Signature files: Design and performance comparison of some signature extraction methods.

Chris Faloutsos

Computer Systems Research Institute,
Univ. of Toronto,
Toronto, Ontario M5S 1A4, CANADA

ABSTRACT.

Signature files seem to be a promising method for text retrieval and document retrieval [29,5,8]. According this this method the documents are stored sequentially in one file ("text file") while abstractions of the documents ("signatures") are stored sequentially in another file ("signature file"). In order to resolve a query, the signature file is scanned first and many non-qualifying documents are immediately rejected. In this paper we present three old and one new signature extraction methods and compare their screening capacities. We derive exact and approximate formulas for the false drop probability of each method and discuss the new method in more detail.

1. Introduction

Traditional data base management systems (DBMSs) are designed for formatted records. Recently there seem to be many attempts to extend these systems so that they will be able to handle unformatted, free text [5,6,14,21,29]. 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, there should exist a system that allows electronic storage and retrieval of these messages.

Another important application of a text retrieval method is the computerized library. The problem of handling queries on the contents has attracted a lot of research interest in the past few decades [27,31]. As the result of the above research activity many text retrieval methods have been proposed in the literature.

In this paper we concentrate on text retrieval methods and especially on the signature file approach, which seems to be most suitable for the office environment [29,5]. Signature files can also be applied for attribute retrieval [25,23] and this is one reason that makes the method suitable for documents. The structure of this paper is as follows: In section 2 we give a brief survey of the existing text retrieval methods. In section 3 we describe four signature extraction methods. In section 4 we

compare the performance of these methods. In section 5 we discuss the new method in more detail. In section 6 we give a summary of the paper and provide research directions.

2. Overview of text retrieval methods

In this paper, we shall ignore the existence of attributes in a document. We shall focus our attention on text retrieval methods only. Many such methods have been proposed in the literature. However, they seem to form the following large 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 [1,3,18], scanning of a large data base may take too much time [15].

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 [17], MEDLARS, ORBIT, LEXIS [27] 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 [13]. 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 a large number of non-qualifying documents are discarded. The rest are either checked (so that the "false drops" are discarded) or they are returned to the user as they are. A document is called a "false drop" if it does not actually qualify in a query, while its signature indicates the opposite. The method is faster than full text scanning mainly because the size of the signature file is much smaller. However it is expected to be slower than inversion for large files [24]. It requires much smaller space overhead than inversion (≈10% [5]) and it can handle insertions easily. If carefully designed, the signature file method can handle queries on parts of words and can tolerate errors [7].

4) Clustering. In this method, similar documents are grouped together to form clusters. Usually, they are stored physically together, too. Clustering has been extensively examined in the literature of library science [26,27,31]. It seems difficult to compare this method with the previous ones. The reason is that the emphasis in clustering is on the "relevance" queries, for example "give me the documents that are relevant to 'information retrieval' (even if a document does not contain the above two words)". Papers on the performance analysis of clustering are mainly concerned with *recall* and *precision* and seem to ignore the space overhead, the retrieval speed and the performance upon insertions. Recall is the proportion of retrieved, relevant documents over the total number of relevant documents. Precision is the proportion of retrieved relevant documents over the total number of retrieved documents. Our opinion is that the space overhead is rather small (O(log$n$), where n is the number of documents). Similarly, the retrieval time seems to be proportional to log$n$, too. However, it seems that clustering can not handle insertions easily: Van Rijsbergen [31, pp. 58-59] 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.

5) Multi-attribute hashing. Gustafson [12] proposed a multiattribute hashing scheme based on superimposed coding. His method is applicable on bibliographic data bases. All the 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. The interesting property of this method 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 the author's knowledge.

This brief discussion on text retrieval methods reveals that none of them is clearly superior over the others Therefore, we have to consider the operational characteristics of the specific environment, in order that we choose the best access method. For the office environment the main features are:

- Large data bases: 1 Gb [2], or 65 Gb [16].
- Large insertion rates, but few deletions and updates [16].
- It is felt that most of the documents in an office are never required after they have been filed. Gravina [11] reports that the access frequency of a document decreases very fast with its age.

Under the above considerations, the signature file method seems to be a reasonable choice. Tsichritzis and Christodoulakis [29] analyze the advantages of the method in more detail. In addition, it should be mentioned that the method seems to integrate well with attribute retrieval methods: Signature files have been recently used for attributes [23,25]. Tsichritzis et al. [30] describe a prototype, multimedia office filing system that applies the signature file approach both for attributes and text. Christodoulakis [4] applies this approach on image captions, in order to handle queries on images.

3. Description of the methods.

In this section we shall go into more detail and describe four methods of signature extraction. The first method [29,19] suggests that each word of the document is hashed into a bit pattern of length f. These patterns ("word signatures") are concatenated to form the document signature (see Figure 1). Searching is performed in the obvious way. For example, on a single word query the signature of the search word is extracted and all the document signatures are searched. Those that contain the signature of the search word are retrieved. In order to improve the performance, common words (e.g., "the", "a", etc.) may be ignored. In the rest of this paper, we shall refer to this method as WS.

Document	free	text	retrieval	methods
	|	|	|	|
	v	v	v	v
Word sign.	0000	0100	0111	1011
Doc. sign.	0000 0100 0111 1011			

Figure 1.
Illustration of the word signature (WS) method.
The document consists of four words only.
Each word yields a 4-bit word-signature (f =4)

The second method [5] is based on superimposed coding [22]. It will be referred to as SC for the rest of the paper. The method works as follows: Each document is divided into "logical blocks". A logical block is defined as a piece of text that contains a constant number D of distinct, non-common words. For the rest of the paper, unless otherwise stated, the term "block" stands for "logical block". Each such 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 2, the word "free" sets to "1" the 3-rd, 7-th, 8-th, and 11-th bits (m=4 bits per word). In order to allow searching for parts of words, the following method is suggested: Each word is divided into successive, overlapping triplets (e.g., " fr", "fre", "ree", "ee " for the word "free"). Each such triplet is hashed to a bit position by applying a hashing function on a numerical encoding of the triplet. In case that a word has l triplets, with $l>m$, the word is allowed to set l (non-distinct) bits. If $l<m$, the additional bits are set using a random number

generator, initialized with a numerical encoding of the word.

