ELEMENTS OF THE RANDOMIZED COMBINATORIAL FILE STRUCTURE

Richard A. Gustafson*
University of South Carolina, Columbia, South Carolina

ABSTRACT

A file structure designed to provide rapid, random access with minimum storage overhead is presented. Storage and retrieval are achieved by direct attribute combination-to-address transformation thereby negating the necessity for large file dictionaries or list-pointer structures. The attribute combination-to-address transformation is conceptually similar to key-to-address transformation techniques, but the transformation is not limited to operations on a single key but operates on the combination of several independent keys (or any subset of the combination) describing an item or request.

A storage and retrieval system utilizing the combinatorial file structure is developed. Storage and retrieval results derived from a simulated document library of 4000 items are presented. The new file organization is shown to have marked value with respect to minimum storage overhead and high retrieval speed.

KEY WORDS AND PHRASES

file structures, file organization, combinatorial file, key-to-address transformation, multiple attribute retrieval, file access method, data structures, hash-coding, attribute combination-to-address transformation, storage and retrieval system

I. Introduction

The nucleus of any information handling system is the file structure. All data in an information handling system flow from information sources into a structured file (collection of items of information) and ultimately to the information user. Effective management of this information flow has led to the application of mechanized, and in particular computer based, systems to alleviate the problems of information storage and retrieval. This application of automated information handling systems has placed a premium on the utilization of effective and efficient file structures. File effectiveness and efficiency may be measured with respect to the amount of storage required to store an information file and the speed with which information may be retrieved from the file. As one might suspect, storage efficiency and retrieval speed are normally at odds with one another.

Multiple-key file structures, presently in use with mechanized systems, range from simple unordered, linear files to sophisticated list and inverted files (4, 8, 9, 18, 23, 27). These files can be said to be typical and illustrate the diversity of storage requirements and retrieval speeds of the various contemporary file structures. On one hand is the linear file requiring storage only for the items and descriptor keys but necessitating a linear search of the entire file for any retrieval; while, on the other hand, are the inverted and threaded list files achieving rapid retrieval but requiring the storage overhead of indexes to the files in the form of dictionaries or pointers. This storage overhead can, in fact, become as large or larger, storage-wise, than the item file itself (8, 14).

The retrieval speed of the inverted and list files can, under certain conditions, also be disappointing (12, 14, 25). This is a result of the "term at a time" search of the inverted and list files which does not fully utilize the combinatorial properties of a multiple-key file. Ghosh and Lum (12, 25) have proposed file organizations designed to exploit key-combination retrieval but they require additional key-combination indexes and duplicate item pointer entries to accommodate all key combinations.

It would be ideal if the index, as a search mechanism, could be replaced by an algorithm (i.e., a short series of computation steps) which would operate on a request and produce the accession numbers of the items in the file relevant to the request. Such a search mechanism would be required to accept multiple-key requests (i.e., multiple index terms) and produce retrievals based upon any combination of terms. The remaining sections of this paper develop and present a search mechanism and file structure possessing these properties.

II. A Combinatorial Structure

The combinatorial properties of the storage and retrieval mechanisms can be easily illustrated graphically. With the assumption that each item can be described by three unique attributes taken from a descriptor set of five attributes, the num-

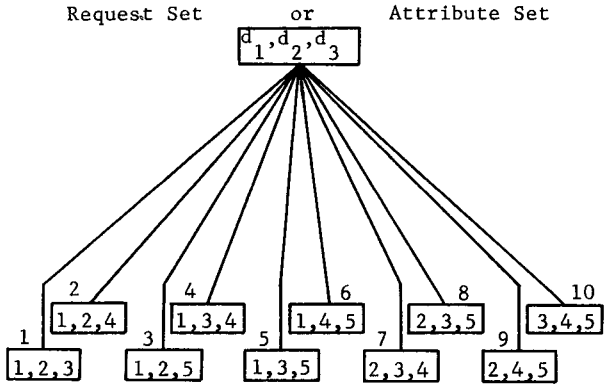
*Present Address: Air Force Technical Applications Center, Alexandria, Virginia

ber of possible attribute combinations can be calculated from the binomial coefficient:

$$\binom{N}{M} = \binom{5}{3} = \frac{5!}{3!(5-3)!} = 10$$

It can therefore be concluded that, in this example, any set of attributes describing an item must be one of ten possible combinations.

