# Improved Techniques for Processing Queries in Full-Text Systems

Y. Choueka[1,2,4], A.S. Fraenkel[3], S.T. Klein[3], E. Segal[1]

[1] Inst. for Information Retrieval and Computational Linguistics (IRCOL) — The Responsa Project
[2] Department of Mathematics and Computer Science, Bar-Ilan University, Ramat Gan, Israel
[3] Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel
[4] On sabbatical leave at Bell Communications Research, Morristown, New Jersey 07960, USA

Part of this work was done while the second and third authors were partially affiliated with IRCOL

**ABSTRACT**

In static full-text retrieval systems, which accommodate metrical as well as Boolean operators, the traditional approach to query processing uses a "concordance", from which large sets of coordinates are retrieved and then merged and/or collated. Alternatively, in a system with $\ell$ documents, the concordance can be replaced by a set of bit-maps of fixed length $\ell$, which are constructed for every different word of the database and serve as occurrence maps. We propose to combine the concordance and bit-map approaches, and show how this can speed up the processing of queries: fast ANDing and ORing of the maps in a preprocessing stage, lead to large I/O savings in collating coordinates of keywords needed to satisfy the metrical and Boolean constraints. Moreover, the bit-maps give partial information on the distribution of the coordinates of the keywords, which can be used when queries must be processed by stages, due to their complexity and the sizes of the involved sets of coordinates. The new techniques are partially implemented at the Responsa Retrieval Project.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1987 ACM 089791-232-2/87/0006/0306—75¢

## 1. Motivation and Introduction

Traditionally, processing queries in static full-text systems which accommodate metrical operators in addition to Boolean operators, has been supported by the (now classical) structure of a text, dictionary and concordance files. In a typical example, to find all the documents that contain the keywords $A$ and $B$ in the same sentence, one has to access the concordance, via the dictionary, to retrieve the sets of coordinates of $A$ and $B$, and to collate these sets in order to find all occurrences of $A$ and $B$ with the same document, paragraph and sentence numbers.

Often, however, each keyword stands for a family of linguistically different variants, all semantically equivalent for the given search, and each with its own set of coordinates. These families can be very large, especially for highly inflected languages such as Hebrew, for which every noun has a few thousand grammatical variants and every verb up to 20,000 (in just one of the seven possible *modes*)! Thus the coordinates of the variants in each family must first be merged before doing the collation. This situation, coupled with the fact that the merged sets can contain hundreds of thousands of elements, make the search almost impossible to conduct with a limited internal memory and a reasonable response time.

Full-text information retrieval systems may be partitioned according to the level of specificity supported by their queries. For example, in a system operating at the *document*-level, queries can be formulated as to the presence of certain keywords in each document of the data base, but not as to their exact locations within the documents. Similarly, one can define the *paragraph*-level and *sentence*-level, each of which is a refinement of its predecessor. The highest specificity level is the *word*-level, in which the requirement is that the keywords appear within specified distances of each other. For example, one could retrieve all the occurrences of $A$ and $B$ such that there are at least two but at most five words between them. Also the paragraph and sentence-levels permit metrical constraints, e.g., at the sentence-level one could ask for all the occurrences of $A$ and $B$ in the same or adjacent sentences.

For systems supporting retrieval only at the document level, a different approach to query processing might be useful. The idea is to replace the concordance of a system with $\ell$ documents by a set of *bit-maps* of fixed length $\ell$. Given some fixed ordering of the documents, a bit-map $B(W)$ is constructed for every distinct word $W$ of the data base, where the $i$-th bit of $B(W)$ is 1 if $W$ occurs in the $i$-th document, and is 0 otherwise. Processing queries then reduces to performing logical OR/AND operations on binary sequences, which is easily done on most machines, instead of merge/collate operations on more general sequences. Davis & Lin [6] were apparently the first to propose the use of bit-maps for secondary key retrieval. It would be wasteful to store the bit-maps in their original form, since they are usually very sparse (the great majority of the words appear in very few documents). Various methods for the compression of such large sparse bit-vectors are suggested in [5] and [8]. However, the concordance can be dropped only if *all* the information we need is kept in the bit-maps. Hence, if we wish to extend this approach to systems supporting queries also at the paragraph, sentence or word-level, the length of each map must equal the number of paragraphs, sentences or words respectively, which is infeasible for large systems. Moreover, the processing of metrical constraints is hard to implement.