Word	Signature
free	001 000 110 010
text	000 010 101 001
block signature	001 010 111 011

Figure 2
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). The signature size F affects directly the number of false drops, while m has to be selected according to eq. (A.7) so that approximately half of the bits are "1" in a block signature.

The next method is a new one and is based on compression. Again, we split the document into logical blocks, as in SC. The idea is that we use a (large) bit vector of B bits and we hash each word into one (or perhaps more, say n) bit position(s), which are set to "1" (see Figure 3). The resulting bit vector will be sparse and therefore it can be compressed.

free	0000 0000 0000 0010 0000
text	0000 0001 0000 0000 0000
retrieval	0000 1000 0000 0000 0000
methods	0000 0000 0000 0000 1000
block signature	0000 1001 0000 0010 1000

Figure 3
Illustration of the compression-based methods.
With $B=20$ and $n=1$ bit per word,
the resulting bit vector is sparse and can be compressed.

Notice that this approach presents certain similarities with the two previous methods. Like SC, the sparse vector is created by OR-ing together the individual word signatures. However the differences are that

1) n is *not* selected according to eq. (A.7) We shall see in the next section which is the optimal value for n.

2) The size B of the sparse vector is much larger than a typical signature size of SC.

The similarity with the WS method is the following: For $n=1$, an obvious (but not optimal) way to compress the sparse vector is to record the positions of the "1"s, using $\log_2 B$ bits for each one. This is exactly what the WS method does, except for the fact that the sequencing information is

preserved with WS.

From the above observations it is expected that methods based on compression should perform comparably with the SC and WS methods and therefore it is worthwhile examining them in more detail.

The compression method we propose is based on bit-blocks. For the rest of the paper, it will be referred to as BC (for bit-Block Compression). In this method the sparse vector is divided into groups of consecutive bits (bit-blocks). The size of the bit-blocks is chosen according to eq. (A.22) in the appendix, in such a way that the performance is optimized. For each bit-block we create a signature, which is of variable length and consists of at most three parts (see Figure 4):

Part I) It is one bit long and it indicates whether there are any "1"s in the bit-block (1) or the bit-block is empty (0). In the latter case the bit-block signature stops here.

Part II) It indicates the number s of "1"s in the bit-block. It consists of $s-1$ "1"s and a terminating zero. This is not the optimal way to record the number of "1"s. However this representation is simple and it seems to give results close to the optimal. (see eq. (4-5) in section 4).

Part III) It contains the offsets of the "1"s from the beginning of the bit-block ($\log_2 b$ bits for each "1", where b is the bit-block size).

	b <-->				
sparse vector	0000	1001	0000	0010	1000
Part I	0	1	0	1	1
Part II		10		0	0
Part III		00 11		10	00

Figure 4.
Illustration of the BC method with
bit-block size b=4.

Figure 4 illustrates how the BC method compresses the sparse vector of Figure 3. Figures 5 and 6 illustrate two different ways of storing the block signature of Figure 4. In the first alternative (Figure 5) the parts of each bit-block signature are stored consecutively and the bit-block signatures are concatenated. In the second alternative (Figure 6) the first parts of all the bit-block signatures are stored consecutively, then the second parts etc.. This latter alternative seems to allow faster searching and it will be discussed later in more detail.

0 | 1 10 00 11 | 0 | 1 0 10 | 1 0 00

Figure 5
BC method - Storing the signature
by concatenating the bit-block signatures.
The vertical lines mark
the bit-block signature boundaries.

Another technique that uses compression was suggested by McIlroy [20] for a different environment. His goal was to compress a dictionary of D=30,000 words for a spelling-error detector program. However this method could be applied to our environment and therefore we shall examine it in detail.

0 1 0 1 1 | 10 0 0 | 00 11 10 00

Figure 6
BC method - Storing the signature
by concatenating the parts.
Vertical lines indicate the part boundaries.

A bit vector of B=2**27 was used and each word was hashed to one bit position (n=1). Then run length encoding was applied, that is the number of zeros between two successive "1"s in the sparse vector was recorded. Making realistic assumptions about the probability distribution of the above intervals (geometric distribution) a coding technique proposed by Golomb [10] and Gallager and Van Voorhis [9] was applied. This technique results in minimum-redundancy codes [9] if the intervals are geometrically distributed. Thus, McIlroy achieved very good compression of the sparse vector. In our discussion, we shall generalize this method, allowing $n \geq 1$ (i.e., each word may hash to one or more positions in the sparse bit vector). In the rest of the paper we shall refer to this generalized method as RL (Run-Length encoding).

The advantage of the method is the excellent compression: According to eq. (3,4,5) of section 4, it requires ≈0.086 bits per word more than the entropy based-bound, versus ≈0.471 of the BC method. The disadvantage of the RL method is the slow searching: In order to determine whether a bit is "1" in the sparse vector, the encoded lengths of all the preceding intervals have to be decoded and summed. Moreover, in order to skip to the next signature, the total length of the current signature has to be determined. This can not be done, unless all of the signature is scanned or unless pointers are used. In contrast, the length of a signature in the BC method can be determined from Parts I and II. More detailed discussion about the features of each method can be found in section 5.

4. Comparison of the methods.

In this section we shall compare the above four methods with respect to their screening capacity. It is obvious that all of these methods introduce "false drops", that is a signature may seem to qualify in a query, although the corresponding text does not qualify. The probability of this event to happen is called *false drop probability* F_d. Mathematically,

$$F_d = \text{Prob}\{\text{the sign. of a block seems to qualify} / \text{the block does not}\}$$

It is important to justify why we choose F_d as a measure for comparison and not another measure, such as the I/O cost or the response time etc.. There are two reasons for our choice:

- Unlike the other measures, F_d depends solely on the method and not on other factors, such as hardware configuration, buffering algorithms etc..
- Discovering the dependency of F_d on the signature size F seems to be a mathematically complicated problem. If this is solved, one could calculate the other measures for a specific setting (hardware, operating system etc.).