Representing the five possible attributes in the descriptor set by the numbers 1 through 5 yields the ten possible combinations illustrated in the following diagram.



Total Possible Combinations

From inspection of the above diagram it is seen that any possible set of three unique descriptors must map into one and only one of the possible combination "buckets".

$$g: \quad \{d_1,d_2,d_3\} \longrightarrow a: \quad a \in \{1,2,3,4\ldots10\}$$

The term "bucket" is used as it is possible that two or more items could have the same attribute set; hence the word "bucket" is used to indicate the possibility of more than one item in a unique combination. The mapping, g, must produce an address, a, which is an element of the set $\{1,2,\ldots10\}$.

The Possible Combinations diagram allows the consideration of both storage and retrieval operations. The storage mechanism may be illustrated by means of the following example.

Item Storage

Item i	Attribute Set d_1,d_2,d_3	Mapped to Bucket a
1	2,3,5	8
2	2,3,4	7
3	1,2,3	1
4	1,3,4	4
5	1,3,5	5
6	3,4,5	10

The retrieval mechanism operates in a similar manner; however, all combinations containing the request attribute set must be addressed. The following Item(s) Retrieval example illustrates an obvious but important property of a combinatorial file structure. A particular request need only address some fraction of the combination buckets. The number of addressed buckets is a function of the number of specified attributes, L, where L must be greater than or equal to 1, and less than or equal to the number of attributes used to describe an item, M. M in this case is three.

Item(s) Retrieval

Number of Attributes in Request	Request Set	Possible Combinations	Mapped to Bucket
L	$d_1\ldots d_L$	d_1,d_2,d_3	a
3	1,3,4	1,3,4	4
2	1,3	1,2,3	1
		1,3,4	4
		1,3,5	5
1	3	1,2,3	1
		1,3,4	4
		1,3,5	5
		2,3,4	7
		2,3,5	8
		3,4,5	10

The number of addressed buckets may be determined from the binomial coefficient:

$$(1) \quad Q_{N,M,L} = \binom{N-L}{M-L} = \frac{(N-L)!}{(M-L)!\,(N-M)!} = \frac{(N-L)_{M-L}}{(M-L)!}$$

Where:

M - number of attributes/item
N - number of possible attributes
L - number of specified request attributes ($1 \le L \le M$)

$$(N-L)_{M-L} = (N-L) \cdot (N-L-1) \ldots (N-M+1)$$

$Q_{N,M,L}$ - number of possible combinations for L specified attributes with fixed N and M

The fraction of combination buckets addressed by a request set consisting of L attributes will be termed the selection ratio. The selection ratio is defined by:

$$(2) \quad S.R._{N,M,L} = \frac{\text{Number of possible combinations for L specified attributes.}}{\text{Number of possible combinations for M attributes.}}$$

$$\frac{\binom{N-L}{M-L}}{\binom{N}{M}} = \frac{(M)_L}{(N)_L}$$

Where:

$$(M)_L = M \cdot (M-1) \ldots (M-L+1)$$
$$(N)_L = N \cdot (N-1) \ldots (N-L+1)$$

Equation (2) yields the simple but interesting result that the worse case selection ratio, L = 1, is:

$$S.R._{N,M,1} = \frac{M}{N}$$

Application of equations (1) and (2) to the case of five possible attributes with three attributes per item with requests of one, two, and three attributes yields respectively:

$$N = 5;\ M = 3;\ \binom{N}{M} = 10$$

L	$Q_{5,3,L}$	$SR_{5,3,L}$
1	6	3/5
2	3	3/10
3	1	1/10

which is in complete agreement with the figures deduced from the possible combinations diagram for the same case.

Combinatorial Mapping

It would appear that the combinatorial structure offers the ideal mechanism of storage and retrieval. That is to say, a request defines a volume of the address space within which all stored items are pertinent to the specified request and outside of which the items are irrelevant to the request. The combinatorial structure does, however, present an addressing or mapping problem.

