Using Bitmaps for Medium Sized Information Retrieval Systems

A. Bookstein and **S.T. Klein**

Center for Information and Language Studies
University of Chicago, 1100 East 57-th Street,
Chicago, Illinois 60637

The second author was partially supported by a fellowship of the Ameritech Foundation

Abstract: We describe the use of various forms of bitmaps as a basic tool for improving the search algorithms in medium sized information retrieval systems. The bitmaps considered include and extend known techniques using occurrence maps and signatures. Such an approach to text retrieval is flexible, efficient and, relative to the customary concordance approach, inexpensive in storage costs.

1. Introduction

Our ability to control textual information is being strongly influenced by a variety of technological advances. These include new means of storing and sharing information that makes possible and realistic an information system model in which large bodies of full text are compactly stored, widely distributed, and shared by a large number of interested persons. Such changes require a careful search for techniques that promise convenient and effective access to such textual databases.

The research that is required in this environment differs from that traditional in Information Retrieval (IR) in several ways. Most apparent, earlier research assumed a model of an Information Retrieval System (IRS) in which IR was based upon the existence of a collection of records about documents rather than on the full text of the documents themselves. In such a system, each record represents, in a manner convenient for machine manipulation, the content of a document. Various matching functions were tested, in which document records were related to requests; mathematical models for generating matching functions were the focus of much effort, and considerable attention was given to finding ways to take advantage of

feedback information (Bookstein, 1985).

For a number of conceptual and practical reasons, research based on the model noted above made heavy use of typically artificial, restricted, test databases which included a set of document records, a set of requests and, for each request, a list of relevant documents. The objective of the research was, given a request, to use machine means to reproduce as well as possible the list of relevant documents. This mode of research was and is highly productive. Through it many very innovative retrieval techniques were generated, and an understanding of how the performance of a retrieval mechanism ought to be evaluated developed (see for example Salton and McGill (1983)). This approach, while still very valuable, is nonetheless limited by the artificiality of the experimental databases and the limitations of the collections in size and character.

In contrast, the developments noted above are drawing our attention to the problem of IR in an environment in which

- the desired textual information is itself available in machine manipulable form as full text; and
- the texts of interest may be widely dispersed physically but easily made available through electronic communication channels.

The primary IR problems in this environment include: developing the ability to locate the desired text(s) within the system; developing the ability to interact effectively with an available database of full, natural-language text to obtain the information or textual segments that are desired; exploring new means of studying text to amplify scholarly productivity by using methods not possible before the existence of large bodies of text in machine readable form; and creating novel techniques for storing and organizing massive quantities of text to permit efficient access, consistent with the characteristics of the new media. Questions also arise regarding the development of new data structures that permit a user to impose a useful personal structure on privately held document collections; such a structure should facilitate his intellectual access to the database and potentially should be sharable with other users.

In this paper we are concerned with the problem of designing for a realistic environment a data structure, the bitmap, that appears to us to be very valuable as an alternative to the customary concordance as a means of accessing full text

databases. It is a flexible mechanism that allows us to retrieve segments of text on the basis of various logical combinations of words, and can also be extended to permit retrieval on the basis of word fragments. Many of the techniques described here have been developed elsewhere, but the sense of their unity and their impact have been lost due to their dispersal through the literature. We here attempt a coherent presentation of the main results on the use of bitmaps in IR, pointing to elaborations and extensions that may enhance their value in an IR environment.

Suppose we are given a large file consisting of text written in some natural language, and a query of the form $A_1\ A_2\ \cdots\ A_m$, where the A_i are *keywords*. Our problem is to find all the "locations" in the text where the keywords $A_1, \ldots, A_m$ "occur together". What exactly a location is and what it means to occur together is defined by the retrieval situation; one could for example require that the keywords appear in the same line, sentence or paragraph, or that they be adjacent in the text; more generally, one could impose certain metrical constraints between the keywords (see Choueka, Fraenkel, Klein and Segal (1987)). For most of what follows, the exact definition is irrelevant: we need only assume that some mechanisms exist for determining both a location and whether for a given unit of text, the co-location condition is satisfied; in practice, the proposed system can be adapted to any of the above choices.