We present here new techniques in which, basically, the concordance and bit-map approaches are combined: at the cost of marginally expanding the inverted files' structure, compressed bit-maps are *added* to the system; these maps give *partial* information on the location of the different words in the text and their distribution. Some parts of the method were recently implemented at the Responsa Retrieval Project (RRP), a full-text information retrieval system of about 48,000 documents, containing some 53 million words, written mainly in Hebrew and Aramaic. More details on RRP can be found in [4] and [7]. The new approach solves the above mentioned problems in all those cases where the number of retrieved documents is not too large, as happens for all well-posed queries.

## 2. Statement of the Problem

A query consists of an optional level-indicator, $m$ keywords and $m-1$ metrical constraints, as in

$$\textit{level}:\ A_1\,(l_1,u_1)\ \cdots\ A_{m-1}\,(l_{m-1},u_{m-1})\,A_m. \tag{1}$$

The keyword $A_i$ represents a set of words $A_i = \bigcup_{j=1}^{n_i} A_{ij}$, all of which are considered synonymous for the given query. Typically, most of the $A_{ij}$ are grammatical variants of some noun or verb. The $l_i$ and $u_i$ are (positive or negative) integers satisfying $l_i \le u_i$ for $1 \le i < m$, with the couple $(l_i, u_i)$ imposing a lower and upper limit on the distance from $A_i$ to $A_{i+1}$. Negative distance means that $A_{i+1}$ may appear before $A_i$ in the text. The distance is measured in words, sentences or paragraphs, as prescribed by the level-indicator. In case the latter is omitted, word-level is assumed.

In its simplest form, the keyword $A_i$ is a single word or a (usually very small) set of words given explicitly by the user. In more complex cases, a variable length don't care character $*$ can be used, which stands for an arbi-

trary, possibly empty, string. This allows the use of prefix, suffix and infix truncated terms in the query. For example, $A_i$ could be **comput***, representing, among others, the words **computer**, **computing**, **computerize**, etc.; or it could be ***mycin**, which retrieves a large class of antibiotics; infix truncation can be very useful for spelling foreign names, such as **Ba*tyar**, where * could be matched by **h**, **k**, **kh**, **ch**, **sh**, **sch**, etc. At the highest level of complexity, all the grammatical variants of some word are required. Although these are easily generated in some languages like English, sophisticated linguistic
tools are needed for languages such as Hebrew, Arabic and many others. For example, suppose that the database is in Hebrew and that $A_i$ is the family of the variants of the word "daughter". An element of this family would be the phrase "and when our daughters", because its translation in Hebrew is a single 10-character word, which has retained only one letter from the original stem of "daughter". For all complex queries, the families $A_i$ are constructed in a preprocessing stage. Algorithms for generating the families identified by truncated terms can be found in [3], and for the families of grammatical variants in [1].

For every word $A_{ij}$ there is an entry $\mathcal{C}(A_{ij})$ in the concordance, listing the coordinates of all the occurrences of $A_{ij}$ in the data base. In the sequel, we identify any keyword $A_i$ with the merged list of coordinates of $A_i = \bigcup_{j=1}^{n_i} A_{ij}$. The most general form of the problem of satisfying query (1) consists of finding all the $m$-tuples $(a_1, \ldots, a_m)$ of coordinates satisfying

$$\begin{array}{c} \forall i \in \{1, \ldots, m\} \quad \exists j \in \{1, \ldots, n_i\} \\ \text{with} \quad a_i \in \mathcal{C}(A_{ij}) \end{array} \tag{2}$$

and

$$l_i \le d(a_i, a_{i+1}) \le u_i \qquad \text{for } 1 \le i < m, \tag{3}$$

where $d(x, y)$ denotes the distance from $x$ to $y$ on the given level. Every $m$-tuple satisfying (2) and (3) is called a *solution.*

The general definition of the metrical constraints allows great flexibility in the formulation of the query. Consecutive words are retrieved with $(l_i, u_i) = (1, 1)$; this can be used for compound names (**Soviet Union**), foreign expressions (**status quo**), etc. The query **true** (-2,2) **false** can be used to retrieve the phrases **true or false** and **false or true**; since these words appear frequently in some mathematical texts, searching for **true** and **false** in the same sentence could generate noise. A lower bound greater than 1 is needed for example when one wishes to locate phrases in which some words $X_1, X_2, \ldots$ appear, but the juxtaposition of these words $X_1X_2 \cdots$ forms an idiomatic expression which we do not wish to retrieve. For example, ... **the security of the council members assembled here** ... should be retrieved by the query **security** (2,4) **council**.