The required mapping is identical to the desired mapping for the combinatorial space as presented in conjunction with the Possible Combinations diagram.

$$(3)\quad g: \left\{d_1, d_2 \ldots d_M\right\} \longrightarrow a: \quad a \in \left\{1, 2 \ldots P\right\}$$

Where:

$$P = \binom{N}{M}$$ (Number of possible combinations in the combinatorial address space)

The combinatorial mapping g, must collapse M indices or attributes into a single index, the bucket address.

The mapping g, may be developed in a fairly straightforward manner by first considering how all possible combinations of N objects taken M at a time without replications and with unordered samples may be formed. There are, of course, $(N)_M/M!$ possible combinations. These combinations, each consisting of M indices (attributes), may be generated by varying the indices in the following manner:

$$(4)\quad \left\{d_{i_1}, d_{i_2} \ldots d_{i_M}\right\}$$

$$i_1 = M, M+L \ldots N$$
$$i_2 = M-1, M, \ldots (i_1 - 1)$$
$$\vdots$$
$$i_M = 1, 2 \ldots (i_{M-1} - 1)$$

Where d_j is the j^{th} element of the object set or, in this case, the j^{th} attribute or index in the descriptor set. If the M^{th} index in the attribute set, $\left\{d_{i_1}, d_{i_2} \ldots d_{i_M}\right\}$, is varied most rapidly, and the $(M-1)^{th}$ varied next, through the first index varying least rapidly, all possible combinations will be produced in a fixed order (thereby effectively deleting permutations in any combination) and no replications in any combination will occur.

Since all possible attribute sets can be generated in a fixed and mathematically predictable manner, it would appear that the location or position of a particular attribute set in the array of possible attribute sets could be determined. Indeed, this is the case (14, 24).

To determine the location of a specified attribute set in the array of possible attribute sets as produced by the possible-combinations generator, one may answer either of two questions. How many combinations are below the specified set; or alternately, how many are above? The former question will be answered as the development is straightforward.

Given an attribute set consisting of M elements ranked from highest to lowest by numeric value, $\left\{d'_1, d'_2 \ldots d'_M\right\}$, it is certainly true that all attribute combinations formed by taking M attributes at a time from a set of (d'_1-1) attributes (note that d'_1 is the largest value in the attribute set) must be below the given set. Likewise, it may be stated that all combinations formed by fixing d'_1 in each combination and taking (M-1) attributes from a set of (d'_2-1) attributes to complete the combinations must also be below the given attribute set in the array of possible-combination sets. This argument may be extended to the d'_M element where $d'_1, d'_2 \ldots d'_{M-1}$ are all fixed in each combination and the combinations are completed by taking M-(M-1) or 1 attribute from a possible set of (d'_M-1) attributes.

The number of ways M attributes may be taken from a set of (d'_1-1) attributes without replication and ignoring order is simply:

$$\binom{d'_1-1}{M}$$, a binomial coefficient.

It may therefore be concluded that the number of possible combinations below a specified attribute set is:

$$\binom{d'_1-1}{M} + \binom{d'_2-1}{M-1} + \ldots \binom{d'_M-1}{1}$$

where: $d'_1 > d'_2 > d'_3 \ldots > d'_M$. Adding one to the number of combinations below a specified attribute set yields the location of that attribute set in the array of possible combinations as produced by the possible-combinations generator. This location is the "bucket address" in the address space. The mapping, g, is therefore:

$$(5)\quad g: \left\{d'_1 d'_2 \ldots d'_M\right\} \longrightarrow a: \quad a \in \left\{1,2,3 \ldots P\right\}$$

$$\text{where: } P = \binom{N}{M}$$

$$g(d'_1, d'_2 \ldots d'_M) = a = \binom{d'_1 - 1}{M} + \binom{d'_2 - 1}{M-1} + \ldots \binom{d'_M - 1}{1} + 1$$

where the primes on the d's simply indicate ordering from highest to lowest.

