Accessing Textual Documents using Compressed Indexes of Arrays of Small Bloom Filters

J. K. MULLIN
Department of Computer Science, University of Western Ontario, London, Ontario, Canada N6A 5B7

A highly compressed index for a collection of variable-sized documents is described. Arrays of small Bloom filters are used to efficiently locate documents where the search probe contains 'anded' and 'ored' combinations of words. Theoretical and experimental results are reported. The method is applicable to unplanned searching of large text files. We further describe a method to provide an index to the filters. Thus only a small proportion of the compressed filter need be examined. The method is highly amenable to parallel processing.

Received November 1985, revised May 1986

1. INTRODUCTION

Libraries often need to search large text files for particular combinations of words. Many of these files of titles and abstracts consist of unstructured text. Office files of memoranda, letters and a mixture of variously formatted documents also tend to be large and, due to diversity of formats, are often treated as unformatted. There is often a need to search these files in unplanned ways. A typical request is to locate all documents which contain boolean combinations of words – or their near meanings.

The conventional method is to construct a fully inverted file. In such a file, each word appears along with a list of all document numbers which contain that word. Function words such as: the, and, but, for, an...are excluded. One then accesses through the word and obtains the associated document identifiers. 'Anded' requests are handled by intersecting the associated document identifiers. Near meanings are treated by 'oring' words from a thesaurus or by stemming algorithms. Such fully inverted files are typically larger than the original document file. See Blair.[1] A B* tree is often used to implement the file. This allows maintenance for additions or deletions. Consider adding a new document with fifty indexable words. There will be fifty searches through the index levels of the B* tree to insert the new document identifier. Thus additions to the file are costly. Storing such a file on a video disc is inefficient due to the need for many direct accesses and block copies. Yet video disc storage is likely to become more prevalent.

The method described herein has the following facilities.

(i) It produces an index file which is nearly an order of magnitude smaller than the original file.

(ii) It permits simple insertions to the end of the file.

(iii) It permits 'anded' requests with exactly the same efficiency as single-word requests. 'ored' requests are also very simple and efficient.

(iv) Documents of various lengths are treated efficiently.

(v) It operates by doing a single sequential scan of the compressed index. This scan is highly efficient and can be done in sections using a number of simple parallel processors if desired. A pre-filter technique will be described where only the possibly satisfying sections need be scanned.

(vi) All documents which correspond to the request will be selected. There is, however, an inherent false drop rate. With a small sacrifice in storage, this can be made as small as desired.

2. BASIC METHOD

The basic method is similar to Harrison's signatures.[2] More recent expositions can be found in Faloutsos[3] or Faloutsos and Christodoulakis.[4] This last paper compares two methods: the method of word signatures and the Bloom[5] filter approach treated in this paper. Knuth[6] refers to the idea as superimposed coding. Gonnet[7] describes a similar method. The technique is to use an array of small Bloom filters. Each element of this array corresponds to a single document. In one sample test done, the documents were all titles and author names in the 1983 and 1984 *Communications of the ACM*. There were 259 documents. Each filter element consisted of five 16-bit words.

The process of building the index is:

```
FOR EACH DOCUMENT DO
  ZERO A FILTER ELEMENT
  FOR EACH NONTRIVIAL WORK IN THE
  DOCUMENT DO
    FOR EACH HASH TRANSFORM DO
        (typically 10 to 15)
      CALCULATE A BIT POSITION IN THE
         FILTER ELEMENT AND SET THAT
         BIT TO ONE.
  WRITE OUT THE FILTER ELEMENT.
```

The search process is:

```
ZERO A FILTER ELEMENT IN MAIN MEMORY.
FOR EACH ANDED WORD IN THE PROBE DO
  FOR EACH HASH TRANSFORM DO
    CALCULATE A BIT POSITION AND SET
       THAT BIT TO ONE.
FOR EACH FILTER ELEMENT IN THE INDEX DO
  READ THE FILTER ELEMENT.
  IF ALL ONES IN THE MAIN MEMORY FILTER
     CORRESPOND TO ONES IN THAT READ,
       THEN WRITE OUT THE DOCUMENT
       IDENTIFIER.
```

These operations are indeed simple. We must now investigate the expected false drop rate and what hash transforms should be used. First we investigate the error rate of the filter.