There are two conventions that the derived formulas are based on:

- In order to keep the analysis tractable, we deal only with single word queries. Conjunctive and disjunctive queries require further assumptions (such as independency of query words) in order to be analyzed. More complicated Boolean queries or queries with sequencing require so many additional assumptions that the results of such an analysis might not be reliable.

We remind the reader that we split each document in logical blocks, as mentioned in the SC method. This is essential for all the methods but the WS. Fixing the number of words per block allows for optimal choice of the number of bits per word m for SC, for optimal choice of the bit-block size b for BC and for the optimal construction of the Huffman codes for the RL. For the purpose of comparison, we do this splitting in the WS method.

Thus the problem we want to solve is:

Given an (expected) block signature size F

Find the false drop probability F_d for each method for *single word* queries.

In addition to the four above methods, we want to derive formulas that give the theoretical bound of what we can achieve with a compression method. These formulas are based on the entropy of a bit in the sparse vector. The quantities in these formulas will have a subscript EN (for entropy).

WS	Word Signatures
SC	Superimposed Coding
RL	compression with Run Length encoding
BC	bit-Block Compression
EN	ENtropy based bounds
VBC	Variable bit-Block Compression

Table I
List of signature extraction methods.

The exact formulas for each method are given in the appendix. They are too complicated to be repeated here and they do not provide much insight about the behavior of each method. More informative are the graphs 1-2 that plot the logarithm of F_d versus the signature size F for D=40 and for n=1,2,3. The value D=40 is an estimate for the average number of distinct non-common words in a 1Kbyte piece of text [5]. From these graphs we can draw the following conclusions:

1) All the compression-based methods (RL, BC and EN) give better results than both WS and SC for n=1.
2) The RL method gives excellent results, very close to the EN curves (Graph 2).
3) For methods based on compression, the optimal value of number of bits per word is n=1.
4) All curves become almost straight lines for large signature sizes. The WS curve is wavy because the method is more sensitive to the round-offs of eq. (A.3).
5) The graphs of all the compression-based methods have the same slope, which is the same with the slope of the WS method. As observed in [8], SC has a smoother slope because it does not make full use of all the available 2**F bit patterns, since it requires that half of the bits are "1" (see eq. (A.7)).

The fourth observation gave us the motivation to look for approximate asymptotic formulas, as the signature size F increases. The derivation of these formulas is presented in the appendix and the results are repeated here:

$$\log_2 F_{d,WS} = \log_2 D - \frac{F_{WS}}{D} \tag{1}$$

$$\log_2 F_{d,SC} = -\frac{F_{SC}}{D \log_2 e} = -\frac{F_{SC}}{D} 0.693 \tag{2}$$

$$\log_2 F_{d,RL} = n(1+\log_2 \log_2 e) - \frac{F_{RL}}{D} = 1.528n - \frac{F_{RL}}{D} \tag{3}$$

Symbol	method(s)	definition
$F_{d,XX}$	all	False drop probability for the "XX" method
F_{XX}	all	(expected) size of a block signature for the "XX" method
D	all	number of distinct non-common words per block
w	SC	probability that a bit is "1" in a block signature.
w	RL, BC, EN	probability that a bit is "1" in the sparse vector
S_{max}	WS	maximum number of distinct word signatures
f	WS	number of bits for a word signature
m	SC	number of bits that a word sets to "1"
m_{opt}	SC	the optimal value of m
l	RL	expected number of bits to encode an interval of zeros
B	RL, BC, EN	size of the sparse vector
n	RL, BC, EN	number of bits that a word sets to "1".
b	BC	size of a bit-block
b_{opt}	BC	optimal value of b

Table II.
Definitions of the symbols.

$$\log_2 F_{d,BC} = n(1+\log_2 e - \log_2\log_2 e) - \frac{F_{BC}}{D} = 1.913n - \frac{F_{BC}}{D} \quad (4)$$

$$\log_2 F_{d,EN} = n\log_2 e - \frac{F_{EN}}{D} = 1.442n - \frac{F_{EN}}{D} \quad (5)$$

It should be noted that the above formulas are very accurate. Graph 3 plots the relative per cent error in the false drop probability for the SC and WS methods (D=40) and Graph 4 the same for the compression-based methods RL, BC and EN (D=40, n=1). From Graph 3 we can observe that the approximate formula for SC is optimistic: The error is negative (between -1.6% and -0.5% approximately) and oscillates, due to the round-off of eq. (A.8). The error for the WS method oscillates much more. However, if we consider only the cases where F is a multiple of D, then the error is positive and tends to zero for large signature sizes From Graph 4 we see that the approximate formulas for RL, BC and EN are all pessimistic, that is they overestimate the false drop probability. For sufficiently large signature sizes, the error seems to become stable. These "steady-state" values are 5.837%, 1.185% and 1.450% for the RL, BC and EN methods respectively.

The second interesting point about the approximate formulas is that they fully justify all the above observations. Finally, they give rise to other quantitative observations. For example, we see that the BC method requires $\log_2 e - 2\log_2\log_2 e = 0.385$ more bits per word than the RL method in the signature size for the same false drop probability.

5. Discussion.

The idea of using a large, sparse bit vector to represent a piece of text reminds us of the clustering method [26]. The difference is that no hashing technique is used in clustering to create the bit vector, at the expense of having a fixed vocabulary and a look-up table. However, any of the above compression methods could be used to save space in representing the centroids and the document vectors in a clustering method. The only subtle point is that the presented compression methods work well if the number of "1"s in the sparse vector is constant (or at least has a small standard deviation).

