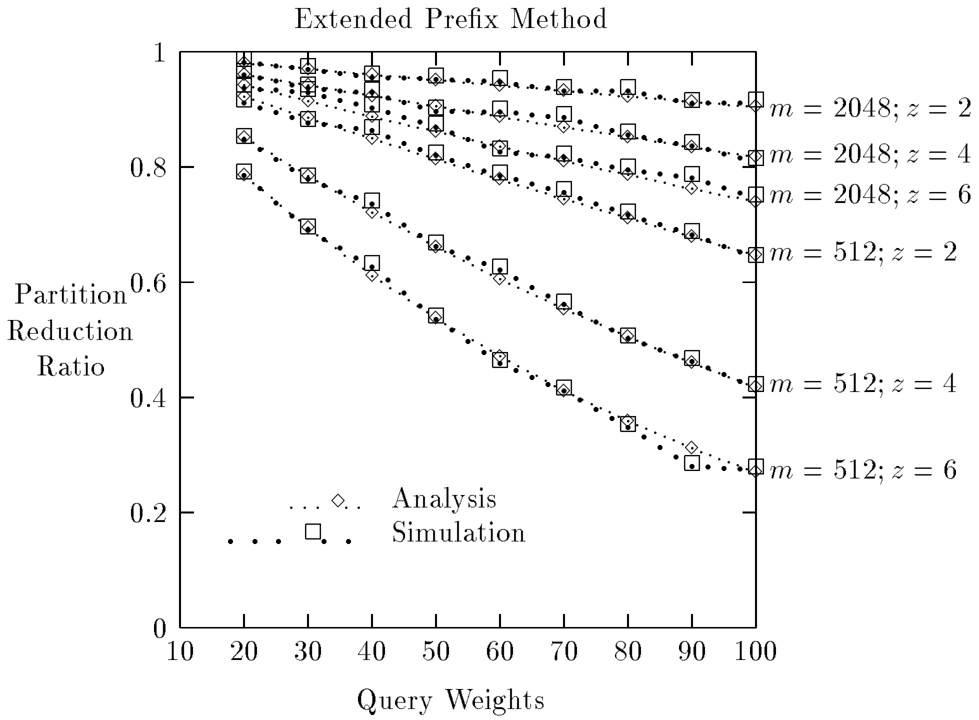
$$\frac{1}{n} P_a \sum_{i=0}^{m-z} \binom{z+i-1}{i}\chi(i,0) \frac{n \cdot 2^{-k} \cdot 2^m}{\sum_{i=0}^{m-z} \binom{z+i-1}{i}\chi(i,0)2^{m-k}} = P_a$$

□

5.2.1 Performance and Discussion

Figure 5 shows the performance of this method. As in the fixed prefix method, the two ratios are identical analytically as well as in the simulation. The analytical result approximates the simulation result very well. For clarity reasons as before, we only show data for $m = 512$ and $m = 2,048$. The graph shows a reduction of the search space which improves as m decreases, as z increases, and as the query weight increases. On the other hand, increasing z also increases the number of partitions generated. Therefore, z must be limited to a small value.

For small z, the performance of the extended prefix method is not as good as the first method. Another problem with this method is that the partition sizes are highly non-uniform. In particular, partitions with $w_{K_{Mi}} = w_{S_i}$ hold only one unique signature. This can be seen from the example in Figure 4(a).



