Information Processing & Management, Vol. 12, pp. 377–384. Pergamon Press 1976. Printed in Great Britain

COMPRESSION OF LARGE INVERTED FILES WITH HYPERBOLIC TERM DISTRIBUTION

E. J. Schuegraf
Department of Mathematics, St. Francis Xavier University Antigonish, Nova Scotia, Canada

Abstract—The storage requirements for retrieval systems utilizing inverted files are calculated assuming different storage modes. Various methods for compression of these large files are analyzed. Binary vectors compressed by run-length coding as well as lists of document numbers were found to be suitable. The problem of minimal storage requirements for the inverted file is solved for different assumptions about index term distributions. A representation combining run-length coded binary vectors with list of document numbers was found to be the most economical. Parameter values for this minimum storage form are calculated and specified in tables as well as displayed graphically.

1. INTRODUCTION

Many interactive data base management systems as well as most automated library systems must respond quickly to user inquiries. This constraint presents substantial problems especially in the design of on-line systems accessing a large data base such as a library catalog file of the document file in a retrospective document retrieval system. Various solutions to achieve reasonable response time have been suggested, ranging from the use of special technology[1, 2] to file structures tailored to a particular application[3].

A common approach to satisfying the above mentioned constraint is to generate from the original data an auxiliary file whose only function is to provide quick response to queries. Unfortunately, the storage requirements nearly double, since the redundant material in the auxiliary file almost duplicates the original data. The main distinction between the original and the auxiliary file is the different structure imposed on the data. As noted by Cardenas[4] the organization of the auxiliary file "becomes another file problem in itself, possibly of the same magnitude as the data base itself".

In theory, it is possible to select the organization of the auxiliary file from the many possible methods described in the literature[5]. However, most systems use either the inverted file (TDMS, GIS) or the tree organization (IMS). Tree structured files are discussed by Sussenguth[6] and Stanfel[7], while inverted files are described by Cardenas[4, 8] and King[9]. A comparison of the efficiency of the two structures for document retrieval was made by Ein-Dor[10]. From these discussions in the literature the fact emerges that an inverted file performs better than a tree structured auxiliary file, especially with respect to complex queries. For the subsequent discussion it is assumed that an inverted file is used as the tool for answering queries.

2. METHODS FOR STORING INVERTED FILES

Two alternate interpretations of inverted files are possible, resulting in two different storage forms and manipulation methods. The first interpretation envisages the file as a store of lists of document numbers[11]. A list $L(k)$ consists of all the document, or accession numbers of all the records in the data base which contain the kth index element. The number of different lists D is determined by the size of the index set, since each element k has one associated list $L(k)$. If the database is comprised of N records, $\lceil \log_2 N$ bits are needed to represent one accession number. ($\lceil(x)$ denotes the smallest integer greater than x.) Problems arising from the distribution of index terms resulting in different list lengths have been discussed by Lowe[12]. The inverted index is usually stored as a sequence of accession numbers and pointers associated with each index term indicating the beginning of each list. Figure 1 shows such an arrangement.

If it is assumed that there are a total of I entries in the inverted file, the storage requirements in bits may be calculated as,

$$S_{\text{LIST}} = I * \lceil \log_2 N + D * \lceil \log_2 I \qquad (1)$$

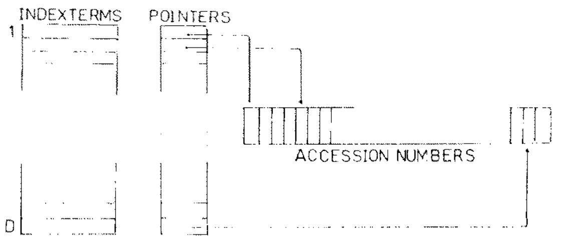

Fig. 1. Inverted file interpreted as a store of lists.

with the first term representing the storage for the accession numbers and the second term accounting for the pointers.