3. ERRORS

If a word is present in a document, then the bits corresponding to that word will be set. When searching, the same bits will necessarily be set and the document selected. There is no possibility of missing a document which exactly meets the search criterion. On the other hand, it is possible to accept a document which should not be selected. Recent work by Blair and Maron[1] suggests that keyword systems do indeed fail to locate, many relevant documents due to a lack of semantic information about what is really desired. We discuss broadening the search later.

Given that t transformations are used on w different words, the probability that a particular bit in the b bits of the filter is not set (given that all bits are equally likely to be selected) is:

$$Pset' = (1-1/b)^{tw}$$

This is the probability that each of the tw transforms set some other of the b bits. The probability that a bit is indeed set is:

$$Pset = 1-(1-1/b)^{tw}$$

The chance that all bits are set by a random word and hence of a false drop is:

$Pallset = Pset^{ta}$, where a is the number of words in the anded request. For safety, a should be considered as its minimum value of 1.

The optimum number of transformations has been shown by Bloom[5] to be that where $Pset = 1/2$. Thus at optimum the probability of a false drop is $(1/2)^t$. Here one-half of the bits are set to one.

4. HASH TRANSFORMATIONS

The above analysis assumes that each of the b bits in the filter is equally likely to be selected by the hash transform. Due to the small size of the bloom filter elements involved, this is difficult to achieve in practice. Good transformations were eventually achieved. The final approach encoded character position with the character to destroy the digram dependencies. This was done by taking the position within the alphabet in the range 1–26 plus blank and ′ adding the position of the character in the word. The last 5 bits of this result were then treated as a character. We merged three such characters per 16-bit machine word, thus increasing the number of possible results. The 16-bit sections were then exclusive ored together. The bit positions were arrived at by dividing the above result by t primes less than or equal to the filter width. Since only one of these primes could span the entire filter, we alternately addressed bits from the left and right in the filter. The example implementation was done on a 16-bit machine. Much of the complexity would be eliminated with a larger machine word. The above methods achieved an acceptable group of hash transforms. It must be noted that the final transforms required considerable experimentation.

5. VARIABLE-WIDTH FILTERS – IMPROVEMENTS

While the basic method would be acceptable for documents with a controlled number of keywords per document, a large variation in number of keys per document will make it unworkable since the number of set bits in each bit vector will become unacceptably high. In the case of longer documents, most of the filter bits will be set. Such documents often cause false drops. Christodoulakis and Faloutos[4] suggest dividing the documents into fixed-length sections. This can be undesirable when anded keys are used from two sections of the document. Consider the case of a letter. One may wish to locate all correspondence between two people. The solution proposed in this paper compensates well for a variable number of words per document. Each filter element is sized proportional to the number of words per document. Currently filter elements are divided into 32-bit entities. A set upper bit of each entity signals the beginning of a new element. Thus only $\frac{1}{32}$ of the space is wasted to signal the beginning of an element. Shorter documents have less filter space, while larger documents have more space. Since the index is read in a single sequential pass, there is no appreciable run-time overhead for this method above that of a fixed element size per record.

Eight different filter sizes were chosen in building the filter. Thus 32 bits were used for one- or two-word titles, 64 bits for three or four, etc., while 256 bits were used for titles of 15 or more words. There were very few long titles. An algorithm to form the document-length groupings effectively will be described later. Building the variable-length filter is very similar to building the basic filter.