The rest of the discussion will be mainly devoted to the BC method. The first observation is that the WS and SC methods can be seen as special cases of the BC method: For bit-blocks of 1 bit long ($b_{opt}=1$) and for $n=B\ln 2/D$ bits per word, the resulting signatures are the same with the ones of SC, except that Parts II and III of the BC method are needless. For $b_{opt}=B$ and $n=1$ we have signatures similar to WS, except that Parts I and II are now redundant.

As discussed in the appendix, Part I of the bit-block signatures behaves like a signature of SC: half of the bits are "1". Storing these parts consecutively (see Figure 6) may speed up the searching of the signature file, because almost half of the signatures will be rejected immediately, after just one bit comparison.

Finally, an important consideration is that the BC method can be slightly modified to become insensitive to changes in the number of words D per block. This is desirable because the need to split documents in logical blocks is eliminated, thus making the resolution of complex queries much easier. The proposed modification extracts the signature of a whole document as follows:

- We count the number of distinct non-common words D of the document.
- Using eq. (A.22) we calculate the optimal value b_{opt} for the bit-block size.
- We create the document signature as described before (the size of the sparse vector B is the same for all documents and $n=1$).
- We store both the signature and the value of b_{opt} (or, in order to save space, we store the binary logarithm of b_{opt}, since b_{opt} is a power of two).

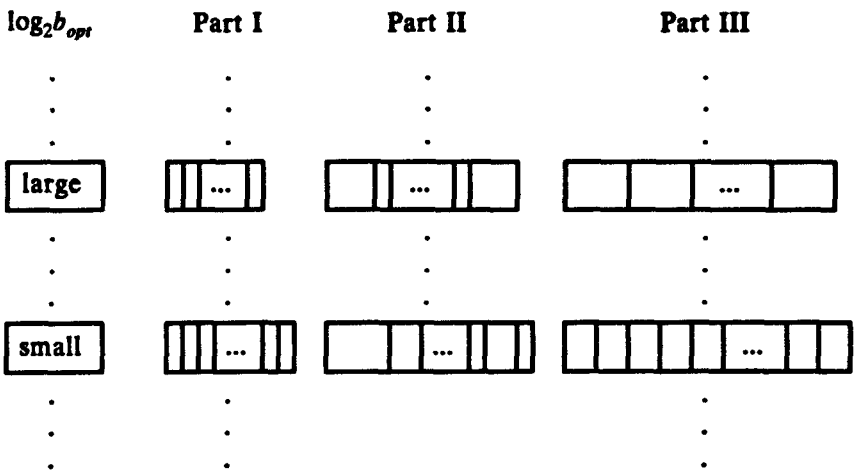


Figure 7
An example lay-out of the document signatures
in the VBC method.

This method will be referred to as VBC (Variable bit-Block Compression) for the rest of the paper. Figure 7 illustrates an example lay-out of the signatures in the proposed VBC method. The upper row corresponds to a small document with small D, while the lower row to a document with large D. Thus, the upper row has a larger value of b_{opt}, fewer bit-blocks, shorter Part I (the size of Part I is the number of bit-blocks), shorter Part II (its size increases with D) and fewer but larger offsets in Part III (the size of each offset is $\log_2 b_{opt}$ bits). It should be noted that, although the lay-out of Figure 7 seems to result in fast searching, it is not anything more than an example. More research is required in order to discover which lay-out is the most preferable.

For reasons explained before, in this paper we focused our attention on the false drop probability of each method for single word queries. The result of the study is that, from the point of view of false drop probability, the best method is RL, followed by BC, WS and SC (in this order). From a practical point of view, there are some additional considerations. To name the most important of them:

1) Speed to search a block signature.
2) Performance on more complicated queries.
3) Ability to answer queries on parts of words.
4) Preservation of the sequencing information.

We shall discuss briefly these points:

1) The fastest method for searching a signature (given that it has been brought in main memory) seems to be SC: It requires only m (typically $\approx$10) bit comparisons to accept or reject a signature in a single word query. The BC method requires some more bit comparisons, as well as some calculations in order to determine the length of Parts II and III of the block signature. The RL method needs approximately half of the encoded zero-intervals to be decoded and added, thus giving slow search time. The WS method requires the whole block signature to be examined, but it does not need any decoding or any additions.
2) All the signature methods do well on conjunctive (AND) queries. Methods that split documents in logical blocks (that is, SC, BC and RL) require more book-keeping than the rest of the methods (such as the WS method without logical blocks and the VBC method).
3) For the present time, only SC [5] can handle queries on parts of words.
4) Only the WS method preserves the sequencing information.

As a final conclusion, it is still difficult to pinpoint the most preferable method. However, we believe that the VBC method concentrates the good features of almost all the other methods:

- It achieves the second best compression (the first being the RL method), requiring approximately 0.471 bits per word more than the entropy-based bound.
- It is fast in searching, second only to SC.
- It handles gracefully documents of arbitrary size, without splitting them in blocks.

6. Summary and research directions.

In this paper we presented signature four extraction methods and analyzed their performance. Two of them were already known (WS, SC) the third (RL) was a generalization of an older method ([20] with $n \geq 1$) while the fourth (BC) is a new one. We decided to use the false drop probability as a measure for the comparison and we studied the relationship between it and the signature size F. Exact formulas were derived, the results were plotted in graphs and observations were made. Further investigation led to simple but accurate asymptotic formulas, which agreed with the previous observations and gave quantitative results. The BC method was discussed in more detail and a modification of it was proposed (VBC) to avoid splitting documents in logical blocks.

The contributions of this paper are:

1) The detailed design of the BC method.
2) The proposal of the VBC method, with variable bit-block size.
3) The derivation of exact and approximate formulas for the four methods and for the entropy-based bounds.
4) The calculation of the optimal value of the number of bits per word n for the compression-based methods (n=1).
5) The observation that the $\log_2 F_d$ vs. F lines for the WS, RL, BC and EN methods have the same slopes. This seems to be the best slope we can achieve, at least when we use hashing methods.

Further research can deal with:

- Investigation of the VBC method in more detail.
- Study the signature-search time of each method.
- Attempt to discover the signature extraction method with the best possible screening capacity, either the method is based on hashing or not.