Although the preceding development of the mapping, g, might appear overly restrictive, it is in fact quite general. With the stipulation that the descriptor set of possible attributes is reduced to the subscripts of the actual attributes ranging from 1 to N, the mapping as defined by (5) is perfectly correct.

Retrieval Operation

While the preceding development of the attribute set to address mapping defines the storage mechanism, a retrieval operation does not necessarily involve a single mapping between the request attribute set and the address space. This property of the retrieval mechanism was discussed earlier and was illustrated by equation (1) which states that Q mappings are required where:

$$Q = \frac{(N-L)_{M-L}}{(M-L)!}$$

It is to be recalled that N and M are, respectively, the number of possible attributes and the number of attributes per item. L is the number of attributes in a request ($1 \leq L \leq M$).

As a request must search the Q buckets contained in the fraction of the volume of the address space as defined by the request, one method of mapping to these buckets would be to generate all possible combinations of attribute sets containing the request attributes and map to the address space one to one for each possible combination. As this technique offers conceptual simplicity, it will be pursued.

An obvious starting point in the development of a retrieval possible-combinations generator is the total possible-combinations generator as previously presented in (4). With the consideration of (4), the retrieval-combinations problem reduces to the generation of all possible combinations of (N-L) things taken (M-L) at a time. This may be accomplished by defining a modified descriptor set for each request consisting of the original descriptor set with the L request attributes deleted. The Q combinations then formed by the combinations generator yield all possible combinations of M attributes relevant to the request when the L original request attributes are included in each combination. These attribute combinations may then be mapped to the Q buckets by the mapping, g. This mapping operation is:

$$(6)\quad g: \left\{d'_{i_1}, d'_{i_2} \ldots d'_{i_M}\right\} \longrightarrow a_i: \quad a_i \in \left\{1,2 \ldots P\right\}$$

$$\text{Where: } P = \binom{N}{M}; \quad i = 1,2 \ldots Q$$

$$\text{and: } Q = \binom{N-L}{M-L}$$

Examination and retrieval of the bucket items complete the retrieval operation.

In that the combinatorial file structure stores and retrieves each item as a function of its several attributes without resorting to complex list structures, the combinatorial structure closely approximates associative memory processing by "software" address decoding (7, 21). This characteristic is basic to all information storage and retrieval systems but in this case, is exhibited on the file structuring level.

III. A Randomized Combinatorial Structure

Although the combinatorial file structure may be pleasing from the standpoint of a storage and retrieval mathematical model, it has limited application in its present form. In order to preserve associations among items and to assign each item in the file a unique address, all possible combinations of descriptors, subject to a priori constraints, could be required to map into unique addresses. Should this requirement be levied, the preceding development of the combinatorial file structure is of little use; as, for any moderately large number of possible descriptors with more than a few attributes per item, the number of possible combinations becomes far too large for any practical application. This conclusion is substantiated by consideration of the number of combinations, 7.5×10^7, of only one-hundred descriptors taken five at a time.

The "possible combinations" problem is not unique to the combinatorial structure, but is encountered in most information storage systems. The classical example is that of using names for record (the basic information unit, e.g., payroll entry, account entry, etc.) identification. As the number of possible names is far greater than the normal record file, techniques for mapping the names to unique locations in the storage media have been developed (16, 22, 28, 31, 34, 35). One of the more popular and efficient methods of accomplishing this mapping is to randomize the record "names" and normalize the random codes to the file length. Each record may then be stored in its randomized location. The "name" randomization may be accomplished by performing arithmetic operations on the internal machine code used to represent the name, i.e., hash-coding.

Although the techniques of randomized file

addressing are varied and the associated problems interesting, of principle interest is the application of the randomizing concept to the combinatorial file structure. Whereas randomized file addressing is concerned with the one to one mapping of a record identification name to a storage address, the combinatorial structure requires not only that a combination of attributes, which forms a record identification, map to a storage address but, additionally, that any subset of the attributes forming the identification maps one to Q where Q defines the number of mapped addresses. It is therefore required that any technique of randomizing aimed at reducing the number of possible combinations retain the combinatorial mapping properties of the combinatorial structure (i.e., attribute subset mappings of one to Q must not be sacrificed by randomization).