When the size of the text is small, say up to a few hundred Kbytes, the problem can be solved by some brute-force method that scans the whole text in reasonable time. Such a method is commonly used in text-editors. At the other extreme, for very large databases spanning tens of Mbytes, a complete scan is not feasible. The usual approach in that case is to use *inverted files*, that is, auxiliary files such as a *dictionary* and a *concordance*. As used here, a dictionary is an easily searchable list of all the different words occurring in the text and usually contains for each word some statistical information such as the total number of times it occurs and the number of documents in which it appears, as well as a pointer into the the concordance. A concordance contains for every word, W, the complete list of locations in the text where W appears. Depending on the underlying hierarchical structure of the text, these references may take various forms, for example: document number, paragraph number (in the document), sentence number (in the paragraph), word number (in the sentence); or, alternatively: book number, page, line; or simply, when any other structure is lacking, the number of the physical block containing W and the offset

within the block. The retrieval process now consists of accessing the concordance for each keyword and collating the corresponding lists of references. The main drawback of this approach is its huge overhead: the size of the concordance is comparable to that of the text itself and sometimes larger. Another problem is the rigidity of the system: in order to allow fast access, the concordance must be kept sorted, so that updating the text is difficult.

In this paper we focus on medium size texts, say up to a few Mbytes; such texts are large enough so that a straightforward scanning approach is impractical. The inverted file method is a possible alternative; however the method we propose allows us to drop the concordance completely. The space and maintenance requirements are therefore drastically reduced, which can be a great advantage for smaller systems where disk-memory is rather scarce.

The idea is first to effectively reduce the size of the database by removing from consideration segments that cannot possibly satisfy the request, then to use pattern matching techniques to process the query, but only over the—hopefully small—remaining part of the database. The filtering process which reduces the amount of text to be scanned is based on assigning *signatures* to text fragments and to individual words. Signature schemes have been used in a variety of ways in information retrieval, such as substring testing (Harrison, 1971, Bookstein, 1973), document ranking (Croft and Savino, 1988) and as an access method for text (Faloutsos, 1985, Faloutsos and Christodulakis, 1984, Sacks-Davis, Kent and Ramamohanarao, 1987). In the method under investigation, the signatures are transformed into a set of bitmaps, on which Boolean operations, induced by the structure of the query, are performed. The resulting bitmap is used to eliminate *a priori* parts of the text which cannot possibly contain a solution. The way in which bitmaps, when used together with a concordance, can enhance the retrieval process for large full-text systems has been studied in Choueka et al. (1987). The present work is an extension to the case where no concordance is used; it is also related to the *grab* command in UNIX (Lesk, 1985) and incorporates some of its ideas.

This paper describes the theoretical framework of the new retrieval system. The system itself is currently in the process of being tested on various subsets of the *Trésor de la Langue Française* (TLF), a large French database available to us. The experimental results, as well as empirical comparisons with other retrieval systems, will be published later.

2. The basic scheme of the retrieval process

2.1 Description of the proposed system

In the system being proposed, every word W is assigned a signature $h(W)$ which is a string of length k bits, for some fixed integer k. The length k should be chosen to allow smooth computer manipulation (e.g., a multiple of a byte size), large enough to permit effective retrieval, and small enough to limit the overall space complexity. The function h should satisfy the basic requirements of a hash function, namely being easily computable and distributing the values $h(W)$ as uniformly as possible over its range. Generally, h will not be one-to-one, but unlike hashing, where great efforts are spent on collision resolution, the fact that several words may have the same signature does not constitute a problem here.