Acknowledgements.

Prof. Stavros Christodoulakis deserves many thanks for his well-thought suggestions and his constructive criticism.

Appendix.

Analysis for WS and SC.

The exact and approximate formulas for the WS and SC method have been derived in the past: See [8] for the former and [28] for the latter. We repeat here the results: For WS we have:

$$F_{d,WS}=1-\left[1-\frac{1}{S_{\max}}\right]^D \tag{A.1}$$

with

$$S_{\max}=2^f \tag{A.2}$$

$$f=\left\lfloor \frac{F_{WS}}{D} \right\rfloor \tag{A.3}$$

and, for large signature sizes F_{WS}, we can approximate:

$$\log_2 F_{d,WS}=\log_2 D-\frac{F_{WS}}{D} \tag{A.4}$$

For the SC method, the exact formula is:

$$F_{d,SC}=w^{m_{opt}} \tag{A.5}$$

with

$$w=1-\left[1-\frac{1}{F_{SC}}\right]^{m_{opt}D} \tag{A.6}$$

and

$$m_{opt}=\frac{F_{SC}}{D\log_2 e} \tag{A.7}$$

or, since m_{opt} has to be an integer

$$m_{opt}=\text{int}\left[\frac{F_{SC}}{D\log_2 e}\right] \tag{A.8}$$

It is interesting to notice that under optimal design $w=1/2$. The approximation is [8]:

$$\log_2 F_{d,SC}=-\frac{F}{D\log_2 e} \tag{A.9}$$

Analysis for RL.

For all the compression methods, the false drop probability is given by:

$$F_{d,RL}=F_{d,BC}=F_{d,EN}=w^n \tag{A.10}$$

with

$$w=1-\left[1-\frac{1}{B}\right]^{nD} \tag{A.11}$$

The expected code length for an interval of zeros has been calculated (see, e.g., [20] or [9]) and it is found to be:

$$\bar{l}=k-1+\frac{r^x}{1-r^m} \tag{A.12}$$

where

$$r=\frac{1}{w+1} \tag{A.13}$$

m is the smallest integer, such that

$$r^m\le\frac{1}{2} \tag{A.14}$$

and

$$k=\left\lfloor\log_2 2m\right\rfloor \tag{A.15}$$

$$x=2^{k-1}m \tag{A.16}$$

Since we have to encode Bw intervals (or equivalently: "1"s) on the average, the expected signature size is

$$F_{RL}=Bw\bar{l} \tag{A.17}$$

For $B\to\infty$ we have

$$w\approx nD/B \tag{A.18}$$

From (A.12) and since $r^m\approx 1/2$, we see that $\bar{l}$ ranges from k to $k+1$, for x varying from m to 0. Thus

$$\bar{l}\approx k$$

or from (A.15)

$$F_{RL}\approx nD\log_2(2m)$$

From (A.14) and (A.13) we have

$$F_{RL}\approx nD(1+\log_2\log_2(1+w))$$

Approximating $\ln(1+w)\approx w$ we have finally:

$$F_{RL}=nD(1+\log_2\log_2 e-\log_2 w) \tag{A.19}$$

Combined with (A.10), we have the relationship between the false drop probability $F_{d,RL}$ and the signature size F_{RL}:

$$\log_2 F_{d,RL}=n(1+\log_2\log_2 e)-\frac{F_{RL}}{D} \tag{A.20}$$

Analysis for BC.

For the BC method we have that the expected number of distinct "1"s in the sparse vector is Bw. According to the description of the method (see Figure 4), we have:

$$F_{BC}=\left\lceil\frac{B}{b}\right\rceil+Bw+Bw\log_2 b \tag{A.21}$$

with each term of the right hand side corresponding to Parts I-III of the bit-block signatures. Optimizing over b we have:

$$\frac{\partial F_{BC}}{\partial b}=0=>b_{opt}=\frac{\ln 2}{w} \tag{A.22}$$

where ln stands for the natural logarithm. Since $\log_2 b_{opt}$ should be an integer, we have

$$\log_2 b_{opt}=\text{int}\left[\log_2\frac{\ln 2}{w}\right] \tag{A.23}$$

Notice that the above equation implies that, under optimal choice of b and for large values of B, half of the bit-blocks will be empty on the average:

$$\text{Prob\{empty bit-block }\}=\left[1-\frac{b_{opt}}{B}\right]^{nD}$$

$$\approx e^{-\frac{b_{opt}nD}{B}}$$

$$\approx e^{-\ln 2}=\frac{1}{2} \tag{A.24}$$

This observation emphasizes the resemblances of BC with SC (see eq. (A.7)), at least as far as the Part I of the bit-block signatures is concerned. Eq. (A.10-11) and (A.21) give the exact correlation between the false drop probability $F_{d,BC}$ and the signature size F_{BC}. In order to obtain a simplified, approximate formula, we ignore the rounding-off in eq. (A.21) and (A.23). Then we have:

$$F_{BC}=Bw(1+\log_2 e-\log_2\log_2 e-\log_2 w) \tag{A.25}$$

or from eq (A 10) and (A 17)

$$\log_2 F_{d,BC}=n(\log_2 e+1-\log_2\log_2 e)-\frac{F_{BC}}{D} \tag{A.26}$$

Analysis for EN.