When a search is initiated, main memory filter elements (masks) are set up using the hash transforms for **each** of the possible filter element sizes. This is a small overhead done before the search. One next loops through the filter elements stored in the index, finds the filter element size by counting and selects the corresponding mask and applies it. This search is quite efficient. The hardware need only do compare, mask and counting operations.

6. FILE MAINTENANCE

New records can be appended to the collection simply by appending filter elements. Deletion can be handled simply by zeroing the corresponding filter element. A zeroed filter element can never be selected. After a sizeable number of deletions, the filter should be reorganised. With a variable-length filter, one needs a sequential scan to locate the appropriate element for deletion.

7. FILTER SIZE AND EFFECTIVENESS

The size of the filter should be proportional to the number of documents, words per document and number of transforms for a given false drop error rate.

$$Pe = [1-(1-1/b)^{tw}]^t,$$

where Pe is the false drop rate, b is the filter width in bits, t is the number of transforms and w is number of words. If $b \gg 1$ and $tw \gg 1$ then $(1-1/b)^{tw}$

Table 1.

Words	Pset	Pe	Optimal bits
40	0.426	1/5077	577
45	0.465	1/2138	649
50	0.500	1/1024	721
55	0.534	1/531	793
60	0.565	1/301	865

can be approximated by $e^{(-tw/b)}$. Thus $Pset = 1-e^{(-tw/b)}$. With *Pset* set to 0.5, its optimal value, taking the natural logarithm of both sides gives:

$$b = tw/\ln 2.$$

One wishes to restrict the number of document length categories so as to limit the number of search vectors which must be initialised yet ensure good performance. We thus wish to study the change in error rate when a filter is sized for one particular number of words and another number of words is actually present. This information will be used to estimate how large a size category may be made. Table 1 provides sample information. The formula for *Pe* is used. The results are theoretical. Fifty words per document was assumed. Thus for an error rate of 1/1024, 721 bits were required. Ten transforms then suffice.

'Optimal bits' shows the number of bits required for an error rate of 1/1024. In this data a 10% addition of words above what the filter was sized for approximately doubles the error rate.

It is important to keep the number of document length categories small, since a search vector must be initialised for each category. The following method provides a fixed upper bound for storage loss as compared to a separate category for each possible document length, while controlling the error rate *Pe*. A reasonably small number of categories results.

(1) Decide on a maximum storage loss compared to a document length category for every possible length (say 10%).

(2) Decide on an allowed error rate for the biggest document in the group. This will be the worst case for the group. One should note the actual average error rate for the group will be less (say 1/1024).

(3) Let *Wu* be the upper length size in terms of number of different words per document and *Wl* be the lower length size for a document length category. Let *s* be the allowed loss. Set *Wu* to the maximum document size to be treated.

(4) $Wl = (1-s)\,Wu$. The required number of bits is $tWu/\ln 2$ for the category.

(5) Set the new $Wu = Wl-1$.

(6) If $Wu > 10$ then go back to step 4.

(7) Depending on the frequency of occurrence and importance of short documents, form pairs of word sizes into groups or form single length-size groups for the remainder. In the example, pairs of document sizes were used with good results.

One can calculate the number of groups which will be formed from the following recurrence.

Let Max = the maximum document length.
$Wu(0) = Max$
$Wu(k+1) = (1-s)\,Wu(k)$
$Wu(k) = (1-s)^k\,Max$

The recurrence limit is $Wu(k) = 10$. Thus tiny group sizes are avoided. One then calculates the number of groups k from:

$$k = \ln(10/Max)/\ln(1-s)+5 \text{ or } 10 \text{ small groups.}$$

If $(1-s) = 0.9$ then one finds:

Max	*k*
10,000	66+5 or 10 small groups
1,000	44+5 or 10 small groups
100	22+5 or 10 small groups

If $(1-s) = 0.8$, the results are even more encouraging. A 20% difference between *Wu* and *Wl* will result in only a 10% storage loss if the actual sizes of documents in the group are uniformly distributed.