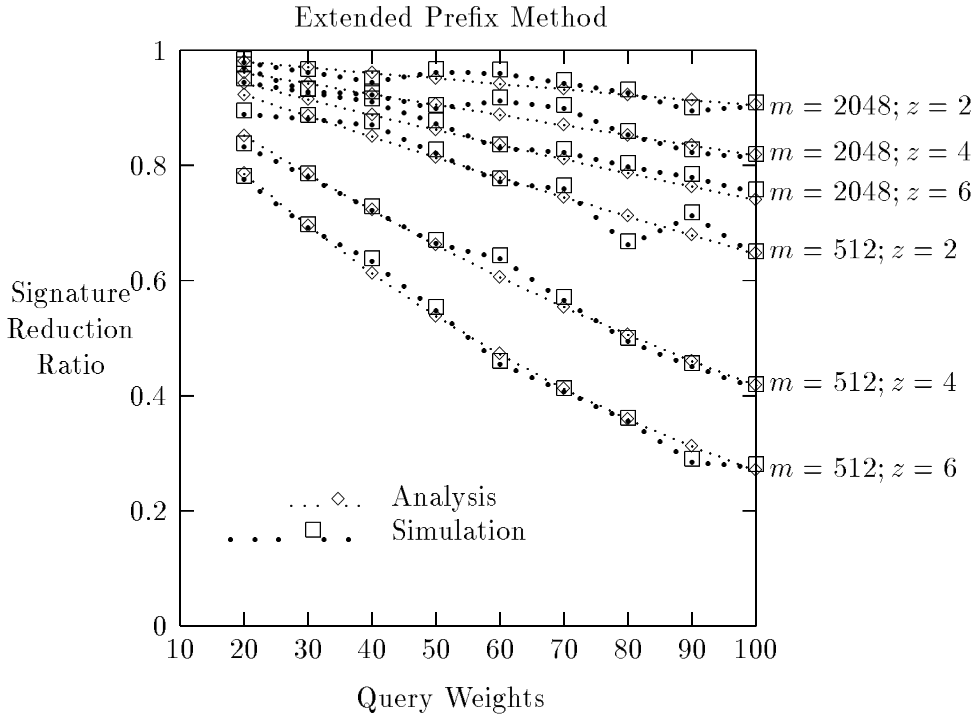


Figure 5: Performance of the extended prefix method.

The advantage we expect from this method is that P_a is uniform. However, a uniform P_a doesn't always mean a uniform distribution of workload if the workload of a partition is measured by the probability that a partition is searched *and* the number of signatures in the partition. Given the highly non-uniform distribution of signatures, the extended prefix method does not result in uniform distribution of workload in this sense, although it does avoid the situation in which a partition is searched always.

In a parallel implementation in which one partition is assigned to one processor (e.g., in a connection machine or hypercube), the extended prefix method may be more promising. A common processor allocation strategy in multicomputer systems is not to release any processor allocated to a query until the query is completed. In this situation, the workload of a processor is measured by the time it is occupied, independent of the number of signatures the processor holds. Since every partition (and therefore every processor) has the same probability of being included in a working set, the extended prefix method results in a more uniform distribution of workload.

5.3 Floating Key Partitioning

We can observe that if keys are selected only from the prefix, signatures with a sequence of ones followed by a sequence of zeroes may result in partitions with very long keys and small sizes, yet most of the bits in the key are ones which cannot be used to reduce the chance of that partition being searched.

In the floating key method, we avoid the problem by choosing a fixed key length, say, k and selecting the k-substring which contains the least number of ones. Specifically, each consecutive, non-overlapping, k-substrings of the signature is examined from left to right, and the leftmost substring with the smallest weight is chosen as the key. The key in this method composes of the *key value* ($K^V_{S_i}$, the substring which forms the key), and the *key position* ($K^P_{S_i}$, the starting bit position of the key in the signature). Figure 4(b) is an example illustrating the floating key method. As with the extended prefix method, the key of each signature (underlined) needs not be stored explicitly in the partition. If we know the key of the partition (key value as well as starting position), the signatures in that partition can always be reconstructed.

This algorithm attempts to maximize the number of zeros in a key. Since the probability that a partition is activated by a query decreases as the number of zeros in the partition key increases, this algorithm tends to minimize the size of the working set of a query.

$P(i,j)$ in this method is much more complex. It depends on both i and j.

Lemma 3 $P(i,j) = \left(2^k - \sum_{t=0}^{i} \binom{k}{t}\right)^j \left(2^k - \sum_{t=0}^{i-1} \binom{k}{t}\right)^{\lfloor \frac{m}{k} \rfloor - j - 1} 2^{-\lfloor \frac{m}{k} \rfloor k}$

Proof: We first show that the partition size decreases as the number of zeros in the key decreases and the value of $K^P_{S_i}$ increases.