Coordinate indexing provides the second interpretation for inverted files. Assume that all index terms are ordered according to some criteria and can thus be numbered sequentially. For each document in the collection it is possible to construct a Boolean vector with N components. The ith component is true if index term i is contained in the document, and false, otherwise. Boolean values may be represented by the digits zero and one, and thus have a convenient representation for computer processing. There are several advantages to using binary vectors[9] as the basis for an inverted file. Most machines allow easy manipulation of bit strings and logical operations on these strings can be performed normally by single machine instructions. The answering of queries thus is quite rapid and requires little programming effort. Furthermore, all vectors are of the same length and the pointers required in the list interpretation of an inverted file can be eliminated and may be replaced by address calculations. The storage requirements for an inverted index based on binary vectors is given by

$$S_{\text{VECTOR}} = N * D. \tag{2}$$

In contrast to (1), the storage needed is independent of the number of index entries and is only a function of the number of documents N in the collection and the size D of the index set. The I non-zero entries in the index represent only a small fraction of the $N * D$ bits required and thus most of the $N * D$ bits are zero. Elimination or compression of the many zeros can reduce storage requirements.

The approach by THIEL and HEAPS[11] combines the list interpretation with the binary vector. If a few successively numbered records contain the same index term a bit map for these records is stored rather than the record numbers. Some storage overhead is introduced by the necessity for special codes indicating the storage mode used (binary vector/document numbers). The method is quite efficient for clustered document collections or documents which have been preordered. For inverted files based on the binary vector another method of compressing the zeros, called run-length coding may be borrowed from picture compression.

3. RUN-LENGTH CODING

The idea of run-length coding is fairly simple. A string of consecutive zeros terminated by a one (called a run) is replaced by the length of the run[13]. The main area of application has been image compression[14], but it has also been suggested for compression of bibliographic files[15, 25]. Various attempts have been made to optimize the code for the length of a run[16, 17]. For reasons of convenient computer manipulation a fixed number of bits b is often chosen to encode the length of a run. Assume that the average length of runs is r, then a value of $b = \lceil \log_2 r$ is the optimal assignment for b and produces minimum storage requirements. Thus, the only problem remaining in the choosing of b is the calculation of the average run length, $r(n, k)$, of a binary vector with n components containg k ones. There are $\binom{n}{k}$ such vectors with $n-k$ zeros and k ones. Assuming that each of these vectors occurs with the same probability, it is sufficient for the computation of r to determine the total number of runs, $R(n, k)$, in all these vectors. The value of R may then be calculated as

$$r(n,k) = \frac{\binom{n}{k}(n-k)}{r(n,k)} \tag{3}$$

$R(n,k)$ may be computed by the following arguments:

Let $S(n,k)$ be the set of all vectors with n components, consisting of $n-k$ zeros and k ones. An element of S is said to have property A if the bit in position n is a one. The set $S(n,k)$ may be divided into two mutually exclusive subsets, $S(n,k,A)$, containing all vectors with property A, and the complement $S(n,k,\bar{A})$. The number of different vectors in each set are given by

$$N(n,k,A) = \binom{n-1}{k-1} \text{ and } N(n,k,\bar{A}) = \binom{n-1}{k}.$$

Assume that $R(n-1,k)$ is already known and $R(n,k)$ is to be calculated. There are three possible cases of constructing a member of $S(n,k)$ (see Fig. 2).

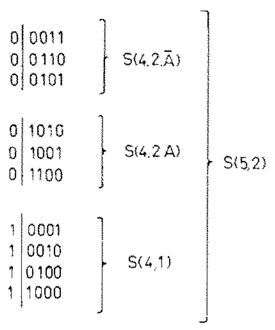

Fig. 2. Construction of S(5, 2).

(1) Take the elements of $S(n-1,k,A)$ and add a zero in position n. This increases the length of the runs but the number of runs remains the same.

(2) Take the elements of $S(n-1,k,A)$ and add a zero in position n. The additional zero adds a run to each element in addition to the number of runs in $S(n-1,k,A)$. The set $S(n-1,k-1,A)$ has $\binom{n-2}{k-1}$ elements and since one run is added per element the number of additional runs is $\binom{n-2}{k-1}$.

(3) The third case of constructing an element of $S(n,k)$ is by taking an element of $S(n-1,k-1)$ and adding a one in position n. The number of runs stay the same as in $S(n-1,k-1)$, given by $R(n-1,k-1)$.