Examination of the binomial coefficient defining the number of possible combinations which may be formed from N attributes taken M at a time yields an insight into possible techniques to alleviate the "possible combinations" problem. It is readily apparent that the reduction of either N, the number of possible attributes, or M, the number of attributes per sample, will reduce the number of possible combinations (this is true with the restriction that N> 2M). The randomization of the M attributes forming an item description, however, would annul the independence of the M attributes and would therefore invalidate a combinatorial mapping scheme when retrieval based upon a partial description (L out of M attributes) is indicated. Although the randomization of attributes on an item description level must be ruled out as combinatorial properties are destroyed, the randomization of the descriptor universe consisting of all N possible attributes is possible while still retaining the combinatorial structure.

Randomization Technique

The randomization technique consists of mapping a subset of the elements of a set D onto the elements of a set C such that the number of elements in set C is less than and normally much less than the number of elements in set D. Or in set notation

$$f: \{d_1, d_2 \ldots d_M\} \rightarrow \{c_1, c_2 \ldots c_M\}$$

where $\{d_1, d_2 \ldots d_M\}$ is a subset of length M of set D being mapped into M elements, $\{c_1, c_2 \ldots c_M\}$, of set C. Normally, the descriptor set, D, of all possible attributes consists of word descriptions in English while the randomized descriptor set, C, may be of any arbitrary representation. It may therefore be concluded that with an a priori knowledge of the representation of set D, a mapping, f, may be devised which, as a result of its operation on an element of set D, produces an element of set C in a convenient representation. The statement "in a convenient representation" is stressed as the combinatorial mapping as developed in the previous chapter operates on the subscripts of the attributes. Naturally, it would be very "convenient" to require that the randomized descriptor set, C, consist of subscripts (integers ranging from 1 to N).

The mapping necessary to achieve randomization of M attributes describing an item is, therefore, in set notation:

$$(7) \quad f: \{d_1 d_2 \ldots d_M\} \rightarrow \{c_1, c_2 \ldots c_M\}: \quad c_i \in \{1, 2 \ldots N\}$$

The randomized code set, C, may be of any length, N, necessary to reduce the possible combinations to manageable levels and the mapping is determined by the digital storage device characteristics.

Duplicate Code Probability

Some immediate consequences of the randomization mapping are apparent. In that any possible descriptor must map to one of the integer elements of the randomized code set and all storage and retrieval operations use these mapped integers, the randomized combinatorial structure is functionally independent of the file descriptor universe. Another, perhaps detrimental, aspect of the randomization of the descriptors is the possibility of two or more descriptors mapping to the same randomized code. This possibility of duplicate codes requires a modification to the combinatorial structure.

The previous section developed the combinatorial mapping structure based upon the assumption of M unique attributes describing an item. This assumption appeared valid in that the duplication of an attribute does not add additional information to an item description. The randomization mapping, however, negates the unique attribute assumption.

In order to better evaluate the impact of possible duplicate randomized codes on the combinatorial structure, it is necessary to form a more definitive measure of this duplication than simply the word "possible". With the assumption that the randomization function produces a truly random mapping of the attribute set of length M, resulting in a randomized code set of length M, a measure of the "degree" of code duplication reduces to finding the probability of obtaining J unique codes in a sample of M codes formed by drawing M codes from a universe of N distinguishable codes with replacement. This probability determination is similar to the classical "birthday problem"; however, it is necessary not only to determine the probability that no two codes in the sample are the same (M unique codes), but also to determine the probability of M-1, M-2...1 unique codes in a sample of M codes. This information is necessary in that each duplication of a code effectively reduces the number of attributes describing an item by one. The degree of this information loss is determined by the following (10, 14).

The probability that in a sample formed by drawing M items from N items with replacement, any of the N items may appear K_1 times while the remaining N-1 items appear $K_2, K_3 \ldots K_N$ times is:

$$(8) \qquad P = \frac{\frac{M!}{K_1!K_2!\ldots K_N!} \cdot \frac{N!}{l_1!l_2!\ldots l_M!}}{N^M}$$

where:

M - number of items in sample

N - total number of items

K_i - total number of times that the i^{th} item appears in the sample

$(K_1 + K_2 \ldots K_N = M)$

l_i - total number of times that the i^{th} digit appears in the set of K's.