$K^P_{S_i}$ is equal to 1, k+1, 2k+1, ..., $k\left(\lfloor\frac{m}{k}\rfloor - 1\right)+1$. Without loss of generality, we consider the case that the key contains all zeros. Each k-bit segment to the left of the key must each contain at least one non-zero bit, leading to $2^k - 1$ possibilities for each segment. The segments to the right of the key are arbitrary. As shown in the following table, the partition size decreases as the value of $K^P_{S_i}$ increases.

$K^P_{S_i}$	Partition Size
1	$(2^k)^{\lfloor\frac{m}{k}\rfloor-1}\cdot 2^{m-\lfloor\frac{m}{k}\rfloor k} = 2^{m-k}$
$k+1$	$(2^k-1)(2^k)^{\lfloor\frac{m}{k}\rfloor-2}\cdot 2^{m-\lfloor\frac{m}{k}\rfloor k} = (2^k-1)2^{m-2k}$
...	...
$i\cdot k+1$	$(2^k-1)^i(2^k)^{\lfloor\frac{m}{k}\rfloor-i-1}\cdot 2^{m-\lfloor\frac{m}{k}\rfloor k} = (2^k-1)^i 2^{m-(i+1)k}$
...	...
$(\lfloor\frac{m}{k}\rfloor - 1)\,k+1$	$(2^k-1)^{\lfloor\frac{m}{k}\rfloor-1}\cdot 2^{m-\lfloor\frac{m}{k}\rfloor k}$

Now, without loss of generality, consider the case when $K^P_{S_i}$ is fixed at $k+1$ and when the number of zeros in the key decreases. Each k-bit segment to the left of $K^P_{S_i}$ must be strictly heavier than the key segment and each k-bit segment to the right of the key either is heavier than or have the same weight as the key. Therefore, we can obtain the following relationships:

$\bar{w}_{K_{Mi}}$	Partition Size
k	$(2^k-1)(2^k)^{\lfloor\frac{m}{k}\rfloor-2}\cdot 2^{m-\lfloor\frac{m}{k}\rfloor k} = (2^k-1)\cdot 2^{m-2k}$
$k-1$	$\left[2^k-\binom{k}{1}-1\right](2^k-1)^{\lfloor\frac{m}{k}\rfloor-2}\cdot 2^{m-\lfloor\frac{m}{k}\rfloor k}$
...	...
$k-i$	$\left[2^k-\sum_{j=1}^{i}\binom{k}{j}-1\right]\left[2^k-\sum_{j=1}^{i-1}\binom{k}{j}-1\right]^{\lfloor\frac{m}{k}\rfloor-2}\cdot 2^{m-\lfloor\frac{m}{k}\rfloor k}$
...	...
1	$\left[2^k-\sum_{j=1}^{k-1}\binom{k}{j}-1\right]\left[2^k-\sum_{j=1}^{k-2}\binom{k}{j}-1\right]^{\lfloor\frac{m}{k}\rfloor-2}\cdot 2^{m-\lfloor\frac{m}{k}\rfloor k}$

When the key weight is k (all-one key), $K^P_{S_i}$ must be 1 and the partition size is also 1, with key 111...1. It can be seen that the partition size is decreasing as the number of zeros decreases.

The maximum size of partitions with keys containing i ones and $K^P_{S_i} = jk+1$ can be obtained by generalizing the above arguments:

$$\left(2^k - \sum_{t=0}^{i}\binom{k}{t}\right)^j\left(2^k - \sum_{t=0}^{i-1}\binom{k}{t}\right)^{\lfloor\frac{m}{k}\rfloor-j-1} 2^{m-\lfloor\frac{m}{k}\rfloor k}$$

and the lemma is proved by dividing the expression with 2^m. $\square$

We can see that partitions with $K^P_{S_i} = 1$ and $K^V_S = 0$ are the largest with an upper bound equal to 2^{m-k}.

Theorem 9 *Activation probability: the probability that a partition is searched is:*

$$P_a = \frac{\binom{m - \bar{w}_{K_{Mi}}}{w_{S_Q}}}{\binom{m}{w_{S_Q}}}$$

Proof: The proof is the same as that of Theorem 1. $\square$

When proving the following theorem, it is important to note that the value of P_a in floating key only depends on the number of zeros in the key but does not depends on the starting position, $K^P_{S_i}$.