Max	*k*
10,000	31
1,000	21
100	11

The search vectors can become very large for large documents. With only a few anded terms in a search probe, these vectors will be sparse indeed. In such cases, it would be better to keep a short list of the bit positions which are set rather than the entire search vector.

8. RESULTS

Experiments were done to determine the false drop rate (*Pe*) of the basic filter versus the variable-length filter. The database is those titles previously described. There was an average of 7.02 words per document (a stop list of 59 very common words were not indexed). The distribution of non-stop words per document is given in Fig. 1. Ten non-'anded' searches were done and the observed false drop rate was recorded; eight transformations were used per word. It was decided to keep the number of transformations modest so as to produce some measurable errors; 50% more filter space could reduce the error rate by more than than order of magnitude.

Table 2 shows the observed error rate, the theoretical error rate, the observed fraction of bits set, the theoretical fraction of bits set, the filter size in bytes and the ratio of filter size to the document collection size.

Notice that due to the discrete machine word sizes, the variable-length filter is 10% bigger than the basic filter; this accounts for the superior theoretical error rate. The theory and practice correspond closely for the variable-length filter, but this is not at all true for the basic filter, where performance is much worse than predicted. This is due to the fact that a high variation in the number of words per document causes some filter elements to have a high number of bits set, this greatly increasing the likelihood that the document will be selected by chance. In the variable-length filter, the fact that the observed fraction of bits set is very close to the theoretical (which assumes a truly random transformation) indicates that the transforms are indeed very close to optimum. In short, we cannot expect to improve much on what was

Count	Documents with this work count
2	* (one document per asterisk; average = 7.0)
3	******
4	**********
5	**********
6	**********
7	**********
8	**********
9	**********
10	**********
11	****
12	*****
13	**
14	*
15	*
16	
17	*
21	*

Figure 1. Distribution of words per document for the collection (259 documents).

Table 2. Comparing the basic versus the variable-length filter

	Variable	Basic
Observed error rate	1/864	1/162
Theoretical error rate	1/917	1/639*
Observed fraction of bits set	0.4283	0.429
Theoretical fraction set	0.4263	0.446
Filter size in bytes	3416	3108
Ratio of filter to file size	0.119	0.108

* Calculations assume a fixed seven words per document.

done. Results are not shown for anded requests, as no failure could be observed with the variable-length filter. This is to be expected, as Pe for A 'anded' words is Pe^A.

8.1. Prefilters

The variable-length filter provides excellent compression and a simple, highly efficient search method adaptable to parallel processing. Its main disadvantage is the necessity of searching the whole filter for each search request. Some means to avoid searching the entire filter is much desired. In fact, such a means is available.

A pre-filter is constructed for sections of the database. The database is partitioned into k sections. The first section is simply the first n/k records, the second section is the next n/k records, etc.; n is the number of records in the database. k should be selected with two criteria in mind.

(1) The probability of the search keys being found in a section should be small – certainly less than 25% – otherwise the main filter section must be examined. This probability depends largely on the kind of searches involved (a small drop rate is assumed).

(2) The pre-filter for a main filter section should be much smaller than that main filter section.

A pre-filter is a large-bit vector initially set to zero. Each word encountered in the section is transformed to set t (not necessarily distinct) bits in the filter. When searching, the anded list of search keys (oring is definitely possible) is collected. The main filter will only be searched if all bits corresponding to those in the transformed search keys have been set in the pre-filter. Thus any section of the filter which contains the search keys in any document or combination of documents will be searched. Three main factors reduce the size and fraction of the pre-filter which must be searched.

(1) Fewer transforms need be used in constructing the pre-filter than the main filter. While the false drop rate of the main filter must be quite small (certainly less than one in 4000), the false drop rate of the pre-filter can be much larger (say one in 16); a false drop in the pre-filter simply means that a main filter section must be accessed.