A well-known problem in retrieval systems is the handling of "negative" keywords, i.e., words the *absence* of which is required in a certain context. A negation operator is particularly useful for excluding known homonyms so as to increase precision. For example, searching for references to the US President, one could submit the query **Reagan** (-2,1) **−Donald**. The general form of a query as given in (1) should therefore include the possibility of negating some — but not all — of the keywords. In connection with negative keywords, the constraints $(l_i, u_i) = (0, 0)$ can be used to restrict some large families of keywords. For example **comput*** (0,0) **−computer*** would retrieve **computing**, **computation**, etc, but not **computer** or **computers**. The definition (3) implies that one can impose metrical constraints only on adjacent keywords. In the query **A** (1,5) **B** (2,7) **C**, the pair (2,7) refers to the distance from **B** to **C**. If we wish to impose positive bounds on the distances from **A** to both **B** and **C**, this can be done by using negative distances: **C** (-7,-2) **A** (1,5) **B**, but this procedure cannot be generalized to tying more than two keywords to **A**.

At the end of the search process, the solutions are presented to the user in the form of a list of the identifying numbers or the titles of the documents that contain at least one solution, possibly together with the text of the sentence (or the paragraph), in which this solution occurs. The exact details of the display depend on the specific system, on the target population and on the human-interface design of the system. One possible presentation is in form of a KWIC-index (see e.g. Heaps [9]). One of the keywords is chosen by the system as *axis*, that is, its occurrences will appear centered on the screen. One line is displayed for every $m$-tuple. Once the axis is fixed, different $m$-tuples may produce the same KWIC-line. For example if the query is `United (1,7) Nations` and a part of the text is `... the United States are part of the United Nations and ...`, then the set of retrieved couples contains two elements. If the keyword `United` is chosen as axis, this will result in two KWIC-lines:

```
          ... the United States are part
are part of the United Nations and ...
```

On the other hand, if `Nations` is chosen as axis, the two solutions produce the same line:

```
 of the United Nations and ...
```

Thus after having chosen the axis, the search problem (1) becomes that of finding the different KWIC lines which satisfy the constraints imposed by the query. Note that for certain values of $m$, $l_i$ and $u_i$, it is possible that a given displayed line does not include the occurrences of all the elements of the corresponding $m$-tuple, because of the fixed length of the former.

## 3. The Power of Bit-Maps

For every distinct word $W$ of the data base, a bit-map $B(W)$ is constructed, which acts as "occurrence"-map at the document level. The length (in bits) of each map is the number of documents in the system. Thus, in the RRP for example, the length of each map is about 6K bytes. These maps are stored in compressed form on a secondary storage device. At RRP, the compression algorithm was taken from [5], reducing the size of a map to 350 bytes on the average. This compression method was used for only about 10% of the words, those which appear at least 70 times; for the remaining words, the *list* of document numbers is kept and transformed into bit-map form at processing time. The space needed for the bit-map file in its entirety is 33.5 MB, expanding the overall space requirement of the entire retrieval system by about 5%.

At the beginning of the search (1), the maps $B(A_{ij})$ are retrieved, for $i = 1, \ldots, m$ and $j = 1, \ldots, n_i$. They are decompressed and a new map ANDVEC is constructed:

$$\text{ANDVEC} = \bigwedge_{i=1}^{m} \left( \bigvee_{j=1}^{n_i} B(A_{ij}) \right). \qquad (4)$$

The bit-map ANDVEC serves as a "filter", for only documents corresponding to 1-bits in ANDVEC can possibly contain a solution. Note that no more than three full-length maps are simultaneously needed for the operations in (4), as e.g. in the following implementation:

```
ANDVEC ← string of 1's
do $i \leftarrow 1$ to $m$
    ORVEC ← string of 0's
    do $j \leftarrow 1$ to $n_i$
        ORVEC ← ORVEC $\vee$ $B(A_{ij})$
    end
    ANDVEC ← ANDVEC $\wedge$ ORVEC
end
```

The logical operators $\vee$ and $\wedge$ are applied on bits with the same indices.

For certain queries, in particular when keywords with a small number of occurrences in the text are used, ANDVEC will consist only of zeros, which indicates that nothing should be retrieved. In such cases the user gets the correct if somewhat meager results, without a single merge or collate action having been executed. But even if ANDVEC is not null, it will usually be much sparser than its components. In order to see how this map can improve the retrieval process, we need first some information on the way the concordance is stored.

The concordance contains, for each distinct word of the data base, the ordered list of the coordinates of its occurrences. Since the file is usually too big to be stored in internal memory, it is kept in compressed form on secondary storage, and parts of it are fetched when needed, and decompressed. In the RRP, the length of every coordinate is 8 bytes, consisting of codes for author, document, paragraph, sentence and word numbers, as well as some flags. The compression method used is the *prefix-omission* technique, which basically consists of storing a common prefix of consecutive entries only once. The method is described in [3] and yields on the RRP database a compression gain of 35%.