Theorem 10 *Average size of a working set: The average number of partitions activated per query is:*

$$\sum_{i=0}^{k-1} P_i \left(\sum_{j=0}^{\lfloor \frac{m}{k} \rfloor - 1} \binom{k}{i} \chi(i,j) \right) + \chi(k,0)$$

where $P_i = \binom{m-k+i}{w_{S_Q}} / \binom{m}{w_{S_Q}}$.

Proof: We define the variable x_i to be the number of partitions which contain i ones in the key. From the last theorem, the only factor affecting P_a is the number of ones in the key. Therefore, the number of all possible partitions containing i ones in the key equals to the number of of possible key positions times the number of permutations of i ones in a key of length k:

$$\left\lfloor \frac{m}{k} \right\rfloor \binom{k}{i} \quad \text{for} \quad i = 0, 1, ..., k-1$$

However, when we consider the partitions generated by a given number of signatures n, each possible partition must be multiplied by $\chi(i,j)$ to get the number of partitions generated:

$$\sum_{j=0}^{\lfloor \frac{m}{k} \rfloor - 1} \binom{k}{i} \chi(i,j)$$

The second term in the theorem accounts for the special case when the key contains all ones, which is possible only when the whole signature has no zero in it. In this case, j is always zero and P_k is 1. The chance for this to occur is extremely small though, and this term could be dropped without any significant effect on the result. □

Theorem 11 *The ratio of the number of partitions searched to the number of partitions generated is:*

$$\frac{\sum_{j=0}^{\lfloor \frac{m}{k} \rfloor -1} \sum_{i=0}^{k-1} \binom{k}{i} P_i \chi(i,j)}{\sum_{j=0}^{\lfloor \frac{m}{k} \rfloor -1} \sum_{i=0}^{k-1} \binom{k}{i} \chi(i,j)}$$

Proof: The numerator is obtained from Theorem (10) with the second term dropped because it is negligible. The denominator, which is the number of partitions generated, is obtained in a similar way. □

Theorem 12 *The ratio of the number of signatures to be searched per query to the total number of signatures is:*

$$\frac{\sum_{i=0}^{k-1} P_i \binom{k}{i} \sum_{j=0}^{\lfloor \frac{m}{k} \rfloor -1} \left(2^k - \sum_{t=0}^{i} \binom{k}{t}\right)^j \left(2^k - \sum_{t=0}^{i-1} \binom{k}{t}\right)^{\lfloor \frac{m}{k} \rfloor - j - 1} 2^{m-k\lfloor \frac{m}{k} \rfloor} \chi(i,j)}{\sum_{i=0}^{k-1} \binom{k}{i} \sum_{j=0}^{\lfloor \frac{m}{k} \rfloor -1} \left(2^k - \sum_{t=0}^{i} \binom{k}{t}\right)^j \left(2^k - \sum_{t=0}^{i-1} \binom{k}{t}\right)^{\lfloor \frac{m}{k} \rfloor - j - 1} 2^{m-\lfloor \frac{m}{k} \rfloor k} \chi(i,j)}$$

where $P_i = \binom{m-k+i}{w_{S_Q}} / \binom{m}{w_{S_Q}}$.

Proof: From Lemma 3, the capacity of partitions with keys containing i ones and $K_{S_i}^P = jk+1$ is:

$$\left(2^k - \sum_{t=0}^{i} \binom{k}{t}\right)^j \left(2^k - \sum_{t=0}^{i-1} \binom{k}{t}\right)^{\lfloor \frac{m}{k} \rfloor - j - 1} 2^{m-\lfloor \frac{m}{k} \rfloor k}$$

When there is only n signatures, the expected size of a partition is $n/2^m$ of its capacity. To account for partitions which are not generated at all, χ is added and the size is further corrected by the capacity of the generated partitions and the total capacity (2^m). Therefore, the number of signatures in a partition with key containing i ones and $K_{S_i}^P = jk+1$ is:

$$\frac{\left(2^k - \sum_{t=0}^{i} \binom{k}{t}\right)^j \left(2^k - \sum_{t=0}^{i-1} \binom{k}{t}\right)^{\lfloor \frac{m}{k} \rfloor - j - 1} 2^{m-\lfloor \frac{m}{k} \rfloor k} \cdot \chi(i,j) \frac{n}{2^m} 2^m}{\sum_{i=0}^{k-1} \binom{k}{i} \sum_{j=0}^{\lfloor \frac{m}{k} \rfloor -1} \left(2^k - \sum_{t=0}^{i} \binom{k}{t}\right)^j \left(2^k - \sum_{t=0}^{i-1} \binom{k}{t}\right)^{\lfloor \frac{m}{k} \rfloor - j - 1} 2^{m-\lfloor \frac{m}{k} \rfloor k} \chi(i,j)}$$