The text is partitioned into relatively small logical units, say sentences, and each unit S is assigned a signature, $h(S)$, obtained by ORing the signatures of all the words in S: $h(S) = \bigvee_{W \in S} h(W)$. This is known as *superimposed coding*. Similarly, for a given query $Q = A_1$ AND A_2 AND $\cdots$ AND A_m, the signature of the query is computed: $h(Q) = \bigvee_{i=1}^{m} h(A_i)$. In order for all the keywords of the query to appear in a given text unit S, it is necessary that $h(S)$ have 1-bits at least at all the positions $h(Q)$ has. This is easily determined, for example, by checking for each unit S if

$$\big(h(S) \quad \text{AND} \quad h(Q)\big) \quad \text{XOR} \quad h(Q) \quad \equiv \quad 0, \tag{1}$$

an operation efficient in machine time. However this condition is not sufficient, not only because h is not injective, but also because the OR operation, viewed as a function from $K \times K$ to K, is not injective, where K is the set of k-bit strings. Therefore the text units retrieved are only *potentially* relevant and must now be scanned to verify that all the keywords $A_1, \ldots, A_m$ actually appear in each of them. Such an approach was taken for example in Bookstein and Rodriguez (1978).

Let n be the number of text units, the text units themselves denoted by $S_1, \ldots, S_n$. When n is fairly large, it is not practical to evaluate (1) for each n. The set of n signatures representing the text can be viewed collectively as an $n \times k$ bit-matrix, M. In the approach considered here, M is created row by row, but stored column by column in so-called *bit-slices* (see for example Roberts (1979)). This yields k vectors of n bits, $C_1, \ldots, C_k$, where C_i, the i-th column-vector in M is the concatenation of the i-th bit-position of the n signatures $h(S_1), \ldots, h(S_n)$.

The retrieval process can now be reformulated as follows: let $I(Q)$ be the set of the indices of the 1-bits in $h(Q)$. We first retrieve the set of vectors $\{C_j\}_{j \in I(Q)}$. These vectors are then ANDed to form a new vector $\mathcal{F} = \bigwedge_{j \in I(Q)} C_j$ of the same length of n bits. The indices of the 1-bits in $\mathcal{F}$ are exactly the indices of the text units S which satisfy (1); in other words, $\mathcal{F}$ serves as a filter allowing us to skip the text units corresponding to its 0-bits.

In practice, the space allocated to each column vector, C_i, should exceed n bits, thus preparing for the efficient handling of a possible future expansion of the database. Indeed, there is no restriction on the order of the text units, so that later additions can simply be appended at the end. The update process would then consist of computing the signatures $h(S_{n+1})$, $h(S_{n+2})$, ... of the newly arriving units and setting the corresponding bits in the tails of the vectors C_j. The deletion of a text unit is implemented by setting all the corresponding bits in the columns C_j to zero. Note the simplicity of these update procedures as compared to the necessary reorganization required by the concordance approach.

2.2 Parameter setting

The effectiveness of the filtering process will be determined by the number of 1-bits in a typical signature $h(S)$. For example, suppose that l bits are chosen, possibly with duplications, by independent random functions in which the probability that any given bit in $h(W)$ be turned on is l/k. As shown by Stiassny (1960), if a text unit S has r words, then the expected number of 1-bits in the bit-pattern $h(S)$ is

$$k\left(1-\left(1-\frac{l}{k}\right)^{r}\right) \approx k\left(1-e^{-lr/k}\right). \qquad (2)$$

Since the function is non-linear, we cannot immediately deduce from this formula an expression for the expected number of 1-bits in $h(S)$, averaged over all the units, with varying values of r. However, if r is chosen to be the average number of words occurring in a unit, we can use (2) to estimate the average number of bits set to one.

Suppose next that a query Q contains r' words which do not appear in the text unit S. The system would still retrieve S in response to Q if the 1-bits corresponding to the r' words comprising Q happen to fall in positions occupied by 1-bits in $h(S)$: in other words, to a good approximation, if in lr' random choices, with repetitions,