The compressed concordance file is partitioned into equi-sized blocks such that one block can be read by a single I/O operation. A block can contain coordinates of many "small" words (i.e., words with low frequency in the database), but on the other hand, the coordinate list of a single "large" (high-frequency) word may extend over several consecutive blocks. In the RRP, for example, about half of the words appear only once, but on the other hand there are some words that occur hundreds of thousands of times! The concordance is accessed via the dictionary, which contains for each word a pointer to the corresponding (first) block. It is for the large words that the bit-map ANDVEC may lead to significant savings in the number of I/O operations. Rather than reading *all* the blocks to collect the list of coordinates which will later be merged and/or collated, we access only blocks which contain coordinates in the documents specified by the 1-bits of ANDVEC. Hence if the map is sparse enough, only a *subset* of the blocks need to be fetched and decompressed. To implement this idea, we need, in addition to the bit-map, also a small list $L(W)$ for each large word $W$, $L(W) = \{(f_j, \ell_j)\}$, where $f_j$ and $\ell_j$ are respectively the document numbers of the first and last coordinate of $W$ in block number $j$, and $j$ runs over the indices of blocks which contain coordinates of $W$. The list $L(W)$ is scanned together with the bit-map, and if there is no 1-bit in ANDVEC in the bit-range $[f_j, \ell_j]$, the block $j$ is simply skipped.

There are, however, savings beyond I/O-operations. Once a concordance block containing some coordinates which might be relevant is read, it is scanned in parallel with ANDVEC. Coordinates with document numbers corresponding to 0-bits are skipped. For the axis, which is the first keyword $A_i$ to be handled, this means that only parts of the lists $C(A_{ij})$ will be transferred to a working area, where they are merged. The lists of the keywords $A_j$ $(j \neq i)$ are directly collated with the axis. Such collations can be involved operations, as the metrical constraints may cause each coordinate of the axis to be checked against several coordinates of every variant of other keywords, and conversely every such coordinate might collate several coordinates of the axis. Therefore the use of ANDVEC may save time by reducing the number of collations. Moreover, after all the variants of the second keyword have been collated with the axis, the coordinates of the axis which were not matched can be rejected, so that the axis may shrink considerably. Now ANDVEC can be updated by deleting some of its 1-bits, which again tends to reduce the number of read operations and collations when handling the following keywords. The updates of the axis and ANDVEC are repeated after the processing of each keyword $A_i$ of (1).

For conventional query processing algorithms, the consequence of increasing the number $m$ of keywords is an increased processing time, whereas the set of solutions can only shrink. When $m$ is increased with the bit-map approach, however, the time needed to retrieve the maps and to perform some additional logical operations is usually largely compensated for by the savings in I/O operations caused by a sparser ANDVEC. The new approach seems thus to be particularly attractive for a large number of keywords. Users are therefore encouraged to change their policy and to submit more complex queries!

Another possible application of the bit-maps is for getting a selective KWIC-display. A user is often not interested in finding *all* the occurrences of a certain phrase in the data base, as specified by the query, but only in a small subset corresponding to a certain author or a certain period. The usual way to process such special requests consists in executing first the search ignoring the restrictions, and then filtering out the solutions which are not needed. This can be very wasteful and time-consuming, particularly if the required sub-range (period or author(s)) is small. The bit-maps allow the problem to be dealt with in a natural way, requiring only minor changes to adapt the search program to this application. All we need is to prepare a small repertoire $\mathcal{R}$ of fixed bit-maps, say one for each author, where the 1-bits indicate the documents written by this author, and a map for the documents of each year or period, etc. The restrictions can now be formulated at the same time the query is submitted. In the first line of the above algorithm, ANDVEC will not be initialized by a string containing only 1's, but by a logical combination of elements of $\mathcal{R}$, as induced by the additional restrictions. Thus user-imposed restrictions on required ranges to which solutions should belong on one hand, and query-imposed restrictions on the co-occurrence of keywords on the other, are processed in exactly the same way, resulting in a bit-vector, the sparsity of which depends directly on the severity of the restrictions. As was pointed out earlier, this may lead to savings in processing time and I/O operations.

If a query including some negative keywords $D_i$ is submitted at the document-level, one can use the binary complements $\overline{B(D_i)}$ of the maps, since only documents with no occurrence of $D_i$ (indicated by the 0-bits) can be relevant. However, for other levels, the processing is not so simple. In fact, if the query is not on the document level, the bit-maps of the negative keywords are useless, and ANDVEC is formed only by the maps of the positive keywords. Nevertheless, ANDVEC will be useful for the negative words to the same extent as to the positive, since only coordinates in the relevant documents have to be checked *not* to fall in the vicinity of the axis, as imposed by the $(l_i, u_i)$.