When a filter is optimally loaded, half the bits are set. We previously found $b = tw/\ln 2$ given that the filter is optimally loaded. In considering the filter elements themselves, the total storage B that is used is the sum of the individual filter elements in the section.

$$B = b*records = tw*records/\ln 2.$$

Here w must be the number of words per individual document. In considering the pre-filter, w takes the role of all words in the section. Thus we see that if all words were different, and the same number of transforms were used, the pre-filter size B would be equal to the size of the main filter which it covers; fortunately, neither of these two assumptions is valid.

While a main filter error rate of 1/4096 (12 transforms) might be acceptable for the main filter, a false drop rate of 1/16 (4 transforms) should suffice. This factor alone reduces the pre-filter to one-third the size of the main filter, given the above false drop rates.

(2) In natural language, common words occur with a much higher frequency than most other words. According to Dewey,[8] the 732 most common words occur 75%

of the time. The 69 most common words occur 50% of the time.

In our database of CACM titles, after excluding 59 common words, out of 1800 total words (including author names) there were 1095 different words. The size of the pre-filter is proportional to the number of different words – not the number of total words. This is because multiple occurrences of the same work set the same bits. Extrapolation of the results shows that with about 300 titles in a section, about half the words will be different. Thus with a section size of 300 titles and author names, the common-word occurrence factor will further reduce the pre-filter by a factor of two compared to the main filter size. A larger section size would reduce the size further. Combining the factor of fewer transforms and number of different words would result in a pre-filter which is one-sixth the size of the main filter section which it covers.

(3) Another consideration reduces the volume of the pre-filter which must be read. With half the bits set, when unsuccessfully searching, the first probe of the pre-filter has half a chance of failure; the second probe increases the chance of failure by a further quarter. The expected number of probes until failure is close to two. If one pre-calculates the bit positions within the filter for all anded words and then sorts these positions from low to high, then only the leading fraction of the pre-filter will usually need to be examined when the request is unsuccessful. 'Anded' words reduce the fraction that need be read much further.

Given t transformations, sorted from lowest bit position to highest, and half of the bits in the pre-filter set, what fraction of the pre-filter does one expect to need to read in a sequential scan? One may choose to access the bit positions directly within the pre-filter. Alternatively a sequential scan of the pre-filter should be preferable with the modest-sized pre-filters envisaged. On average, if four transforms are used, only 38% of the pre-filter need be scanned when a prefilter rejects a filter section.

We shall find what fraction of the pre-filter one expects to scan when the filter section does not contain the searched-for words. With four transformations the average bit positions are at $\frac{1}{5}$, $\frac{2}{5}$, $\frac{3}{5}$ and $\frac{4}{5}$. The fraction which one expects to scan is then:

$$\text{fraction} = [1+\tfrac{1}{2}+\tfrac{1}{4}+\tfrac{1}{8}]/(t+1)$$
$$\text{fraction} = 0.375$$

When $t = 8$, fraction is 0.22.

With the conservative assumptions made here, one will need to read $\frac{1}{16}$ of the volume of a filter section when rejecting a section of the database (we assume a small number of records selected, so that most sections will be rejected). It must be noted that when a large fraction of the database is selected, much input/output activity is involved – but in such a case this is unavoidable. There is yet another factor which can further reduce the work involved. If the database is ordered such that related areas cluster sequentially, say using subject classification order, then fewer main filter accesses will be required, as fewer sections would be accepted by the pre-filter.

In summary, the amount of comparisons can be drastically reduced using pre-filters when the proportion of selected records to total records is small for a search request.

9. FUTURE DIRECTIONS

A recent paper by Blair and Maron[1] noted a much smaller document recall rate than previous researchers had reported. This study used a system (STAIRS) which searches for documents based on boolean combinations of keywords. We are proposing a different technology but functionally the same type of system. Some means is necessary to broaden a search request. The current implementation when given a request to find documents containing 'computer' will do so. it will however miss documents containing 'computers' or 'computing' or 'machine' or 'digital device'.