only positions of 1-bits in $h(S)$ are selected. Thus the probability of such a "false drop" is $p_{\text{fd}} = \left(1 - e^{-lr/k}\right)^{lr'}$. Since p_{fd} is a decreasing function of k, the larger the size chosen for the signature, the more effective would the resulting filter $\mathcal{F}$ be, though at the price of a larger space overhead. Therefore k will be determined by implementation considerations, such as the total available space and the ease of manipulating k-bit blocks. For a fixed k, the effect of decreasing l is on one hand to decrease the probability $1 - e^{-lr/k}$ of a bit in $h(S)$ being set to 1, making a false drop more difficult, but on the other hand, to reduce the number lr' of random choices that must succeed for us to get a false drop, making a false drop easier. We are interested in chosing l such that, given k, p_{fd} is minimized (the parameter r is fixed for the database). Stiassny (1960) showed that the optimum is obtained for $l = \frac{k}{r} \ln 2$, a result independent of r'. Intuitively this result is not surprising, since for fixed k, the optimal value of l is such that the probability of a bit in $h(S)$ being set to 1 is $\frac{1}{2}$; in other words, about $k/2$ of the bits would be turned on. This agrees with our intuition that the probability of a false drop will be minimized when the number of possible k-bit signatures with s 1-bits, $\binom{k}{s}$, is maximized; this requirement again yields $s = k/2$.

As an example, let us take a sentence as a text unit; the average number of words in a sentence can be computed from the database being processed. For example, on the Trésor de la Langue Française (TLF), a large database available to us, the average length of a sentence is about 22 words; thus if k is set to 64 bits, the best choice for l would be $(64 \ln 2)/22 \simeq 2$ bits.

We are now able to assess the implications of increasing the size of a text retrieval unit—say from sentence to paragraph. If a new unit is α old units, we can increase k to αk without changing the size of our bitmap matrix. If each unit introduces different words, then r will also increase by a factor of α, to αr, and r/k in p_{fd} would remain the same, as would the optimal l, and no gain (or loss) is realized. However, in practice, if we combine α contiguous units, most likely r will become $\alpha' r$, for $\alpha' < \alpha$; if so, $\frac{r}{k} \to \frac{\alpha'}{\alpha} < 1$, and to preserve the condition that $(1 - e^{-lr/k}) = \frac{1}{2}$, l must increase. Thus $p_{\text{fd}} = (.5)^{lr'}$ will decrease. This implies that from the point of view of minimizing false drops, given the constraint of a fixed size bitmap matrix, we would like to use text units that are as large as possible. Increasing the size of a text unit, however, increases the cost of physically retrieving the unit and then of pattern matching. The actual size used should be a compromise

of these conflicting considerations.

2.3 Implementation

The following procedure for computing the signature of a word W could be used. Consider the concatenation of the internal representations of the characters of W as one long binary integer $v(W)$. Let f be a good pseudo-random number generator (see Knuth (1973), Chapter 3), returning values between 0 and 1, and denote by $f_1, f_2, \ldots$ the sequence of values returned by f when initially called with $v(W)$ as seed. Generate the first l' numbers of this sequence, where $l' \geq l$ is the smallest integer for which the set $J = \{\lceil f_1 \times k \rceil, \lceil f_2 \times k \rceil, \ldots, \lceil f_{l'} \times k \rceil\}$ contains l distinct values; J is the set of indices of the bits in $h(W)$ which shall be set to 1.

When the optimal value of l is not an integer, we propose to set the number of 1-bits in $h(W)$ either to $\lfloor l \rfloor$ or to $\lceil l \rceil$, such that on the average, l bits are set. First $\lfloor l \rfloor$ indices are chosen as above, then the next random number $f_{l'+1}$ is generated. If $f_{l'+1} > l - \lfloor l \rfloor$, we are done; otherwise an additional index has to be selected.

3. System Considerations

3.1 Including rare and frequent words

We believe bitmaps have a potential as a retrieval mechanism beyond that which has hitherto been attempted. We have described an approach, based on word signatures, for generating bitmaps that are approximate in the sense that while a 0 guarantees that the corresponding segment will not satisfy the request, a 1 does not guarantee that it will. In practice this could be combined with other bitmap based approaches to create an efficient IR system.