Finally, the lower bound in the compression of the sparse vector is calculated using the entropy $H(w)$:

$$F_{EN} = BH(w)$$

that is

$$F_{EN} = B(-w\log_2 w - (1-w)\log_2(1-w)) \tag{A.27}$$

Although the above formula is simple enough, we can still simplify it using the approximations:

$$\ln(1-w) \approx -w$$

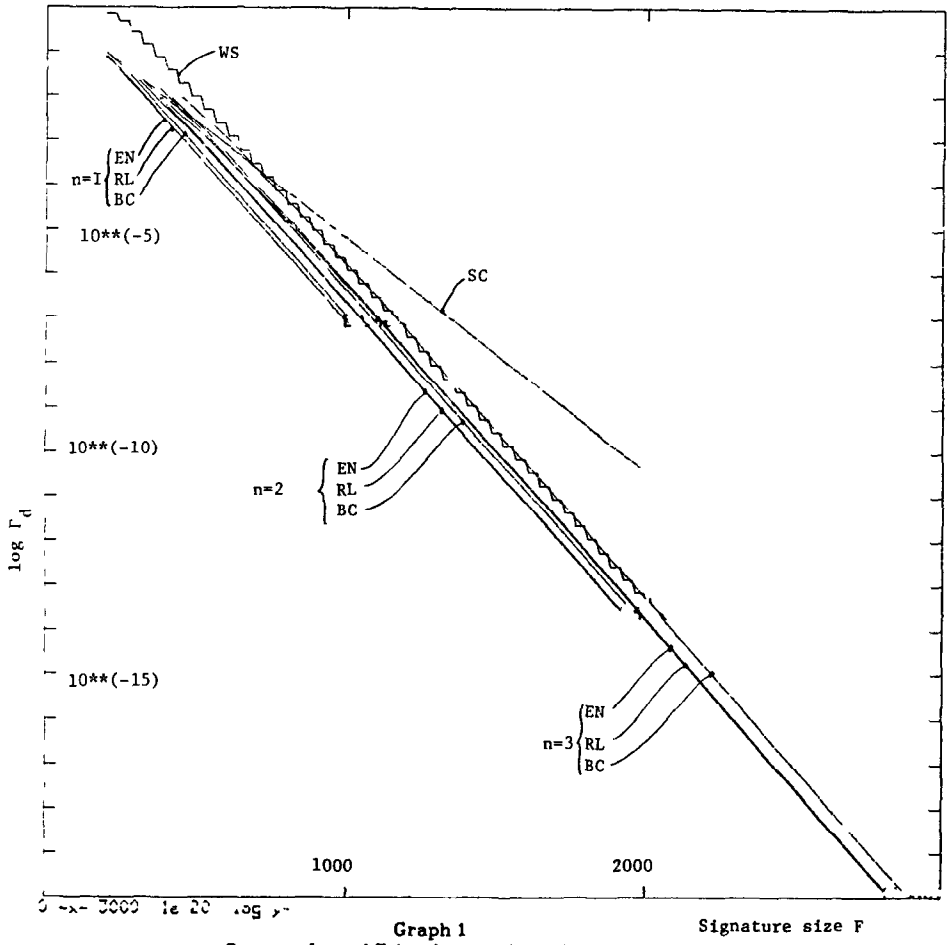
$$1-w \approx 1$$

which are true when $w \ll 1$. Then we have:

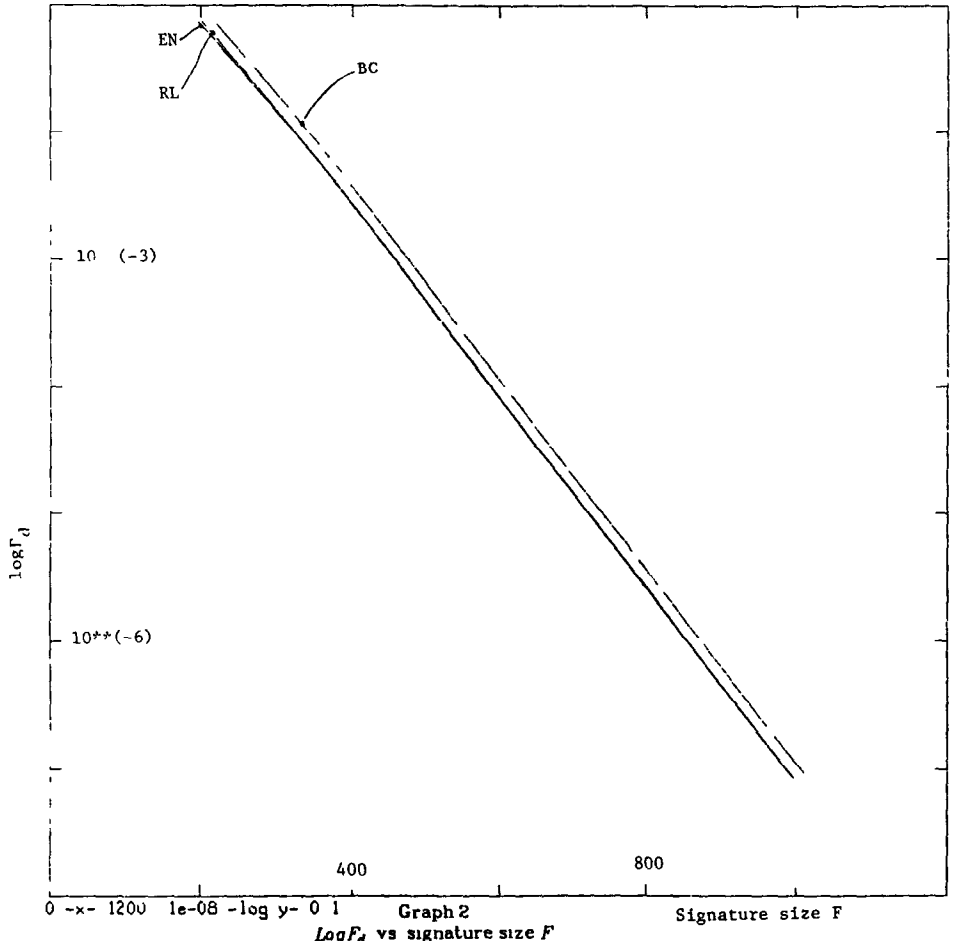
$$F_{EN} = nD(\log_2 e - \log_2 w) \tag{A.28}$$