All the elements of $S(n,k)$ are covered by these three cases and thus the number of runs, $R(n,k)$, is given by

$$R(n,k) = R(n-1,k-1) + R(n-1,k) + \binom{n-2}{k-1}. \tag{4}$$

The last two terms are the contribution from cases 1 and 2, whereas the first is the contribution of case 3. This recursion formula has its start given by the following obvious values:

$$R(j,j-1) = j$$

$$R(j,1) = 2(j-1). \tag{5}$$

Standard mathematical procedures give, after considerable manipulation, the following solution to the recursion:

$$R(n, k) = \binom{n-1}{k}(k+1), \tag{6}$$

a result which may be verified by substituting (6) into (4). Using (3) and (6), the average run-length may be computed as

$$r(n, k) = \frac{n}{k+1}, \tag{7}$$

which in turn yields for the number of bits $b(n, k)$ for encoding the runs

$$b(n, k) = \lceil \log_2 \frac{n}{k+1}. \tag{8}$$

The above result is optimal only for a single binary vector with n components of which there are k ones, and is optimal only for those entire inverted files, where each term is used to index exactly the same number of documents. As this assumption is not very realistic, eqn (8) may not specify the optimal value for the inverted file as a whole. For optimization of the total storage the vectors associated with all the index terms should be considered, since the number of non-zero components in the vectors may vary widely.

4. HYPERBOLIC TERM DISTRIBUTIONS AND RUN-LENGTH CODING

Various studies and experiments have been conducted to determine the empirical distribution of index terms, while many attempts have been made at resolving the question of the most optimal distribution for retrieval[18–21]. As in other bibliometric areas[22], a distribution commonly referred to as Zipf's Law[23], is known to fit most of the observed data for index term distributions.

Zipf's Law implies, that if the D index terms are ranked in order of decreasing frequency, then the frequency f of a term of rank r is given by

$$f(r) = C/r. \tag{9}$$

The constant C, the only parameter in the distribution, may be computed using a standard series approximation, as

$$C = \frac{I}{\log_e D + 0.577}$$

with I being the total number of entries in the index. The knowledge of the term distribution may be utilized to find the most economic storage mode for the entire index, since eqn (9) specifies the number of non-zero entries in the N-component binary vector associated with a term of rank r. If the index is stored as lists of document numbers, then (9) may be interpreted as the length of the lists.

The choice of representation for the inverted file under the assumption of a hyperbolic distribution appears difficult. For terms of high frequency the binary vector seems to be more economical, especially if run-length coding is used for compression. On the other hand, document numbers appear more suitable for low frequency terms. An uncompressed bit vector need not be considered since even for the most frequent terms run-length coding is more economical. The frequency K of the term, for which run-length coding and bit vector representation use the same amount of storage, can be determined by solving the equation

$$N = (K+1)\log_2\left(\frac{N}{K+1}\right), \tag{10}$$

for K. However, this equation has no solution since the right side of eqn (10) has a maximum at $K = (N/e) - 1$ with a value of $(N/e \log 2)$. Therefore, uncompressed bit vectors can be eliminated, and only run-length coded bit vectors and document numbers remain for consideration.

If it is assumed that a combination of the two representations is to be the most economical, then there must be a certain rank, say r_0, at which run-length coded representation is changed to document numbers. The storage requirement for the entire index, assuming Zipf's distribution for the index terms, can thus be expressed as,

$$S_C = \sum_{r=1}^{r_0} \left(\frac{C}{r}+1\right) \lceil \log_2 \frac{N}{\frac{C}{r}+1} + \sum_{r=r_0+1}^{D} \frac{C}{r} \lceil \log_2 N, \tag{11}$$

with the first sum expressing the contribution from run-length coded vectors, while the second term results from the lists of document numbers. To determine the optimal value for S_C it is sufficient to find the minimum of S_C with respect to r_0. Straightforward computations lead to the following equation for r_0

$$r_0 = \frac{C \log N}{\left(1+\frac{C}{r_0}\right) \log \frac{N r_0}{C}} - 1$$

which has no analytical solution but may be solved numerically. Tables 1 and 2 show r_0, for typical values of C and N, while Figs. 3 and 4 are graphic representations.

The results show the expected behavior, that the rank r_0 at which the mode of representation is to be switched decreases as the document collection increases. The rank r_0 also increases when the Zipf constant C increases, since this implies normally a large vocabulary and more entries in the index.

Having determined the optimal rank r_0 it is still necessary to find the number of bits to be used for representing a run. For computers with fixed word length it is easier to represent a run by a code of fixed length rather than by a variable length code. The advantage of fixed length codes over variable length codes is discussed in detail by SCHUEGRAF and HEAPS[24].

