Design of a Signature File Method that Accounts for Non-Uniform Occurrence and Query Frequencies.

Christos Faloutsos
Stavros Christodoulakis
Computer Systems Research Institute
Univ. of Toronto

Abstract. In this paper we study a variation of the signature file access method for text and attribute retrieval. According to this method, the documents (or records) are stored sequentially in the "text file". Abstractions ("signatures") of the documents (or records) are stored in the "signature file". The latter serves as a filter on retrieval: It helps discarding a large number of non-qualifying documents. We propose a signature extraction method that takes into account the query and occurrence frequencies, thus achieving better performance. The model we present is general enough, so that results can be applied not only for text retrieval but also for files with formatted data.

1. Introduction.

Traditional data base management systems (DBMSs) are designed for formatted records. Recently there are many attempts to extend these systems so that they will be able to handle unformatted, free text (Dattola 1979 [8]), (McLeod 1981 [16]), (Haskin & Lorie 1982 [11]), (Tsichritzis & Christodoulakis 1983 [22]), (Christodoulakis & Faloutsos 1984 [5]). The major application of such extended systems is office automation. Many types of messages circulate in an office: correspondence, memos, reports etc. These messages consist not only of attributes (e.g., sender, date etc.) but also of text. In an automated office, these messages should be stored and retrieved automatically.

Another important application of a text retrieval method is the computerized library. The problem of handling queries on the contents has attracted much research interest in the past few decades (Salton & McGill 1963 [21]), (Van Rijsbergen 1979 [24]). As result of the above research activity many text retrieval methods have been proposed in the literature.

2. Overview of text retrieval methods.

The text retrieval methods that have appeared in the literature seem to form the following classes:

1) Full text scanning. Given a search pattern, the whole data base is scanned until the qualifying documents are discovered and returned to the user. The method requires no space overhead and minimal effort on insertions and updates. The main disadvantage is the retrieval speed. Despite the existence of some fast string searching algorithms (Aho & Corasick 1975 [1]), (Knuth, Morris & Pratt 1977 [15]), (Boyer & Moore 1977 [2]), scanning of a large data base may take too much time (Hollaar 1979 [12]).

2) Inversion. This method uses an index. An entry of this index consists of a word (or stem or concept) along with a list of pointers. These pointers point to documents that contain this word. Many commercial systems have adopted this approach: STAIRS (IBM 1979 [14]), MEDLARS, ORBIT, LEXIS (Salton & McGill 1983 [21]) etc. The main advantage of the method seems to be its retrieval speed. However, it may require large storage overhead for the index: 50%-300% of the initial file size, according to Haskin (1981) [10]). Moreover, insertions of new documents require expensive updates of the index.

3) Signature files. The documents are stored sequentially in the "text file". Their abstractions are stored sequentially in the "signature file". When a query arrives, the signature file is scanned sequentially and most of the non-qualifying documents are discarded. The rest are either checked (so that the "false drops" are...
4) Clustering. In this method, similar documents are grouped together to form clusters. Usually, they are stored physically together, too. Clustering is the dominating access method in library science (Salton 1971 [20]), (Van Rijsbergen 1979 [24]), (Salton & McGill 1963 [21]). However, it seems that clustering cannot handle insertions easily: Van Rijsbergen (1979) [24, pp. 59-59] observes that "sound" clustering methods usually need O(n) time to perform an insertion, while "iterative" clustering methods may need reorganization, which takes O(n log n) time. Moreover, the method allows false dismissals, that is, it may fail to retrieve some documents, even though they qualify ("reCALL" < 1).

5) Multiattribute hashing. Gustafson [9] proposed a multiattribute hashing scheme based on superimposed coding in 1969. His method is applicable on bibliographic data bases. A constant number of keywords of a title are hashed and yield a signature for the title. A sophisticated one-to-one function transforms this signature into an address of the hash table. Notice the difference between this method and signature files: In the latter, the signature is stored at the end of the signature file; during the query resolution, the signature file is scanned sequentially. The interesting property of multiattribute hashing is that the number of buckets to be searched decreases exponentially with the number of search terms in the (conjunctive) query. However, no commercial system has applied this method, to the best of our knowledge.

One of the main characteristics of the office environment is the large insertion rate (Hollaar et al. 1983 [13]). Moreover, many documents are never retrieved after they have been filed. Gravina [a] reports that "sound" clustering methods usually need O(n) time to perform an insertion, while "iterative" clustering methods may need reorganization, which takes O(n log n) time. Moreover, the method allows false dismissals, that is, it may fail to retrieve some documents, even though they qualify ("reCALL" < 1).