We first recognize that different terms may benefit from different treatments. The terms occurring in an actual database can be usefully partitioned, for our purposes, into three classes. First we must consider the class of terms occurring very rarely; it is well known that these account for the major part of the dictionary, if not of the text itself. These terms can be controlled most efficiently by simply storing their locations explicitly in a list, perhaps as part of the dictionary. Effectively, *exact* bitmaps associated with these terms can be generated algorithmically as needed directly from the information in the dictionary rather than being stored on disk, and can then be used as described below. Thus this mechanism is consistent with

the approach we are proposing. This way of dealing with the rare words also eliminates another problem caused by such words within the signature scheme, viz., that especially for these terms, the number of bits set to 1 that will cause false drops will greatly exceed the number of actual occurrences of these terms.

The second class of terms requiring special consideration consists of words occurring very frequently, commonly called *stop-words*, such as `to`, `for`, `the`, etc. For such words, each bit set to 1 will substantially fill a column of the bit-matrix, effectively reducing the size of the signature. Typically, such words are simply ignored. It should however be noted that the removal of stop-words, while appropriate for languages like English, cannot easily be applied to languages like Hebrew, in which most of the words are homographic. For example, at the Responsa Retrieval Project (RRP), which has a database of about 60 million words, written mainly in Hebrew and Aramaic (see for example Choueka (1980) or Fraenkel (1976)), no stop-words are defined and *all* the words are searchable, including some with hundreds of thousands of occurrences. But information retrieval systems in other languages, for example French, have also found it advisable to include even the most frequently occurring words (Dendien, 1986). Thus a low cost approach that extends the above scheme to encompass the stop-words is desirable. For example, new bit-columns, serving as occurrence maps for the stop-words, could be added to the k columns of the signatures (see Sacks-Davis et al. (1987)). There will be one column for each of the most frequent stop-words; these will virtually consist only of 1's. For the somewhat less frequent stop-words, two or more could share the same column, yielding a new bit-matrix with a very high density of 1's. This matrix can therefore be compressed efficiently, for example by complementing each column and then using one of the known techniques for compressing sparse vectors, e.g., (Teuhola, 1978) or (Fraenkel and Klein, 1985). Thus we retain the bitmap approach, but at less cost than actually increasing the value of k.

The scheme of Section 2 seems to be best suited for the words of the third class, the intermediate range, which, when stop-words are ignored, account for the large majority of entries in the concordance. Summarizing, the filter $\mathcal{F}$ used in the retrieval process we are proposing is generated using bit-columns for *every* term in the query, though the way in which these columns are generated and stored depends on the frequency of each term in the database. Also, some bitmaps are accurate representations of the occurrence of their terms, others are approximate in that some segments will be retrieved that do not contain the term. At the highest level, the

system calls a *bitmap server* to produce bitmaps as needed. The server has detailed information about the different terms and may use a relatively complex algorithm to construct a bitmap or simply retrieve it from storage, as appropriate for the class of which the bitmap is a member.

3.2 General Boolean queries

To be consistent with most of the earlier literature, we have considered until now only simple queries consisting of the ANDing of some keywords. In more general queries, an OR operator can be used, as in `information` AND (`science` OR `retrieval`), which should retrieve locations in the text including either of the two phrases `information science` or `information retrieval`. The above procedure is readily extendable to complex Boolean queries. It is most convenient if the query is given in *disjunctive normal form*, that is in the form $Q = D_1 \vee \cdots \vee D_t$, where each of the disjuncts D_i consists of the conjunction of keywords, $D_i = D_{i1} \wedge D_{i2} \wedge \cdots \wedge D_{im_i}$. Then each of the D_i is processed separately: to compute the signature of D_i, the signatures of D_{ij} are ORed yielding $h(D_i) = \bigvee_{j=1}^{m_i} h(D_{ij})$; the 1-bits of $h(D_i)$ indicate the columns C_ℓ which should be ANDed to compute the bitmap that points to possible text-units that satisfy D_i. Finally, the resulting vectors are ORed and we get

$$\mathcal{F} = \bigvee_{i=1}^{t} \left(\bigwedge_{\ell \in \{\text{indices of 1-bits of } h(D_i)\}} C_\ell \right). \tag{3}$$

It is easy to implement an algorithm for computing $\mathcal{F}$ keeping never more than three vectors of the size of $\mathcal{F}$ in main memory, independently of t and m_i (see Choueka et al. (1987)).