Minimization of wasted storage leads to the following formula for the number of bits b to be used for coding a run:

$$b = \frac{\sum_{r=1}^{r_0} \left(\frac{C}{r}+1\right) \log_2 \frac{N}{(c/r)+1}}{\sum_{r=1}^{r_0} \left(\frac{C}{r}+1\right)}$$

Table 1. Values for r_0, b, R, as a function of Zipf's constant C

	N = 100,000			N = 1,000,000		
ZIPF C	r_0	b	R	r_0	b	R
10,000	1773	8.43	0.707	1561	11.64	0.750
30,000	5314	7.63	0.641	4673	10.85	0.695
50,000	8855	7.26	0.603	7785	10.48	0.654
80,000	14,165	6.92	0.537	12,454	10.14	0.602
100,000	17,706	6.76	0.516	15,566	9.98	0.588

Table 2. Values of r_0, b, R, as a function of the number of documents N

	C = 10,000			C = 100,000		
N	r_0	b	R	r_0	b	R
10,000	2069	5.23	0.642	—	—	—
50,000	1852	7.46	0.686	—	—	—
100,000	1773	8.43	0.707	17,706	6.76	0.516
500,000	1618	10.67	0.735	16,145	9.01	0.565
1,000,000	1560	11.64	0.750	15,566	9.98	0.588
10,000,000	1398	14.88	0.780	13,937	13.21	0.675

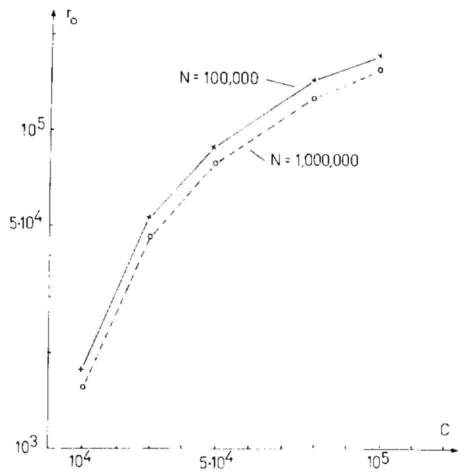

Fig. 3.

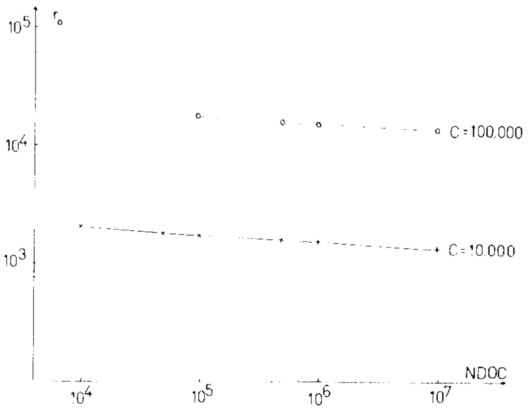

Fig. 4.

Fig. 3. r_0 as a function of C.

Fig. 4. r_0 as a function of N.

which may be expressed as

$$b = \frac{1}{\log 2}\left\{\log\frac{N}{C} + \frac{\frac{\log^2 r_0}{2} + r_0 \cdot C \cdot [\log r_0 - 1] + C}{r_0 + C \cdot (\log r_0 + 0.577)}\right\}. \qquad (12)$$

Values for b are also contained in Tables 1 and 2, as functions of C and N, with Figs. 5 and 6 showing the graphic representation. As anticipated, the code length for a run increases with the number of documents, but decreases with vocabulary growth as indicated by large values of C.

Since the code length b must be an integer the values given in Tables 1 and 2 should be rounded to the next highest integer. However, practical considerations may force other values of b, because of the fixed character or word length of the computer to be used. For example, a choice of 8- or 12-bits for the code length of a run would be quite suitable for computers with an 8-bit character representation, since all computers have special instructions for processing of characters. If this approach is chosen, storage space is sacrificed for convenient programming and easy processing.

The advantage of the combined run-length coded binary vector and document number representation over the two other storage modes becomes apparent when the compression ratio R is calculated. The compression is defined as the ratio of the length of the compressed file to that

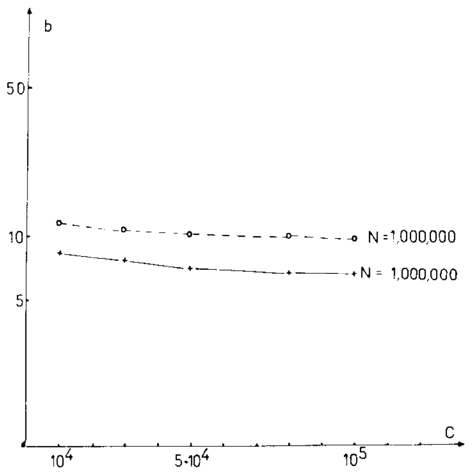

Fig. 5. b as a function of C.