A logical block is defined as a piece of text that contains a constant number D of distinct, non-common words. The reason we have to do that is that performance deteriorates fast if there is a large variance on the number of words in a block. Each distinct, non-common word yields a bit pattern of size $F$. These bit patterns are OR-ed together to form the block signature. The concatenation of the block signatures of a document form the document signature. The word signature creation is rather sophisticated and needs more details: Each word yields m bit positions (not necessarily distinct) in the range 1-$F$. The corresponding bits are set to "1", while all the other bits are set to "0". For example, in Figure 1, the word "free" sets to "1" the 3-rd, 7-th, 0-th, and 11-th bits (m=4 bits per word). In order to facilitate searching for parts of words, each word is divided in successive overlapping triplets. Each such triplet yields one bit position that is set to "1".

<table>
<thead>
<tr>
<th>Word</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>free</td>
<td>001 000 110 010</td>
</tr>
<tr>
<td>text</td>
<td>000 010 101 001</td>
</tr>
<tr>
<td>block signature</td>
<td>001 010 111 011</td>
</tr>
</tbody>
</table>

Figure 1
Illustration of the superimposed coding method.
It is assumed that each logical block consists of $D=2$ words only.

The signature size $F$ is 12 bits. $m=4$ bits per word.

Searching for a word is handled as follows: The signature of the word is created. Suppose that the signature contains "1" in positions 2, 3, 6, and 9. Each block signature is examined. If the above bit positions (i.e., 2, 3, 6, and 9) of the block signature contain "1", then the block is retrieved. Otherwise, it is discarded. More complicated Boolean queries can be handled easily. In fact, conjunctive (AND) queries result in a smaller number of false drops. Even sequencing of words can be handled: It is replaced with conjunction (at the expense of increasing the number of false drops).

3. Problem formulation.
In previous analyses [19,5] it was assumed that the probability that any word appears in users' query is uniform (uniform query frequency assumption), as well as that the words appear with equal frequency in the text (uniform occurrence frequency assumption).

Both these assumptions are unrealistic. It is well known that the frequency of word occurrence in large texts follows Zipf's law [25]. For example, if a database consists of e.g., technical reports about computers, then words such as 'data', 'program', 'algorithm' etc. will have rather high occurrence frequency, while words that do not belong to the computer science terminology will have lower occurrence frequency. On the other hand, the 'frequently' occurring words will probably not be used in queries often because they will retrieve too many documents.

We can improve the performance of the described method by allowing special treatment to words with high discriminatory power (high query frequency and low occurrence frequency): Each such word will be hashed to m1 bit positions within the block signature, discarded) or they are returned to the user as true drops.
while the rest of the words will be hashed to \( m_2 < m_1 \) bit positions. Next we try to derive a formula that gives the optimum values of \( m_1, m_2 \).

We can partition the set \( S \) of all possible words of the text-file into two sets \( S_1 \) and \( S_2 \) such that \( S_1 \cup S_2 = S \) and \( S_1 \cap S_2 = \emptyset \). \( S_1 \) will contain the discriminatory words and \( S_2 \) the rest. We have to define a measure for the occurrence and query frequency. Let:

\[
q_1 = \text{Prob}\{ \text{a single word query is about a word in } S_1 \}
\]

and

\[
q_2 = 1 - q_1 = \text{Prob}\{ \text{a single word query is about a word in } S_2 \}.
\]

Let \( D_1 \) be the average number of (distinct) words of the set \( S_1 \) in a block and similarly \( D_2 \) for \( S_2 \). Obviously, \( D_1 + D_2 = D \): expected number of (distinct) words in a block.

As measure of the performance we choose the false drop probability \( F_d \). This is the probability that a signature will qualify in a query, while the block does not actually qualify. Formally:

\[
F_d = \text{Prob}\{ \text{a signature qualifies / the block does not} \}
\]

To make the analysis tractable, we assume single word queries only. Note that we may partition \( S \) into more than 2 subsets without significant increase in complexity. Thus, assume that \( S \) is partitioned into \( n \) subsets \( S_1, S_2, \ldots, S_n \), which are disjoint and whose union is \( S \). The \( q_i \)'s and \( D_i \)'s are defined in the obvious way. The restrictions \( \sum_i q_i = 1 \) and \( \sum_i D_i = D \) still hold. The definitions are summarized in Table I. Now we are ready to define the problem.