It should be noted that it is possible to use the signature approach with arbitrary queries involving AND and OR: either all the Boolean operations are directly performed with large bit-columns (each term being replaced by the bitmaps derived from its signature) or the request can be reformulated into a form permitting more efficient bitmap manipulation. The decision of whether a query should be used as given or transformed into another logically equivalent form will depend on the expected different processing times.

Extending the retrieval process to general Boolean expressions that allow also the use of the NOT operator is however not straightforward, as was already noted in Choueka et al. (1987). A NOT operator is useful, e.g., in queries like `security` AND (NOT `council`), which retrieves locations of the word `security` only if it is

not in the same retrieval unit as the word `council`; here council is said to be *negated*. Such requests often come from proximity searchs—here, all instances of `security` not followed by `council`. The first idea which comes to mind for satisfying this request is to use for a negated word the 1's complement of the bitmap used for the word when it is not negated. But recall that while, in the maps considered so far, a 0-bit indicates the absence of the word from a text unit, a 1-bit does not necessarily indicate its presence: the text unit is only potentially relevant, and must ultimately be scanned in order to see whether the word appears or not. If we complement such a bitmap, a text unit corresponding to a 0-bit in the complemented map will not be retrieved, even though it might not contain the negated word and thus possibly be relevant. Therefore using complemented maps for negated words may result in loss of information.

The simplest way to proceed is to insist that a negated term only occurs in clauses with non-negated terms to which it is connected with an AND operator, as in the case above. If so, it is easy to see that, when converted to disjunctive normal form, such a negated term will occur in disjuncts together with at least one non-negated term. Hence we can initially process the request ignoring the negated terms in the bitmap manipulation stage, and afterwards test whether the full request, including proximity as well as negation requirements, is satisfied. This approach is well within the signature philosophy of using bitmaps as a filter preceding thorough search.

3.3 Truncated terms

A desirable feature of a retrieval system is the ability to locate words which are only partially specified and include *don't care characters*. For example, using the keyword `comput*`, one wishes to retrieve words like `computer`, `computing`, `computation`, etc. This is straightforward in a scanning approach, where just the truncated keyword is searched for. For systems with inverted files, the solution is trickier (see Bratley and Choueka (1982)). However the immediate extension of the signature approach, in which the same hashing function h, that defined the segment signature, is applied to the substring, fails completely, because even if one string is a substring of another, their signatures may be completely unrelated.

The following variant, based on ideas from (Harrison, 1971), could be used. Instead of computing signatures for every word directly, we divide the word into

successive substrings (or fragments), t bytes in length, and compute a signature for every word-fragment; the signature of the word is then the ORing of the signatures of the fragments comprising it. For example, if $t = 3$ and the word `compute` appears in the text unit, then signatures are generated for the trigrams `com`, `omp`, `mpu`, `put` and `ute`. The choice of t sets a lower bound to the minimal length of the word stem which can be searched for, so to maximize the degree of truncation that can be accommodated, t should be 1. However, reduction of terms to individual characters would result in very inefficient searches. Letter pairs or triplets ($t = 2$ or 3) seem to be reasonable choices; a larger t would impose too strong a restriction on the extent to which terms could be truncated.

The signature of a fragment could again be computed using some randomizing function. Alternatively, we could use the fact that there is fairly good knowledge of the distribution of bi- and trigrams in natural languages (Heaps, 1978): the set of possible bi- or trigrams being relatively small, a table could be used which assigns signatures with near to uniform distribution. For such an application, the number of signatures to be ORed for a word of w characters is $w + 1 - t$, so the size k of the signature should probably be chosen larger than for the variant assigning signatures to whole words; as above, the number of 1-bits per fragment signature will be $l = (k \log 2)/(w + 1 - t)r$, where the denominator is the average number of word-fragments per text unit. The signature of a query is now formed in a parallel fashion; for example if $t = 3$ and the query consists of the term `comput*`, the signature is obtained from the strings `com`, `omp`, `mpu` and `put` (see also Bookstein et al. (1978)).