Combined with (A.10), we have

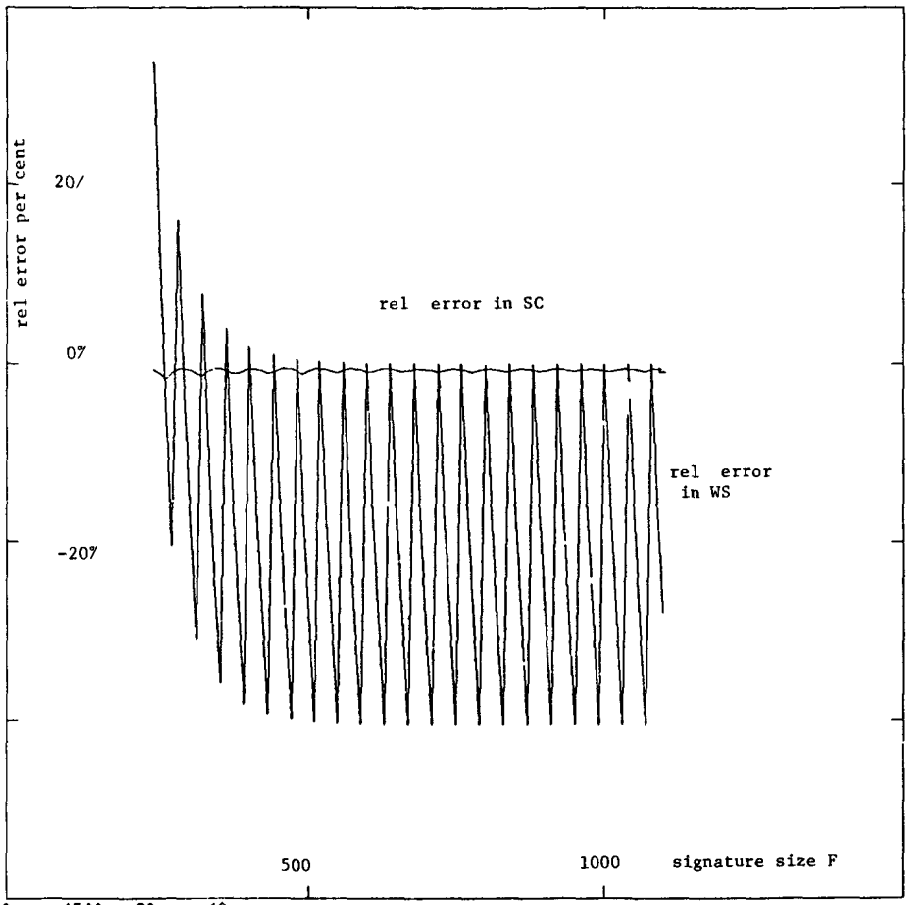
$$\log_2 F_{d,EN} = n\log_2 e - \frac{F_{EN}}{D} \tag{A.29}$$



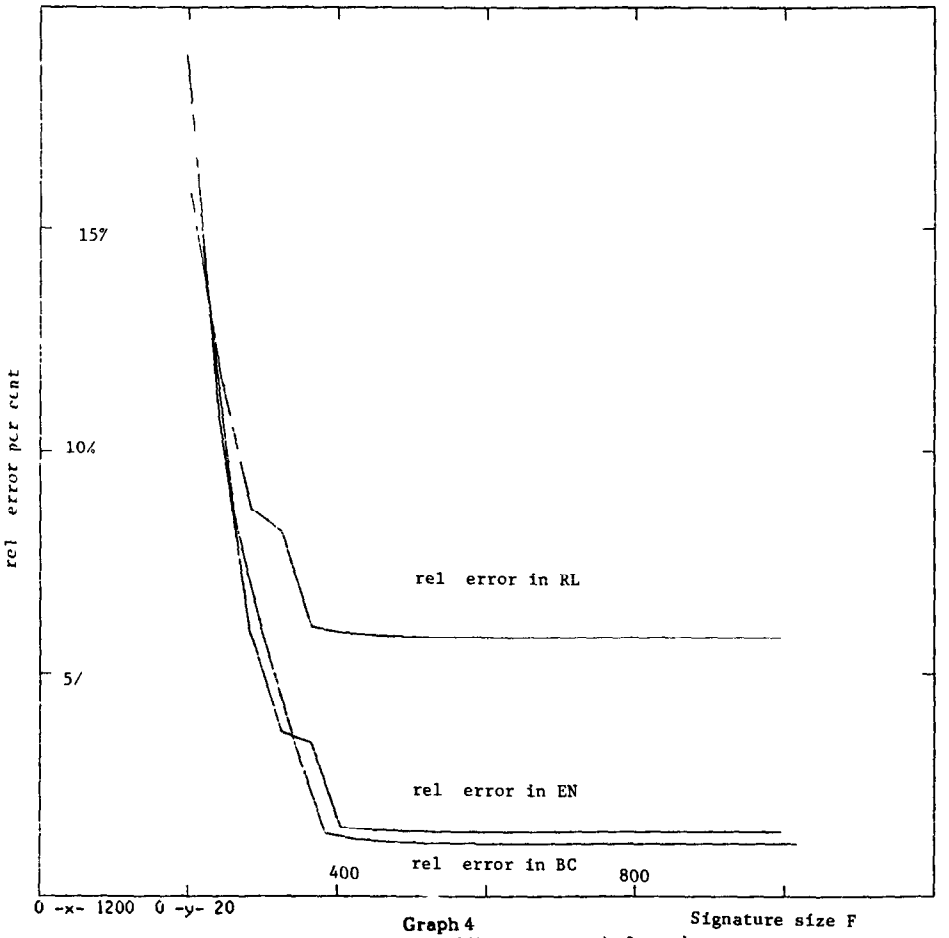
Graph 1
Common Log of False drop probability $LogF_d$
vs signature size F for all of the methods
F varies from 0 to 3000
F_d varies from 10**(-20) to 1



Log F_d vs signature size *F*
for the compression based methods with
n = 1 bit per word
F varies from 0 to 1200,
F_d varies from 10**(-8) to 0 1



Graph 3
Relative per cent error in F_d of the approximate formulas
for the SC and WS methods $D = 40$
The error varies from -60 to 40%,
F varies from 0 to 1200



Graph 4
Relative per cent error in F_d of the approximate formulas
for the compression-based methods $n = 1$, $D = 40$
The error varies from 0 to 20%,
F varies from 0 to 1200

References

1 Aho, A.V. and M.J. Corasick, "Fast Pattern Matching: An Aid to Bibliographic Search," *Communications ACM*, vol. 18, no. 6, pp. 333-340, June 1975.