Given the query frequencies \( q_1, q_2, \ldots, q_n \), the expected number of distinct words per block \( D_1, D_2, \ldots, D_n \) for each subset \( S_i \) and the signature size \( F \), find the bits set to "1" per word \((m_1, m_2, \ldots, m_n)\) for each subset \( S_i \), such that the false drop probability \( F_d \) is minimized:

\[
F_d = q_1 w^{m_1} + q_2 w^{m_2} + \ldots + q_n w^{m_n}
\]

with

\[
w = \frac{M}{F}
\]

the ratio of "1"'s in a block signature.

The details of the analysis are presented in the Appendix. The results are repeated here (Eq. (A.4), (A.5), (A.9), (A.10)):

\[
\frac{q_1 w^{m_1}}{D_1} = \frac{q_2 w^{m_2}}{D_2} = \ldots = \frac{q_n w^{m_n}}{D_n}
\]

\[
w = \frac{1}{2}
\]

\[
m_i = \frac{F \ln 2}{D} \left[ \ln \frac{q_i}{D_i} - \sum_{i=1}^{n} \frac{D_k \ln \frac{q_k}{D_k}}{D_k} \right]
\]


1) An interesting observation is that the optimum ratio \( w \) of "1"'s in the block signature is still 0.5 (Eq. (4)). This value is optimum for many variations of superimposed coding, e.g., [Stiassny '80], [Mooers, '59], [Roberts, '79]. This value corresponds to maximum entropy in the block signature, from an information-theory point of view.

2) Eq. (3) confirms the intuition about the dependency of the optimal \( m_i \) on the query and occurrence frequency \( q_i \) and \( D_i \). Since \( 0 < w < 1 \), large query frequency implies small \( m_i \) while large occurrence frequency implies large \( m_i \).

3) If

\[
\frac{q_1}{D_1} = \frac{q_2}{D_2} = \ldots = \frac{q_n}{D_n}
\]

then all the subsets \( S_i \) enjoy similar treatment (\( m_1 = m_2 = \ldots = m_n \)). This means that if the occurrence frequency of each set is proportional to each query frequency, then the best we can do is to use the same number of bits \( m_i \) for every set.

4) It is easy to show that both Eq. (5) and (6) reduce to the corresponding formulas if there is no partitioning (\( n = 1 \)). (see, e.g., [5]).

5) Knowledge of the \( q_i \)'s and the \( D_i \)'s and use of Eq. (5) may be useful. For illustration, assume that the 80-20 rule holds: 20% of the vocabulary in the text file receives the 80% of the users' interest (queries). In that case, we have two sets of words \( S_1 \) and \( S_2 \) with \( q_1 = .8, q_2 = .2, D_1 = 8, D_2 = 32 \) (\( D = 40 \) distinct words per block). In order to compare the two methods we allow the same signature size \( F \) and compare the false drop probability \( F_d \) that each method can achieve. From Eq. (6) we see that the relative savings \( s \) in false drops are given by the formula

\[
s = 1 - \left( \frac{1 - 2q_1}{1 - 2q_2} \right)^{q_1}
\]
For \( q_1 = 0.8 \), the savings are 
\[ s = 56.47 \text{ per cent} \]
while for \( q_1 = 0.9 \) (90-10 rule) the savings are 
\[ s = 62.75 \text{ per cent} \]

7) Another strong point of our model is that it is
general and several real situations can be considered
as more specific cases of the general case. For exam-
ple, if we do not distinguish between words at all, we
can use this method with \( m = 1 \) and \( q_1 = 1 \). The method
can also handle even the extreme case, where each
subset \( S_i \) consists of one only word. The "extremity"
of this case lies in the fact that rarely is it cost
effective to measure the query and occurrence fre-
quency of each individual word.

5. Application in formatted data bases.

The proposed model can be easily applied to for-
matted records, with or without repeating groups. An
example best illustrates how this can be done: Con-
sider telephone directory records with the following
attributes:
A1: NAME, consisting of 3 character strings on the
average ("repeating group").
A2: ADDRESS, consisting of 5 alphanumeric strings on
the average ("repeating group").
A3: TELEPHONE NUMBER, a 7-digit integer.

An example data base is depicted in Figure 2.

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
<th>TEL. #</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td>125 Queen Victoria St. 999.9999</td>
<td></td>
</tr>
<tr>
<td>Mike Queen</td>
<td>17 Nelson Rd.           777.7777</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.
Example data base.