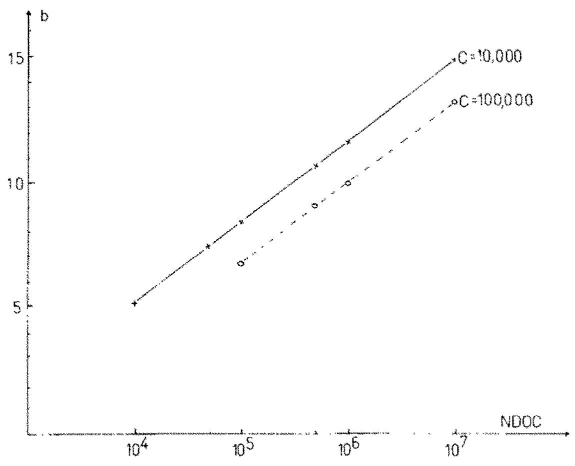

Fig. 6. b as a function of N.

of the uncompressed file. Since the storage space occupied by the list representation is always less than that of uncompressed bit vectors, the former storage mode is used in the calculation of the compression. It is defined as

$$R = S_C / S_{\text{LIST}}$$

with S_{LIST} from eqn (1) and S_C from eqn (11). Neglecting the contribution from the pointers in (1) allows to compute R as a simple expression:

$$R = 1 - \left(1 - \frac{b}{\lceil \log_2 N}\right) \frac{\log r_0}{\log D} \tag{13}$$

Tables 1 and 2 contain values for R which were calculated by rounding the proper values for b to the nearest integer and using eqn (13). Considerable savings in storage space, between 30 and 40%, may be realized and are especially significant for inverted files containing many documents and a large vocabulary.

5. CONCLUSION

The analysis of storage methods for inverted files in the previous sections has produced two interesting results. It has been shown that under certain conditions a binary vector compressed by run-length coding will occupy less space than the corresponding list of document numbers. The conditions are satisfied if there are k non-zero entries in the n-component binary vector and the inequality

$$(k+1)\lceil \log_2 \left(\frac{N}{k+1}\right) < k \lceil \log_2 N$$

holds. Unfortunately, real systems rarely exhibit the property that every index term has the same number of postings, so that the above conditions are almost never satisfied for an entire file.

Recently, it was proposed to use equifrequent fragments rather than words as index elements for a retrieval system[24–26, 29]. In this case, the above conditions would be fulfilled and eqn (8) would specify the best choice for the code length of a run. The storage space for the entire inverted file would be optimized by choosing that particular value.

Because most systems do not exhibit the equifrequency property, a more realistic assumption about index term distribution was made. A hyperbolic distribution, the Zipf Law, was adopted, since it is the one most often observed in real systems. Assuming that distribution, the problem of minimizing the storage space for the entire file was solved by a combination of run-length coded binary vectors and lists of document numbers. Some values of the two parameters controlling this representation were calculated and specified in two tables. A choice of these values will result in a minimum value for the storage requirements. The relative insensitivity of the values for the storage space with respect to the parameters controlling the index term distribution allows immediate application to practical systems.

The implementation of a combined run-length coded vector and document number inverted file appears not to be difficult, since similar ideas have been already explored[9, 11]. The combined approach promises considerable economic benefits, a very important fact, since very few other methods for the compression of general binary matrices exist.

It has been shown[27], that even the use of unambiguous bit matrices for compression of large binary matrices is infeasible, although convenient retrieval algorithms based on this storage form exist[28]. As an alternative to the use of unambiguous bit matrices for the compression of binary matrices, such as inverted files, the feasibility of using multiple projections for compression is presently being investigated. Practical problems regarding retrieval and dynamic updating of inverted files in various storage modes are also being considered.

Acknowledgements—This research was supported by the National Research Council of Canada and the University Council for Research of St. Francis Xavier University. The many helpful discussions with Dr. S. K. AALTO are also gratefully acknowledged.