3.4 Hierarchies

The reason for keeping the length of the signature, k, relatively small is that the space requirements for storing the bitmap table are proportional to k. Since, for given k, the optimal value of l is such that on the average about half of the bits in the signature of each text unit are set to 1, this will also be true for the columns; this means the bitmap table can probably not be compressed. However, consider the effect of increasing k, while keeping l fixed. Such an increase in signature size enables better discrimination between the words and thus should lead to savings in retrieval time. Further, a larger k implies sparser columns, and if these are sparse enough, they can perhaps be compressed, using for example the

hierarchical bit-vector compression technique described, for example, in (Vallarino, 1976, Jakobsson, 1982, Choueka, Fraenkel, Klein and Segal, 1986). The levels of the hierarchy would most naturally be made to conform to the structure of the text, e.g., if the components of the original columns correspond to sentences, the next level could be a vector with one bit per paragraph, where bit i is obtained from ORing the bits of the sentences in the i-th paragraph. The next level could be a vector for chapters, etc. The structure of the text being constant, only the non-zero blocks in each level need to be kept for any column, usually resulting in savings if the original vector is sparse enough.

The different levels can now be used directly when one wishes to redefine the retrieval unit to be, for example, a paragraph instead of a sentence. The levels can also enhance efficiency of the retrieval process, since the basic operation defined by equation (3) can initially be performed on the highest level, instead of involving the original bit-columns. The next lower level is only accessed for the units corresponding to the 1-bits of the vector $\mathcal{F}$. If, however, the vectors are not sparse enough, hierarchical processing would be a waste of time and space, as most of the higher levels would consist of 1's.

A different way to use hierarchies is to use two signatures h_1 and h_2 per text unit, of respective lengths k_1 and k_2, where $k_1 < k_2$. The columns C_j constructed from the signatures of length k_1 would be used as described above; however these will only be used in a first step to reduce the number of units to be accessed during the second step. But in the earlier procedure, we would continue by performing a pattern match on the surviving text units. Instead, we now use the longer signature of k_2 bits to improve discrimination. In the system we envisage, the k_2 bits of h_2 are to be stored with the text itself. A unit S will actually be scanned for a query Q only if *both* signatures $h_1(Q)$ and $h_2(Q)$ correspond to the two signatures $h_1(S)$ and $h_2(S)$, as defined by (1). To summarize this approach, the basic scheme of Section 2 is applied to reduce the number of *accesses*, while a second signature serves to lower the number of units to be *scanned* once they are accessed. Working in two steps may permit us to choose a smaller k_1 than would be necessary with a single step, thus speeding up the first part, at the expense of a larger k_2 (see Sacks-Davis et al. (1987)).

4. Conclusions

The use of bit-slices in conjunction with segment signatures is a technique which has recently received a considerable amount of theoretical attention in the literature. In this paper, we described this technique and indicated how it could be used to satisfy the manifold requirements of a practical information retrieval system. Some of our considerations are important for implementation: for example, integrating into the system very rare and very frequent words; implications of the possibility of introducing modern compaction techniques on the bitmaps; extending the method to include OR as well as AND operations (and perhaps the NOT operator as well); and testing various hierarchical methods for effectively increasing the size of the signature.

Implementing an IRS based on bitmaps, a number of practical problems have to be resolved. For example, a first step is to decide on the basic text units. While a sentence seems to be a good choice, defining a sentence in terms of the information available in an existing database is not always trivial. The format of full text databases has not yet been standardized, and for many databases (e.g., TLF) there exists very little structural information. Since different categories of words are treated differently, we must collect statistics on the number of words (with and without stop-words) in different text units. Samples of real queries which were submitted by scholars in linguistics and literature should be analyzed in order to evaluate characteristics of the search terms in a typical query. Based on such information, we can evaluate the distribution of the number of bitmaps used per query so the parameters of the system could be set, namely, the size k of the signature and the minimal number of 1-bits to be used for the signature of a single word in order to assure efficient processing.