$(0 \leq i^{th}$ digit $\leq M$: i^{th} digit is integer)

$(l_1 + l_2 \ldots l_M = N)$.

The probability of finding M unique items in a sample of M items is therefore:

$$P = \frac{\frac{M!}{1!1!\ldots1!0!\ldots0!} \cdot \frac{N!}{M!(N-M)!0!}}{N^M}$$

$$= \frac{N\cdot(N-1)\ldots(N-M+1)}{N^M} = \frac{(N)_M}{N^M}$$

as: $K_1, K_2 \ldots K_M = 1$ and $K_{M+1}, K_{M+2} \ldots K_N = 0$, and

$l_1 = M$, $l_2 = N-M$ and $l_3 l_4 \ldots l_M = 0$,

which reduces to the expected solution of the "birthday problem".

In that the probability of J unique items appearing in a sample of size M is not always determined by a single combination of duplications (the K set), all possible combinations of duplications resulting in a specified number of unique items must be listed and their probabilities evaluated and summed to yield the probability of J unique items in a random sample. The next section will contain an application of formula (8) which will further illustrate all "possible duplication combinations".

Duplicate Code Handling

Because of the possibility of duplicate attribute codes arising due to the randomization mapping of (7), it is necessary to devise some method capable of handling duplicate attribute codes in the context of combinatorial mapping. One possible solution to this duplication problem would be to redevelop the combinatorial structure allowing the duplication of attributes. In view of the preceding development, this would not be particularly difficult; however, a simple fact illustrates the futility of a combinatorial structure with the possibility of duplicate attributes: all combinations in a duplicate attribute combinatorial structure are not equally likely.

Another, more attractive, solution to the duplicate attributes problem would be to ignore duplicate attributes and, for the purposes of the storage mapping, generate at random non-duplicate attribute codes to replace any duplicates. Although this obviously constitutes a loss of retrieval efficiency, degradation occurs only at the search level as, although a set of requestors could possibly encompass a large area of the file, the requestors would, of course, define the area into which the original attribute set had mapped. It is to be again noted that the duplication of attributes is a probabilistic function as defined by (8) and the action of randomizing duplicates would be taken only on occasion. Elaboration on the probabilistic nature of randomized attribute duplication will be given in the next section.

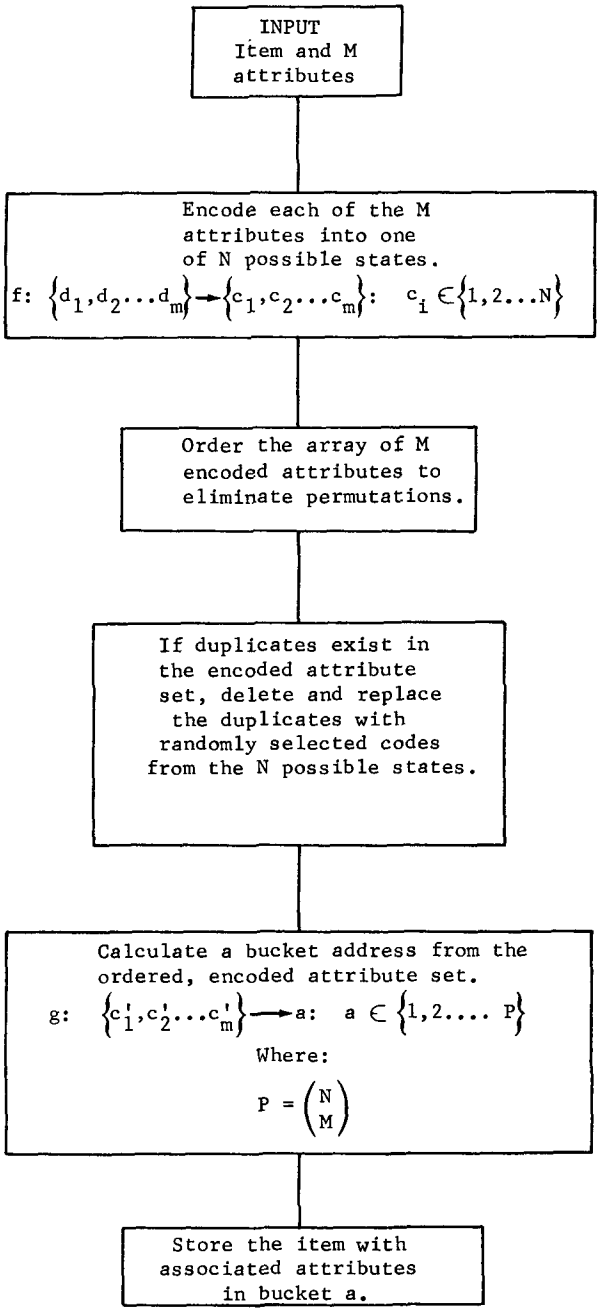
The random replacement of duplicate attribute codes as well as the normal randomization of the original attributes necessitates a search for original descriptor/requestor attribute matches subsequent to bucket address decoding during retrieval operation. This search necessity is a result of the attribute randomization phase (encoding) where mapping of original attributes is many to one. This simply means that a mapped bucket address may or may not contain the original requestor attributes, hence, the need for the search operation. It is hastened to indicate that all unmapped buckets will not contain the request attributes. The retrieval operation may therefore be perhaps best described as a "divide and conquer" technique.