## 4. More Details on the Search Process

As mentioned earlier, the system has to choose the axis for the collations. More generally, the order of handling the keywords needs to be fixed. Not all the $m!$ orderings are possible, as varying metrical constraints may force a linear search, once the axis is fixed. For example, if in the query "A (2,3) B (1,3) C" the keyword A is chosen as axis, then the next one to be collated should be B. If we deal first with C, assuming that it must appear between 3 to 6 words after A, and then only turn to look for B, 1 to 3 words before C, then the following string would be retrieved: "... x A x x x B x C x ..."; however, it does not satisfy the query because the distance from A to B is 4. Hence the order of processing the keywords is more restricted. In fact, at any stage, one of the keywords can be chosen which is adjacent to one of those already handled. Let $\sigma = (\sigma(1), \ldots, \sigma(m))$ be a permutation of $(1, \ldots, m)$ giving the order of processing of the keywords. If $A_i$ is the axis, i.e., $\sigma(1) = i$, then $\sigma$ is determined by choosing the subset of size $i-1$ from the set $\{\sigma(2), \ldots, \sigma(m)\}$ for the elements $j$ of which $\sigma(j) < i$. Thus the number of possible orderings is $\sum_{i=1}^{m} \binom{m-1}{i-1} = 2^{m-1}$.

Intuitively, the keyword with the smallest number of coordinates seems to be a good choice for the axis. This, however, is not always optimal as can be seen from the following example. Suppose A, B, C and D have 90, 10000, 100 and 100 coordinates respectively and the query is "A (1,3) B (2,4) C (3,5) D". If the axis is A, but then the large list of B is the next to be collated with 90 coordinates. If on the other hand we collate first C with D, then the list to be collated with B will probably be much smaller. In addition, the corresponding ANDVEC, which was updated after the collation of C with D, will be sparser, so that chances are good that not all the coordinates of B need to be considered.

We propose the following heuristic for deciding the collate order (not yet implemented). Every keyword is assigned a normalized *weight* $W_i$, which is the number of coordinates of keyword $A_i$, divided by the number of coordinates of the largest keyword list, so that $0 < W_i \leq 1$. Among the $2^{m-1}$ possible permutations $\sigma$, choose one which minimizes

$$P(\sigma) = \sum_{i=1}^{m} \left( \prod_{j=1}^{i} W_{\sigma(j)} \right)$$

Note that information on the number of coordinates of each keyword is usually stored in the dictionary, so that this heuristic can be applied without accessing the concordance.

The formula $P(\sigma)$ is based on the assumption that the size of the collation of two lists is proportional to the sizes of these two lists. It is of course easy to construct artificial examples in which very large lists of adjacent keywords have an empty collation, whereas the much smaller lists of other adjacent keywords produce many relevant couples of coordinates. Nevertheless, the above heuristic can be justified when no other a priori knowledge of the distribution of coordinates is available. As to the time needed to evaluate $P(\sigma)$, it should be noted that the number of keywords $m$ is usually small. Therefore even the straightforward $O(m2^m)$ algorithm could be used, but there is a dynamic programming algorithm, the time complexity of which is only $O(m^2)$.

Consider the query "A (1,9) B (5,9) C" and suppose A is chosen as axis. Only the coordinates of A will be stored in the working area. However, for the correct processing of the keyword C, we need to store, for every coordinate of A, the distances to several matching coordinates of B. More precisely, given the ordered set $S$ of coordinates of B which "intersect" a fixed coordinate $x$ of A, we keep the distance from $x$ to the first and last elements of $S$. If the axis is not at one of the ends $A_1$ or $A_m$, additional space is needed for the possible matchings in *both* directions. In order to reduce the needed space, one can restrict the processing to only *one* direction: after having fixed the axis $A_i$, possible processing orders are either $A_{i+1} \cdots A_m A_{i-1} \cdots A_1$ or $A_{i-1} \cdots A_1 A_{i+1} \cdots A_m$. For $i = 1$ and $i = m$, only one order is possible, thus there are only $2(m-1)$ collate orders which have to be considered.