The theorem is obtained by summing the above expression over all i and j, multiplying each term with P_i and the number partitions with key weight i and starting position j, and finally dividing the result by n to obtain the ratio. As in Theorem 10, the case when the key is 11...11 is ignored. □

5.3.1 Performance and Discussion

Unlike the previous two methods, both the analytical and simulation result show that the two ratios are not the same (see Figure 6). Comparing the graphs reviews that this method is superior than the first two methods in both signature and partition reduction ratios. The result is not surprising considering that the number of zeros contained in the keys are maximized and that the average partition size is small in this method. The simulation and analytical results for the two ratios are very close each other.

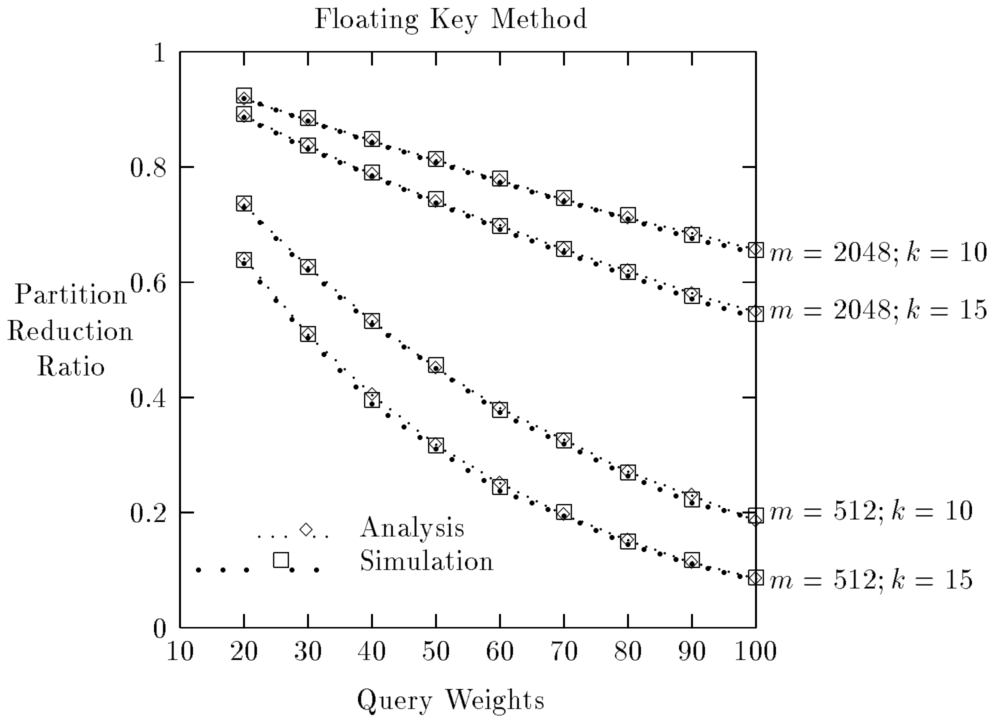
Unlike the extended prefix method, the partition activation probability in this method is non-uniform. In fact, the activation probability of this method has an identical expression as that of the fixed prefix method, but this method has a much better uniformity in partition activation than the fixed prefix method. This is because when the signature weight is controlled to $m/2$, as in the real case, the weight of the key varies only from 0 to $k/2$ (versus 0 to k in the fixed prefix method). Even if the signature weight is not controlled, a key of all ones is generated only when a signature of all ones is generated; the probability for this to happen is practically zero.

We conclude that, of the three methods we study, the performance of this method is the best while maintaining a balance between the uniformity of partition size, uniformity of partition activation, and the number of partitions generated. A minor drawback of this method is that the algorithm for obtaining the key from a signature and for comparing between keys is more complex, but this should not be a serious problem as computation time is negligible compared to I/O costs.