Sometimes a user wants documents which must exactly match the request. In the request FIND 'Mullin' one does not want documents by 'Mullins' or 'Mulling'.

The filters and pre-filters can be adapted to permit searches on either word stems or the full word. The transform functions could be modified so that the first $t-1$ or $t-2$ transforms use only the stem of a word. The final 1 or 2 transforms could use the complete word. In doing an exact match, all transforms would be employed in the search vector. The search could be broadened to terms with the same stem by using only the first $t-1$ or $t-2$ transforms. There will be an increase in the error rate Pe by a factor of two or four, but more relevant documents could be found. We plan to investigate appropriate stemming methods.

Another way to broaden the search is to use a thesaurus with near synonyms. Any search term may be broadened by oring that term with near synonyms. A search request would be factored into disjunctive normal form and then each 'anded' combination would have a search vector prepared, each being matched to the filters. One must note that 'oring' increases the error rate Pe by a factor of the number of ored terms. Appropriate error rates need to be chosen with this factor in mind, increasing the size of filters and number of transforms appropriately. The increase in storage need not be drastic. One more transformation will halve the error rate at an increase in storage in the ratio of $t+1$ to t bits. The extensions of stemming and use of a thesaurus do not address all the major issues addressed by Blair and Maron, but they are certainly needed. The use of 'ored' combinations of words is yet another use for parallelism.

10. RECOMMENDATIONS

Experiments are in progress using larger documents (abstracts) to determine the appropriate number of documents for a section of the pre-filter. These results are incomplete, but demonstrate a working system with larger documents. There are many refinements to be studied, for example stemming algorithms. The basics are now known; actual use will depend on needs. I see the real problem as follows. Is there a need for a method which compresses an index to $\frac{1}{8}$ the file size and selects which portions of that index need to be read by examining a pre-filter which is small compared to the index? One can freely index any word and do ad hoc searches. The price of a search is a sequential scan of those portions of the index selected by the pre-filter; the main cost of such a search is input–output time. Only simple boolean masks and compares are needed from the processor. Boolean

AND and OR operations are possible. The NOT operation is error-prone due to false drops.

One particularly nice feature of this design for an information retrieval system is the ability to use a number of simple parallel processors. Different processors can be used to scan different sections of filters and pre-filters. There is an added use for parallelism in 'ored' requests. This method could truly use the high degree of parallelism envisaged in future computers.

One possible drawback must be noted. Unless the actual document identifiers are stored with the filter vectors, one selects purely by counting from the beginning of the file. These counts must be converted to actual document identifiers. A simple table lookup into a file storing document identifiers will suffice. In many cases the main file can itself be used for this purpose if there is a fixed-size record containing the document identifier pointing off to the remainder of the document.

REFERENCES

1. D. C. Blair and M. E. Maron, An evaluation of retrieval effectiveness for a full text document retrieval system. *CACM* **28** (3), 289–299, (1985).
2. M. Harrison, Implementation of the substring test by hashing. *CACM* **14** (12), 777–779, (1971).
3. C. Faloutsos, Access methods for text. *ACM Computer Surveys* **17** (1), 49–74, (1985).
4. C. Faloutsos and S. Christodoulakis, Signature files: an access method for documents and its analytical performance evaluation. *ACM Trans. on Office Information Systems*, **2** (4), 267–288 (1984).
5. B. Bloom, Space time tradeoffs in hash coding with allowable errors. *CACM* **13** (7), 422–426, (1970).
6. D. Knuth, *The Art of Computer Programming, III: Sorting and Searching*. Addison Wesley, Don Mills, Ontario (1973).
7. G. Gonnet, Unstructured data bases or very efficient text searching. *Proc. 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, March 1983, Atlanta USA.
8. C. Dewey, *Relative Frequency of English Speech Sounds*. Harvard University Press, Cambridge, Mass. USA (1950).
9. J. K. Mullin, A second look at bloom filters. *CACM* **26** (8), 570–571, (1983).