We are arguing, that once the effort is made, bitmap approaches can be very valuable as a means of organizing an information retrieval system.

References

Bookstein A., (1973). On Harrison's substring testing technique, *Comm. ACM* **16** 180–181.

Bookstein A., (1985). Probability and fuzzy-set applications to information retrieval, *Annual Review of Inf. Sc. and Technology* **20** 117–151.

Bookstein A., Rodriguez C.E., (1978). Performance test of hybrid access method, *J. Library Automation* **11** 41–46.

Bratley P., Choueka Y., (1982). Processing truncated terms in document retrieval systems, *Inf. Processing & Management* **18** 257–266.

Choueka Y., (1980). Full text systems and research in the humanities, *Computers and the Humanities* **XIV** 153–169.

Choueka Y., Fraenkel A.S., Klein S.T., Segal E., (1986). Improved hierarchical bit-vector compression in document retrieval systems, *Proc. 9-th ACM-SIGIR Conf.,* Pisa; ACM, Baltimore, MD 88–97.

Choueka Y., Fraenkel A.S., Klein S.T., Segal E., (1987). Improved Techniques for Processing Queries in Full-Text Systems, *Proc. 10-th ACM-SIGIR Conf.,* New Orleans 306–315.

Croft W.B., Savino P., (1988). Implementing ranking strategies using text signatures, *ACM Trans. on Office Inf. Systems* **6** 42–62.

Dendien J., (1986). Un système de gestion de bases de données textuelles, *Proc. Conf. on Computers and the Humanities,* University of Toronto 252–266.

Faloutsos C., (1985). Signature files: Design and performance comparison of some signature extraction methods, *Proc. ACM-SIGMOD Conf.,* Austin; ACM, New York 63–82.

Faloutsos C., Christodulakis S., (1984). Signature files: An access method for documents and its analytical performance evaluation, *ACM Trans. on Office Inf. Systems* **2** 267–288.

Fraenkel A.S., (1976). All about the Responsa Retrieval Project you always wanted to know but were afraid to ask, Expanded Summary, *Jurimetrics J.* **16** 149–156.

Fraenkel A.S., Klein S.T., (1985). Novel compression of sparse bit-strings, *Combinatorial Algorithms on Words*, NATO ASI Series Vol **F12**, Springer Verlag, Berlin 169–183.

Harrison M.C., (1971). Implementation of the substring test by hashing, *Comm. ACM* **14**, 777–779.

Heaps H.S., (1978). *Information Retrieval, Computational and Theoretical Aspects*, Academic Press, New York.

Jakobsson M., (1982). Evaluation of a hierarchical bit-vector compression technique, *Inf. Proc. Letters* **14** 147–149.

Knuth D.E., (1973). *The Art of Computer Programming, Vol.* **II**, *Semi-numerical Algorithms*, Addison-Wesley, Reading, Mass.

Lesk M.E., (August 1985). GRAB — Inverted indexes with low storage overhead, Memorandum, Bell Communications Research.

Roberts C.S., (1979). Partial match retrieval via the method of superimposed codes, *Proceedings IEEE* **67** 1624–1642.

Sacks-Davis R., Kent A., Ramamohanarao K., (1987). Multikey access methods based on superimposed coding techniques, *ACM Trans. on Database Systems* **12**, 655–696.

Salton G., McGill M.J., (1983). *Introduction to Modern Information Retrieval*, McGraw Hill, New York.

Stiassny S., (1960). Mathematical analysis of various superimposed coding methods, *Amer. Documentation* **11** 155–169.

Teuhola J., (1978). A compression method for clustered bit-vectors, *Inf. Proc. Letters* **7** 308–311.

Vallarino O., (1976). On the use of bit-maps for multiple key retrieval, *SIGPLAN Notices, Special Issue* Vol. **II** 108–114.