6 Discussion and Conclusion

We discuss the advantages of partitioning signatures into a number of partitions by a deterministic mapping algorithm. With very little overhead, this method reduces the number of partitions searched for a query. We discuss issues concerning the designs of mapping algorithms and their properties. The performance of the algorithms is studied both analytically and with simulation.

Three partitioning algorithms, namely, the fixed prefix, extended prefix, and floating key methods, are presented, and their performance evaluated. in terms of the partition reduction ratio and signature reduction ratio. The fixed prefix method is the simpliest, and performs well in a sequential environment in which one partition is searched at a time.



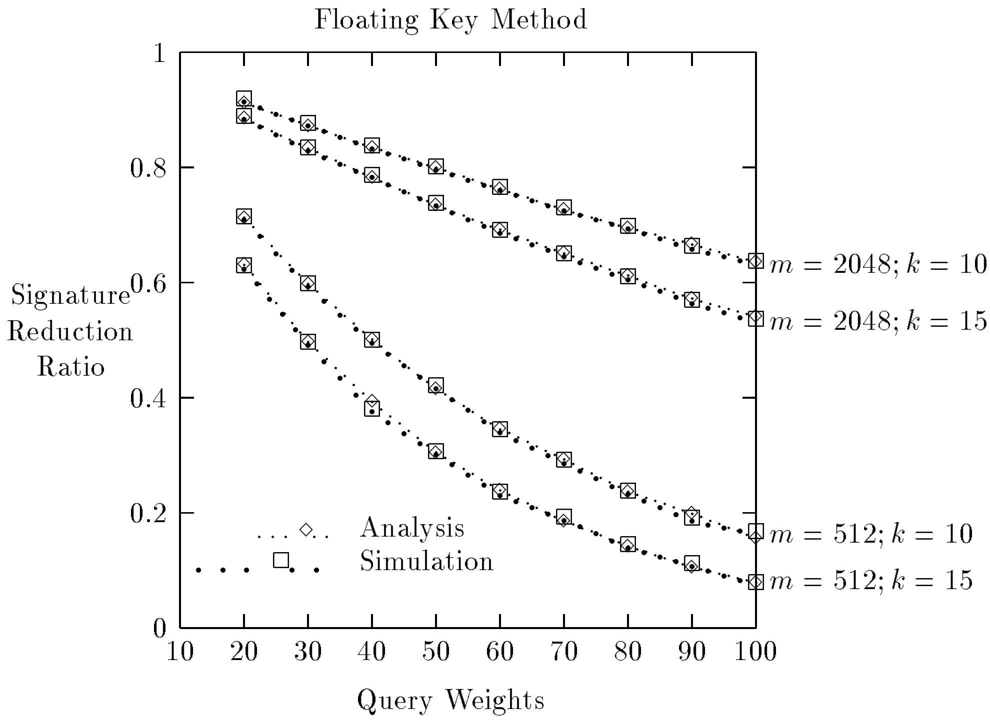


Figure 6: Performance of the floating key method.

It is less suitable for a parallel implementation in which multiple partitions are search simultaneously, since the probability that a partition is searched is highly non-uniform.

The extended prefix method does not perform as well as the fixed prefix method, in addition to problems such as highly skewed partition size and a relatively large number of partitions generated. However, every partition has about the same probability of being searched by a query. This property may be useful in parallel systems in which each partition is searched with a processor, because the workload, under some common processor allocation strategy, is distributed more evenly over the processors.

The floating key method is the most complex one, and so is its analysis. Its performance is the best in terms of the reduction of the search space. Its other characteristics are favorable too: its partition sizes do not vary as much as the extended prefix method; its activation probability is non-uniform, but not as skewed as the fixed prefix method. Therefore, it should perform reasonably well in a parallel environment.

We conclude that the fixed prefix method is the simplest one and incur the least overhead. The floating key method is most attractive in both a sequential or a parallel environment and worth further study. Of particular interest is the allocation of i partitions onto j processors, where $i > j$, so that a uniform workload can be achieved. If this can be done, we can achieve the advantages of both the extended prefix method and floating key method. Our study also reveals many interesting properties of the algorithms which are difficult to observe by studying the algorithms qualitatively. For instance, the two reduction ratios are identical and independent of n in the first two methods.