As mentioned in the introduction, the number of coordinates of every keyword may be very large for certain queries. This may be true even after the filtering process by ANDVEC, in case each keyword includes some very frequent words as variants. Since the available space in internal memory is restricted, it will sometimes not be possible to store all the coordinates of the axis simultaneously. Note that not only space for the coordinates is required, but for each of them additional information needs to be kept as to the locations of other coordinates with which they collate. In such cases, we partition the data base into sub-ranges corresponding to disjoint sets of documents, and the query will be processed by *stages*, dealing at each stage only with coordinates of one sub-range. The size of each sub-range is limited on one hand by the available space. On the other hand, choosing the range too small would unnecessarily increase the number of stages and thus the overall processing time, because there is a certain overhead when passing from one stage to the following. This overhead is caused by the fact that when reading a concordance block, the coordinates which belong to subsequent sub-ranges should either be stored separately (which is only possible for small sets, since the reason for processing by stages was a problem of too small memory space), or else in one of the next stages, the same block will have to be read again.

Let $M$ denote the maximal number of coordinates which can be stored at each stage (as mentioned above, only coordinates of the axis are stored, together with some additional fields needed to process the metrical constraints). Here and below, let $i$ be the index of the axis. When $A_i$ consists of a single word ($n_i = 1$), then the blocks $\mathcal{C}(A_{i1})$ can be read sequentially, until $M$ coordinates are collected. However for $n_i > 1$, the coordinates of *all* the variants of $A_i$ in the given sub-range will be merged, therefore the size of the sub-range has to be fixed in advance. The bit-maps, giving partial information on the distribution of the coordinates, are used to estimate the number $K$ of documents which should be included in each sub-range.

Define a random variable $X_j^k$ as the number of coordinates of the word $A_{ij}$ (the $j$-th variant of the axis) in document number $k$. For our probabilistic analysis we assume:

(a) for fixed $j$, the random variables $X_j^k$ are equally distributed; let $\mu_j$ denote the expectation $\mathbf{E}(X_j^k)$, and $\sigma_j^2$ the variance $\mathbf{V}(X_j^k)$;

(b) the random variables $X_j^k$ and $X_j^l$, with $k \neq l$, are non-correlated;

(c) the random variables $X_j^k$ and $X_t^k$, with $j \neq t$, are non-correlated.

Let $Y_K$ denote the number of coordinates of the axis $A_i$ in a certain set of $K$ documents. From assumption (a) follows that $Y_K$ does not depend on the specific set of documents chosen, but only on its size $K$. We are interested in finding the maximal possible $K$ such that the probability $\mathbf{P}$ satisfies

$$\mathbf{P}(Y_K > M) < \epsilon,$$

where $\epsilon$ is some small predetermined probability. Then the size of each sub-range is chosen to be $K$. In case the number of coordinates in a certain sub-range actually exceeds the permitted maximum $M$ (which happens with probability $< \epsilon$), some of the coordinates are rejected, using a rather complex and time-consuming procedure. Let $\mu = (\sum_{j=1}^{n_i} \mu_j)/n_i$ and $\sigma^2 = (\sum_{j=1}^{n_i} \sigma_j^2)/n_i$; denote by $N(x)$ the number of 1-bits in the bit-map $x$, and define

$$\alpha = \frac{\sum_{j=1}^{n_i} N(B(A_{ij}) \wedge \text{ANDVEC})}{N(\bigvee_{j=1}^{n_i} B(A_{ij}) \wedge \text{ANDVEC})}. \tag{5}$$

**Theorem.** *Given the assumptions (a), (b) and (c) and the constants $M$ and $\epsilon$, we get $\mathbf{P}(Y_K > M) < \epsilon$ by choosing*

$$K \leq \frac{2\epsilon\mu M + \sigma^2 - \sqrt{\sigma^2(4\epsilon\mu M + \sigma^2)}}{2\epsilon\mu^2\alpha}. \tag{6}$$

**Proof:** We first evaluate the expectation $\mathbf{E}(Y_K)$. Let $H(K)$ be any set of indices of $K$ documents. Then $Y_K = \sum_{k\in H(K)} \sum_{j=1}^{n_i} X_j^k$. We now partition the set $H(K)$ into $n_i$ mutually disjoint subsets $H_s(K)$, $1 \leq s \leq n_i$: $H_s(K)$ is the set of the indices of the documents in which exactly $s$ of the $n_i$ variants of the axis appear. Since we shall restrict ourselves to documents corresponding to 1-bits in ANDVEC, for every document $k$ in $H(K)$, at least one of the variants $A_{ij}$ has coordinates in $k$. Thus to each document $k \in H_s(K)$ corresponds a set $P_{sk} \subseteq \{1, \ldots, n_i\}$ of size $s$, such that the word $A_{i\ell}$ has coordinates in $k$, for each $\ell \in P_{sk}$. Therefore