REFERENCES

[1] J. L. PARKER, A logic per track retrieval system. *Proceedings IFIP Congress*, pp. 711–716. North Holland, Amsterdam (1971).
[2] B. PARHAMI, A highly parallel computing system for information retrieval. *Proceedings Fall Joint Computer Conference*, pp. 681–690 (1972).
[3] J. J. DIMSDALE and H. S. HEAPS, File structure for an on-line catalog of one million titles. *J. Lib. Autom.* 1973, **6**, 37.
[4] A. F. CARDENAS, Analysis and performance of inverted data base structures. *Commun. ACM* 1975 **18**(5), 253.
[5] D. LEFKOVITZ, *File Structures for On-Line Systems*. Spartan Books, New York (1969).
[6] E. H. SUSSENGUTH, Use of tree structures for processing files. *Commun. ACM* 1963, **6**(5), 272.
[7] L. E. STANFEL, Three structures for optimal searching. *J. ACM* 1970, **17**(3) 508.
[8] A. F. CARDENAS, Evaluation and selection of file organization—a model and system. *Comm. ACM* 1973, **16**(9) 540.
[9] D. R. KING, The binary vector as the basis of an inverted index file. *J. Lib. Autom.* 1974, **7**(4), 307.
[10] P. EIN-DOR, The comparative efficiency of two dictionary structures for document retrieval. *INFOR* 1974, **12**(1), 87.
[11] L. H. THIEL and H. S. HEAPS, Program design for retrospective searches on large data bases. *Inform. Stor. Retr.* 1972, **8**, 1.
[12] T. C. LOWE, The influence of data base characteristics and usage on direct access file organizations. *JACM* 1968, **15**, 535.
[13] P. ELIAS, Predictive coding. *IRE Trans. Inform. Theory IT*-1, 1955, **16**.
[14] H. KOBAYASHI and L. R. BAHL, Image data compression by predictive coding. *IBM J. Res. Devel.* 1974, **18**, 164.
[15] M. F. LYNCH, Compression of bibliographic files using an adaptation of run-length coding. *Inform. Stor. Retr.* 1973, **9**, 207.
[16] S. W. GOLOMB, Run-length encoding. *IEEE Trans. Inform. Theory* **IT-12**, 1966, **IT-12**, 399.
[17] S. D. BRADLEY, Optimizing a scheme for run-length encoding, *Proc. IEEE* 1969, **57**, 108.
[18] N. HOUSTON and E. WALL, The distribution of term usage in manipulative indexes. *Am. Docum.* 1964, **15**, 105.
[19] G. A. SALTON, Computer evaluation of indexing and text processing. *JACM* 1968, **15**, 8.
[20] P. ZUNDE and V. SLAMECKA, Distribution of indexing terms for maximum efficiency of information transmission, *Am. Docum.* 1967, **18**, 104.
[21] E. SVENONIOUS, An experiment in index term frequency. *J. ASIS* 1972, **23**, 109.
[22] R. A. FAIRTHORNE, Empirical hyperbolic distributions (Bradford-Zipf-Mandelbrot) for bibliometric description and prediction. *J. Docum.* 1969, **25**, 319.
[23] G. K. ZIPF, *Human Behaviour and the Principle of Least Effort*. Addison Wesley, New York (1949).
[24] E. J. SCHUEGRAF and H. S. HEAPS, Selection of equifrequent word fragments for information retrieval. *Inform. Stor. Retr.* 1973, **9**, 697.
[25] I. J. BARTON, S. E. CREASEY, M. F. LYNCH and F. F. SNELL, An information theoretic approach to text searching in direct access systems. *Commun. ACM* 1974, **12**, 345.
[26] E. J. SCHUEGRAF and H. S. HEAPS, Query processing in a retrospective document retrieval system that uses word fragments as language elements. *Information Proc. Mgmt.* 1976, **12**, 283.
[27] S. K. AALTO and E. J. SCHUEGRAF, A. determination of the number of unambiguous bit matrices. *Int. J. Comp. Inform. Sci.* to be published.
[28] Y. R. WANG, Data retrieval algorithm for unambiguous bit matrices. Paper presented at the *7th Annual Princeton Conf. on Inform. Sci. and Systems*, 1973, 22–23 March.
[29] A. C. CLARE, E. M. COOK and M. F. LYNCH, The identification of variable length equifrequent character strings in a natural language data base. *Comp. J.* 1972, **15**, 259.