Roberts [19] suggested that, e g , the strings "Queen"
as a last name and "Queen" as a street name yield
different signatures. He also suggested that, in gen-
eral, all strings enjoy the same \( m \) in their signatures,
with the exception perhaps of frequently occurring
strings: In that case he suggested a rule of thumb in
order to find a (smaller) \( m \) to assign to these frequent
strings.

Our model gives exact solution to the above prob-
tem, taking not only the occurrence but also the query
frequency into account. In the example data base of
Figure 2, the set of frequently occurring words could
be \( S_1 = \{ \text{John, Smith, Mike} \} \), while the set of non-
frequent words would be
\[ S_2 = \{ \text{Queen, 125, Victoria, St, 17, Nelson, Rd} \} \]

Another use of the developed model is to provide
help in assigning different \( m \)'s to each of the attrib-
utes. For example, if most of the queries are on
"NAME" (but we are not willing to maintain statistics
for each individual name), it would be reasonable to
assign larger \( m \) to all the strings (values) of this attrib-
ute. In this case, the sets of our model would be:
\[ S'_1 = \{ \text{John, Smith, Mike, Queen} \} \]
\[ S'_2 = \{ 125, Queen, Victoria, St, 17, Nelson, Rd \} \]
\[ S'_3 = \{ 999.9999, 777.7777 \} \]

The generality of the proposed model allows even
more complicated situations. In the previous setting,
it may be found that the name "Smith" is very com-
mon and deserves special treatment. Under this con-
dition, the sets will become:
\[ S''_1 = \{ \text{Smith} \} \]
\[ S''_2 = \{ \text{John, Queen, Mike} \} \]
\[ S''_3 = S''_2 = \{ 125, Queen, Victoria, St, 17, Nelson, Rd \} \]
\[ S''_4 = S''_3 = \{ 999.9999, 777.7777 \} \]

6. Applications in text data bases.

The proposed method can be easily applied in a
textual data base, using two sets \( S_1 \) and \( S_2 \). \( S_1 \)
contains the discriminatory words while \( S_2 \) contains
the rest and need not be stored. One might argue that it
is not practical to use the proposed approach with
more than two sets \( S_1 \) and \( S_2 \). The problems with
many sets are:
1) We need look-up tables, in order to record which
word belongs to which set. This might consume
too much space.
2) Insertion of a new document may introduce some
new words and therefore rewriting of the look-up
tables may be necessary. Thus, insertions are
not handled easily any more.

The remedy we propose to these problems is to use
triplets of letters as hashing elements, instead of
words. We have already seen that a hashing method
based on triplets is useful in our environment [5]. Our
model and all the derived formulas can be used
without significant changes: A "logical block" should
contain a constant number of triplets now, \( U_i \) is the
number of times that the \( i \)-th triplet appears in a
block, \( q_i \) is the probability that a single triplet query
refers to the \( i \)-th triplet. \( F_{d,triplet} \) or better \( F_{d,triplet,small} \) is the
false drop probability in a single-triplet query. Notice
that a word with say, 4 triplets, will have a false drop
probability of \( F_{d,triplet,small} \).

Using triplets as hashing elements eliminates all
of the above problems (each triplet forms its own, sin-
gleton, set):
1) We need a look-up table of manageable size: The
possible number of triplets is \( 27^3 = 19683 \) (the
alphabet size is 27, if we include the blank and
ignore the case of the letters). A straightforward
approach would be to store all the possible tri-
plets in a table. However, we can sort the entries
lexicographically, in which case we need not store
the triplets themselves, but just the values of the
\( m \)'s. If one byte is enough to store each of the
\( m_i \)'s, we need \( \approx 20 \text{ Kb} \) of additional space, which
is reasonable. Further savings can be achieved, if
we use half a byte for each \( m_i \) (assuming that no
\( m_i \) is greater than \( 16 = 2^4 \)).
2) Insertions of new documents can not introduc-
new triplets, because our look-up table contains
all the possible triplets.

The search time of the table might seem to be
problem. However, only one disk access is required
per triplet (or even none, if we store the table in main
memory (20 Kb), during the signature extraction).
Moreover, the search time of the table is not going to
affect greatly the time to answer a query. The time to
extract the signature of the search-word is negligible with the respect to the time to scan the signature file.

The only disadvantage of using triplets versus using words as hashing elements is the increased number of false drops: When looking for a word, a block may contain all the relevant triplets, but not in consecutive positions. However, we feel that this will be rather improbable for words of reasonable size (5-6 letters long).