$$Y_K = \sum_{s=1}^{n_i} \sum_{k\in H_s(K)} \sum_{\ell \in P_{sk}} X_\ell^k.$$

Denote by $\mathcal{A}_{sk}$ the random variable $\sum_{\ell\in P_{sk}} X_\ell^k$. We evaluate the expectation of $\mathcal{A}_{sk}$ by conditioning on the possible choices of the set $P_{sk}$ for a given $k$. Let $P_s$ be the set of the $\binom{n_i}{s}$ subsets of size $s$ of $\{1, \ldots, n_i\}$. Then

$$\mathbf{E}(\mathcal{A}_{sk}) = \sum_{p\in P_s} \mathbf{E}(\mathcal{A}_{sk} \mid P_{sk} = p)\ \mathbf{P}(P_{sk} = p).$$

All the choices are equally likely, thus

$$\mathbf{E}(\mathcal{A}_{sk}) = \frac{1}{\binom{n_i}{s}} \sum_{p\in P_s} \sum_{\ell \in p} \mu_\ell.$$

Now each of the indices $\ell \in \{1, \ldots, n_i\}$ appears in exactly $\binom{n_i-1}{s-1}$ subsets $p \in P_s$, hence

$$\begin{aligned}\mathbf{E}(\mathcal{A}_{sk}) &= \frac{1}{\binom{n_i}{s}} \binom{n_i - 1}{s-1} \sum_{\ell=1}^{n_i} \mu_\ell \\ &= \frac{s}{n_i} \sum_{\ell=1}^{n_i} \mu_\ell \ = \ s\,\mu,\end{aligned}$$

so that

$$\mathbf{E}(Y_K) = \mu \sum_{s=1}^{n_i} s\,|H_s(K)|. \tag{7}$$

Let $R = N(\bigvee_{j=1}^{n_i} B(A_{ij}) \wedge \text{ANDVEC})$ be the number of 1-bits in the vector corresponding to the merged axis, after intersection with ANDVEC, i.e., $R$ is the number of documents including all the possibly relevant coordinates of the axis. For a given set of $K$ documents, consider the corresponding bit-positions in the map $B(A_{ij})$, and denote by $N_j(K)$ the number of 1-bits in these positions. Due to assumption (a), $N_j(K)$ does in fact only depend on the size $K$ and not on the specific set, and

$$\begin{aligned}N_j(K) &= \frac{K}{R} N_j(R) \\ &= \frac{K}{R} N(B(A_{ij}) \wedge \text{ANDVEC}).\end{aligned} \tag{8}$$

However, by the definition of $H_s(K)$, we have

$$\sum_{s=1}^{n_i} s\,|H_s(K)| = \sum_{j=1}^{n_i} N_j(K), \qquad (9)$$

so that we get from (7)–(9) $\mathsf{E}(Y_K) = K\,\mu\,\alpha$. Using a similar proof and the assumptions (b) and (c), we evaluate the variance of $Y_K$ and get $\mathsf{V}(Y_K) = K\sigma^2\alpha$. By Chebychev's inequality

$$\begin{aligned}\mathsf{P}(Y_K > M) &= \mathsf{P}(Y_K - \mathsf{E}(Y_K) > M - \mathsf{E}(Y_K)) \\ &\le \frac{\mathsf{V}(Y_K))}{(M - \mathsf{E}(Y_K))^2}.\end{aligned}$$

This will be bounded by $\epsilon$ if $\mathsf{V}(Y_K) \le \epsilon\,(M - \mathsf{E}(Y_K))^2$. Substituting the formulæ for expectation and variance, we get the quadratic equation

$$(\epsilon\alpha^2\mu^2)\,K^2 - (2\epsilon M\alpha\mu + \alpha\sigma^2)\,K + \epsilon M^2 \ge 0,$$

which yields the bound in (6). ∎

Intuitively, it is not surprising that $\mathsf{E}(Y_K))$ is proportional to both $K$ and $\mu$. The problem is that a 1-bit in $\bigvee_{j=1}^{n_i} B(A_{ij}) \wedge$ ANDVEC can be caused by one or more of the components, so that we need some measure, to which extent the 1-bits of the maps $B(A_{ij}) \wedge$ ANDVEC are overlapping. This measure is $\alpha$. Indeed, if the $n_i$ maps have no overlapping 1-bits (in particular, if $n_i = 1$), then $\alpha = 1$. On the other hand, if the $n_i$ maps are all logically equivalent, then $\alpha = n_i$.