2. Bird, R.M., J.C. Tu, and R.M. Worthy, "Associative/Parallel Processors for Searching Very Large Textual Data Bases," *Proc. of the 3rd ACM Workshop on Computer Architecture for Non-numeric Processing*, pp. 8-16, New York, May 1977.

3. Boyer, R.S. and J.S. Moore, "A Fast String Searching Algorithm," *CACM*, vol. 20, no. 10, pp. 762-772, Oct. 1977.

4. Christodoulakis, S., "A Framework for the Development of a Mixed-Mode Message System for an Office Environment," *Proc 3rd Joint ACM-BCS Symposium on Research and Development in Information Retrieval*, Cambridge, 1984.

5. Christodoulakis, S. and C. Faloutsos, "Design Considerations for a Message File Server," *IEEE Trans. on Software Engineering* , vol. SE-10, no. 2, pp. 201-210, March 1984.

6. Dattola, R., "FIRST: Flexible Information Retrieval System for Text," *JASIS*, vol. 30, pp. 9-14, Jan. 1979.

7. Faloutsos, C., "Text Retrieval Methods," *ACM Computing Surveys*, 1984. submitted for publication

8. Faloutsos, C. and S. Christodoulakis, "Signature Files: An Access Method for Documents and its Analytical Performance Evaluation," *ACM Trans. on Office Information Systems*, vol. 2, no. 4, Oct. 1984.

9. Gallager, R.G. and D.C Van Voorhis, "Optimal Source Codes for Geometrically Distributed Integer Alphabets.," *IEEE Trans. on Information Theory*, vol. IT-21, pp. 228-230, March 1975.

10. Golomb, S.W., "Run Length Encodings," *IEEE Trans on Information Theory*, vol. IT-12, pp. 399-401, July 1966.

11 Gravina, C.M., "National Westminster Bank Mass Storage Archiving," *IBM Systems J.*, vol. 17, no. 4, pp 344-358, 1978.

12. Gustafson, R. A., "Elements of the Randomized Combinatorial File Structure," *ACM SIGIR, Proc of the Symposium on Information Storage and Retrieval*, pp. 163-174, Univ. of Maryland, Apr. 1971.

13 Haskin, R L , "Special-Purpose Processors for Text Retrieval," *Database Engineering*, vol. 4, no. 1, pp. 16-29, Sept 1981.

14 Haskin, R L and R.A Lorie, "On Extending the Functions of a Relational Database System," *Proc ACM SIGMOD*, pp. 207-212, Orlando, Florida, 1982.

15 Hollaar, L.A., "Text Retrieval Computers," *IEEE Computer Magazine*, vol. 12, no. 3, pp. 40-50, March 1979.

16. Hollaar, L.A , K.F. Smith, W.H. Chow, P.A. Emrath, and R.L. Haskin, "Architecture and Operation of a Large, Full-Text Information-Retrieval System," in *Advanced Database Machine Architecture*, ed D.K. Hsiao, pp 256-299, Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

17. IBM,, *STAIRS/VS Reference Manual*, 1979. IBM System Manual

18. Knuth, D.E., J.H Morris, and V.R. Pratt, "Fast Pattern Matching in Strings," *SIAM J. Comput* , vol. 6, no. 2, pp. 323-350, June 1977.

19. Larson, P.A., "A method for speeding up text retrieval," *Proc. of ACM SIGMOD Conference*, San Jose, CA, May 1983.

20. McIlroy, M.D., "Development of a Spelling List," *IEEE Trans on Communications*, vol. COM-30, no 1, pp. 91-99, Jan. 1982.

21. McLeod, I.A , "A Data Base Management System for Document Retrieval Applications," *Information Systems*, vol. 6, no. 2, pp. 131-137, 1981.

22. Mooers, C , "Application of Random Codes to the Gathering of Statistical Information," Bulletin 31, Zator Co., Cambridge, Mass., 1949. based on M.S. thesis, MIT, January 1948

23. Pfaltz, J.L., W.H. Berman, and E.M. Cagley, "Partial Match Retrieval Using Indexed Descriptor Files," *CACM*, vol. 23, no. 9, pp. 522-528, Sept. 1980.

24. Rabitti, F. and J. Zizka, "Evaluation of Access Methods to Text Documents in Office Systems," *Proc. 3rd Joint ACM-BCS Symposium on Research and Development in Information Retrieval*, Cambridge, 1984.

25. Roberts, C.S., "Partial-Match Retrieval via the Method of Superimposed Codes," *Proc IEEE*, vol 67, no. 12, pp. 1624-1642, Dec. 1979.

26. Salton, G , *The SMART Retrieval System - Experiments in Automatic Document Processing*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1971.

27. Salton, G. and M.J. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill, 1983.

28. Stiassny, S., "Mathematical Analysis of Various Superimposed Coding Methods," *American Documentation*, vol. 11, no. 2, pp. 155-169, Feb. 1960.

29. Tsichritzis, D. and S. Christodoulakis, "Message Files," *ACM Trans. on Office Information Systems*, vol. 1, no. 1, pp. 88-98, Jan. 1983.

30. Tsichritzis, D., S. Christodoulakis, P. Economopoulos, C. Faloutsos, A. Lee, D. Lee, J. Vandenbroek, and C. Woo, "A Multimedia Office Filing System," *Proc 9th International Conference on VLDB*, Florence, Italy, Oct.-Nov. 1983.

31. Van-Rijsbergen, C.J., *Information Retrieval*, Butterworths, London, England, 1979. 2nd edition