IV. File Design and Application

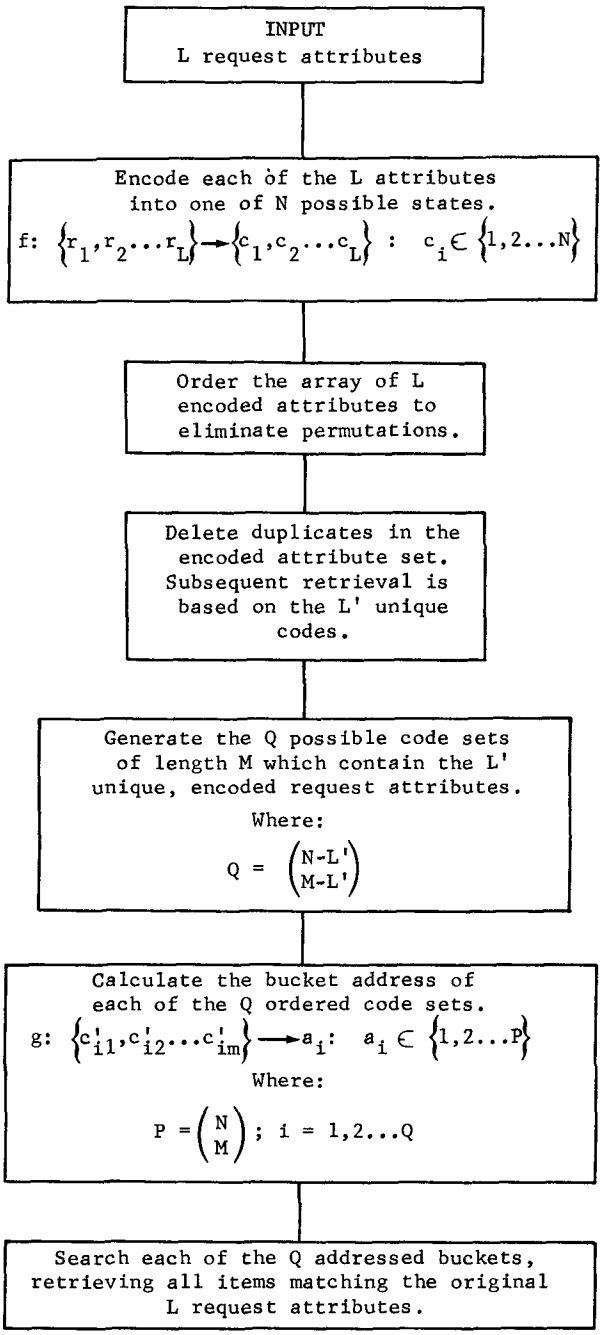
In order to demonstrate the feasibility of the Randomized Combinatorial File Structure and to illustrate the techniques and considerations of a system design utilizing this file structure, an information storage and retrieval simulator was designed. This simulator encompassed all of the randomized combinatorial file structuring concepts and the system design closely followed the item storage and retrieval block diagrams as presented on the following page. It is to be noted that the choice of the term "simulator" over the term "system" is made simply to indicate that no attempt was made to optimize instruction coding, an artificial data base was used, and core storage was used as the information storage media.