Acknowledgment

The authors wish to thank J. Cunnyngham, M. McDonald, M. McGill, P.W.Y. Wong, and C. Zeidner for their helpful discussion in this project. The anonymous referees also made suggestions which greatly improved the contents and presentation of this paper.

REFERENCES

[1] S.R. Ahuja, and C.S. Roberts, "An associative/parallel processor for partial match retrieval using superimposed codes," *Proc. 7th Annual Symp. Comput. Arch.*, France, May 1980, 218–227.

[2] B. Berra, S.M. Chung, and N.I. Hachem, "Computer architecture for a surrogate file to a very large data/knowledge base," *IEEE Computer*, Vol. 20, No. 3, Mar. 1987, 25–32.

[3] S. Christodoulakis, and C. Faloutsos, "Design considerations for a message file server," *IEEE Trans. Soft. Eng.*, Vol. SE-10, No. 2, Mar. 1984, 201–210.

[4] U. Deppisch, "S-Tree: A dynamic balanced signature index for office retrieval," *Proceedings of the ACM Conference on Research and Development in Information Retrieval*, Pisa, Italy, Sept. 1986, 77–87.

[5] C. Faloutsos, "Access methods for text," *ACM Computing Surveys*, Vol. 17, No. 1, Mar. 1985, 49–74.

[6] C. Faloutsos, and S. Christodoulakis, "Description and performance analysis of signature file methods for office filing," *ACM Transactions on Office Information Systems*, Vol. 5, No. 3, July 1987, 237–257.

[7] C. Faloutsos, and S. Christodoulakis, "Optimal signature extraction and information loss," *ACM Transactions on Database Systems*, Vol. 12, No. 3, Sept. 1987, 395–428.

[8] R.A. Gustafson, "Elements of the randomized combinatorial file structure," *Proc. Symp. on Information Storage and Retrieval*, College Park, MD, Apr. 1971, 163–174.

[9] D.L. Lee, "A word-parallel, bit-serial signature processor for superimposed coding," *Proceedings of the Second International Conference on Data Engineering*, Los Angeles, Feb. 1986, 352–359.

[10] D.L. Lee, and C.-W. Leng, "A partitioned signature file structure for multiattribute and text retrieval," *Proceedings of the Sixth International Conference on Data Engineering*, Los Angeles, Feb. 1990, 389–397.

[11] D.L. Lee, and F.H. Lochovsky, "Text retrieval machines," in *Office Automation*, D.C. Tsichritzis, ed., Springer-Verlag, New York, N.Y., 1985, 339–375.

[12] G. Orosz, and L. Tackacs, "Some probability problems concerning the marking of codes in the superimposition field," *J. Documentation*, Vol. 12, No. 4, Dec. 1956, 231–234.

[13] J.L. Pfaltz, W.J. Berman, and E.M. Cagley, "Partial match retrieval using indexed descriptor files," *Communications of ACM*, Vol. 23, No. 9, Sept. 1980, 522–528.

[14] R.L. Rivest, "Partial-match retrieval algorithms," *SIAM J. Computing*, Vol. 5, No. 1, 1976, 19–50.

[15] C.S. Roberts, "Partial-match retrieval via method of superimposed codes," *Proc. IEEE*, Vol. 67, No. 12, Dec. 1979, 1624–1642.

[16] R. Sacks-Davis, A. Kent, and K. Ramamohanarao, "Multikey access methods based on superimposed coding techniques," *ACM Transactions on Database Systems*, Vol. 12, No. 4, Dec. 1987, 655–696.

[17] 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.

[18] C. Stanfill, and B. Kahle, "Parallel free-text search on the connection machine system," *Communications of ACM*, Vol. 29, No. 12, Dec. 1986, 1229–1239.

[19] D. Tsichritzis, and S. Christodulakis, "Message files," *ACM Trans. Office Info. Syst.*, Vol. 1, No. 1, January 1983, 88–98.

[20] Lewis W. Tucker, and George G. Robertson, "Architecture and Applications of the Connection Machine," *IEEE Computer*, Aug. 1988, 26–38.