The most serious implementation problem of the proposed methods (as well as of any other method that takes the above frequencies into account) is the measurement of the occurrence and query frequencies $D_i$ and $q_i$. One approach is to try to obtain representative samples of the documents and the queries. Based on statistics on the above sets, estimates of the $D_i$'s and $q_i$'s can be obtained. Another problem is that the frequencies may change in the future. An elegant method to achieve reorganization in this case is not known yet. However, we may reasonably hope that changes in the occurrence frequencies will be smooth. On the other hand, sharp changes in the query frequencies may indicate that the whole data base is obsolete (the interests of the office workers have changed drastically), in which case reorganization is out of question.

7. Summary - Conclusions.

In this paper we have examined the signature file approach as an access method for formatted records and text. Specifically, we have discussed the method of superimposed coding and we have proposed a modification to it. We suggest assigning a different number $q_i (\text{number of bits set to '1'})$ to each element, according to the occurrence and query frequency. We have developed a mathematical model and derived closed form formulas that allow the optimal choice of $m_i$ for each set $S_i$ of the elements.

The theoretical results are interesting. The derived formulas indicate that, regardless of the occurrence and query frequencies, the block (or record) signatures should have half of their bits set to "1", under optimal design. The savings in false drops can be significant (≈50 per cent if the 80-20 rule holds).

The proposed model can be applied to formatted records, as well as text. In the former case, the method is mostly appropriate for files with large number of attributes and frequent insertions, where indexing does not perform well. Our method can handle even repeating groups or queries on parts of words (attribute values). Finally, since the signature file is searched sequentially, the requests can be batched. Tree organizations can not exploit the advantages of batching well [3].

Signature files using superimposed coding have been used for text retrieval in an office environment [5]. The method proposed here results in better performance, when the query and occurrence frequencies are known.

Future research could deal with handling time-varying occurrence and query frequencies. Another interesting problem is to extend the multiattribute-hashing method proposed by Gustafson [9] to take into account the occurrence and query frequencies.

Appendix

The problem is to calculate $m_i (i=1,...,n)$, such that the false drop probability $F_d$ is minimized. $w$ is the proportion of "1"'s in the signature and is given by the formula

$$w = \frac{M}{F}$$

where $M$ is the expected number of "1"'s in the signature. We have

$$M = F \left[ 1 - \left( 1 - \frac{1}{F} \right)^{m_1 + m_2 + \ldots + m_n} \right]$$

or

$$M \approx F \left( 1 - e^{-\frac{m_1 + m_2 + \ldots + m_n}{F}} \right)$$

The approximation holds for large values of $F$ (typically $F \approx 800$). Thus, $w$ is given by the formula

$$w = 1 - e^{-\frac{m_1 + m_2 + \ldots + m_n}{F}}$$

or

$$q_i w = \frac{w}{F} \ln w + q_i w = m_i w_D + m_2 w_D + \ldots + m_n w_D = 0$$

This is equivalent to

$$\frac{q_1 w_{m_1}}{D_1} + \frac{q_2 w_{m_2}}{D_2} + \ldots + \frac{q_n w_{m_n}}{D_n} = \frac{F_d}{D} = K$$

where $K$ is a constant, independent of $i$. Substituting Eq. (A.4) into (A.3) we obtain:

$$F = w - \ln w + m_1 D_1 + m_2 D_2 + \ldots + m_n D_n = 0$$

which, due to Eq. (A.2) gives:

$$\frac{w}{1 - w} = \ln (1 - w)$$

or

$$w = \frac{1}{2}$$

Eq. (A.5) and (A.2) give:

$$m_i D_i = \frac{F_d}{D}$$

and Eq. (A.4) and (A.5) give:

$$m_i = \frac{1}{\ln 2} \left[ \ln q_i - \ln K \right]$$

Eq. (A.6) and (A.7) can be solved for $K$ and give:

$$K = \frac{F_d}{D} \left[ \ln 2 \right]$$

$$\ln K = \frac{F_d (\ln 2)^2 + \sum_{i=1}^{n} D_i \ln q_i}{D}$$
which allows to solve for $r_{k}$'s:

$$m_k = \frac{F \ln 2}{D} + \frac{1}{\ln 2} \left[ \ln \left( \sum_{i=1}^{q} \frac{D_k}{D} \ln \frac{q_i}{D} \right) \right]$$

(A.9)

Thus, we have calculated the optimal values for the $m_k$'s for a given $F$. From Eq. (A.5) and (A.9) we see that

$$\ln F_d = \ln D - \frac{F \ln 2}{D} \left[ \ln \left( \sum_{i=1}^{q} \frac{D_k}{D} \ln \frac{q_i}{D} \right) \right]$$

(A.10)

References