To evaluate the bound for $K$ given in (6), we need the quantities $\mu$, $\sigma^2$ and $\alpha$. We propose to include in the dictionary for each word $A_{ij}$ also estimates for $\mu_j$ and $\sigma_j^2$. In fact, the estimate for $\mu_j$ is the number of coordinates of $A_{ij}$ divided by the number of documents in which they appear, but the dictionary does generally not provide the necessary information to estimate $\sigma_j^2$. The above algorithm for the evaluation of ANDVEC can easily be adapted to calculate also $\alpha$. The order of execution of the main loop should be chosen so that the *last* iteration corresponds to the axis. If ANDVEC$_j$ denotes the value of ANDVEC at the end of the $j$-th iteration, note that we evaluate $N(B(A_{ij}) \wedge$ ANDVEC$_{m-1})$ in the last iteration, whereas the vector ANDVEC mentioned in (5) refers to the final form ANDVEC$_m$. However, the result will be correct, because ANDVEC$_m \Leftrightarrow$ ANDVEC$_{m-1} \wedge (\bigvee_{j=1}^{n_i} B(A_{ij}))$, so that the maps $B(A_{ij}) \wedge$ ANDVEC$_{m-1}$ and $B(A_{ij}) \wedge$ ANDVEC$_m$ are logically equivalent. Therefore the space of only one additional map is needed to adapt the algorithm to this application. Efficient algorithms for the function $N(x)$ are in [10, Section 1.1].

For certain full-text retrieval systems, the ideal probabilistic model assumed in the Theorem is not always appropriate. For systems with great variability in the lengths of its documents, it would be more realistic to assume that for fixed $j$, $X_j^k$ is proportional to the length of document $k$. Assumption (b) seems to hold, but sometimes the documents are ordered by topics, and then adjacent documents often treat the same subject, so that $X_j^k$ and $X_j^\ell$ may be positively correlated if $|k - \ell|$ is small. As to assumption (c), the co-occurrences in a given document of variants of a keyword are certainly correlated, if for example these variants are different grammatical forms of the same word. Therefore the size of each sub-range should for certain retrieval systems be chosen smaller than suggested in (6).

The treatment of queries by stages can also be used in the following way: the user of an on-line retrieval system is usually not willing to wait more than a few minutes to get a response from the computer, but complex queries, as the ones discussed here, might need more time. Using the approach by stages, the size of the sub-ranges can be kept small enough to permit a reasonable processing time (say, one minute) for each stage, and the KWIC-list of the first stage will be displayed immediately when available. The user can then browse through this list while the computer prepares the subsequent ones.

Certain special cases deserve particular treatment. When all the metrical constraints are constant ($l_i = u_i$ for all $i$ in (1)), the processing of the query is not restricted to progress by adjacent keywords, and in this case the keywords are dealt with by increasing order of the number of their coordinates. Another feature on constant distances is that no coordinate can appear in more than one solution. Therefore we can reduce the additional space needed to store information on possible collations for each coordinate, or put otherwise, the fixed available space in memory can accommodate more coordinates of the axis in each stage.

## 5. Conclusions

By adjoining a set of compressed bit-maps to large full-text information retrieval systems, their overall space requirements are only moderately increased, but the search process may be significantly speeded up for most queries. This approach would be particularly attractive in an optical storage context, where both the texts and the inverted files are stored on a CD-ROM or a Write-Once medium. Indeed, these technologies are characterized by the availability of huge amounts of disk-memory (at almost no cost), while on the other hand, input/output processes are typically slow, and, when driven by micro-computers (which is usually the case), computing power is also limited. Hence, any method that can minimize input/output operations and CPU cycles at the expense of additional disk-memory is obviously appropriate. At the other end of the spectrum, for smaller systems, where disk-memory is rather scarce, the bit-map approach can perhaps be pushed further by dropping the concordance; the missing information can then be retrieved by standard pattern matching, using for example the algorithm of Boyer & Moore [2]. We leave this approach for further research.

## References

[1] **Attar R., Choueka Y., Dershowitz N., Fraenkel A.S.,** KEDMA — Linguistic tools for retrieval systems, *J. ACM* **25** (1978) 52–66.

[2] **Boyer R.S., Moore J.S.,** A fast string searching algorithm, *Comm. ACM* **20** (1977) 762–772.

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

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

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

[6] **Davis D.R., Lin A.D.,** Secondary key retrieval using an IBM 7090–1301 system, *Comm. ACM* **8** (1965) 243–246.

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

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

[9] **Heaps P.,** *Information Retrieval, Computational and Theoretical Aspects*, Academic Press, New York (1978).

[10] **Reingold E.M., Nievergelt J., Deo N.,** *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1977).