The first step in the design of a simulator is to select the combinatorial file structure parameters. The basic parameters are those of the number of descriptor attributes per item, M, and the total number of possible attribute codes, N. The former is dictated by the item description requirements while the latter is selected by maximum retrieval efficiency versus required file size considerations.

In order to illustrate the dependence of the selection ratio (retrieval efficiency) and the number of possible attribute code combinations (a consideration dependent on the required file size) on N, a computer program was devised to produce a



ITEM STORAGE BLOCK DIAGRAM



ITEM RETRIEVAL BLOCK DIAGRAM

table listing the number of possible attribute combinations (buckets) for N ranging from four to one hundred while M ranged from three to ten. In addition to the number of buckets, the table also listed the selection ratio and the number of addressed buckets for one out of M to M out of M retrieval attributes for each combination of N and M. Although this table is far too lengthy for inclusion in this paper, a portion of these data is presented in graphical form on Graphs 1 and 2. Both graphs cover only the narrow range of N from 6 to 20 and M from 4 to 6 which is of importance to the simulator design.

Graph 1 presents a plot of the number of buckets against the number of possible attribute codes, N, for various values of M, the number of attributes per item. Graph 2, on the other hand, presents the worst case selection ratio plotted against N for various values of M. Inspection of Graph 1 yields the fairly obvious observation of the rapid rise of the number of buckets due to an increase in N for a fixed M. This rapid increase in the number of buckets places a limit on the selection of N for a fixed M as discussed in the previous section. Graph 2 illustrates the decrease in the worst case selection ratio as N (possible codes) is increased for a fixed M. From the standpoint of retrieval efficiency, therefore, it is advantageous to select an N as close as possible to the limit as dictated by Graph 1.

The choice of core storage as an information storage medium precluded the use of a very large data base. In view of this constraint, an artificial data base of 4000 items with five descriptor attributes per item was formed. With the assumption of 4000 items with five descriptor attributes per item, the selection of N may be made.

Reference to Graph 1 with M = 5 indicates that N must be below 15 to yield fewer than 4000 buckets. In order to allow for the random distribution of items in the buckets, N = 14 is chosen. This choice yields 2002 buckets with a reasonable likelihood that any examined bucket will contain at least one item after a uniform random distribution of the items among the buckets.

The choice of N = 14 yields a worst case selection ratio of .357 as seen from inspection of Graph 2. The remaining selection ratios with the number of buckets addressed by each are presented in the table below.

Known Attributes (Requestors)	Selection Ratio	Addressed Buckets
1 out of 5	.35714	715 out of 2002
2 out of 5	.10989	202 out of 2002
3 out of 5	.02747	55 out of 2002
4 out of 5	.004995	10 out of 2002
5 out of 5	.004995	1 out of 2002

SELECTION RATIOS

(14 Possible Attribute Codes with 5 Attributes per Item)

Consideration of the preceding table illustrates the "divide and conquer" retrieval operation of the combinatorial file structure.

Effect of Duplicate Codes

Although the preceding table presents a true picture of the retrieval efficiency from the attribute code level, the effect of possible duplicate codes must be treated. A table presenting the probabilities of no duplicates of requestor attributes is shown below. The detail of possible duplications may be derived from Equation 8 and is not presented.

Request Attributes	Probability of No Duplications
5	.44669
4	.62536
3	.79592
2	.92857
1	1.00000

PROBABILITY OF NO DUPLICATE CODES
(Drawn from 14 Codes)

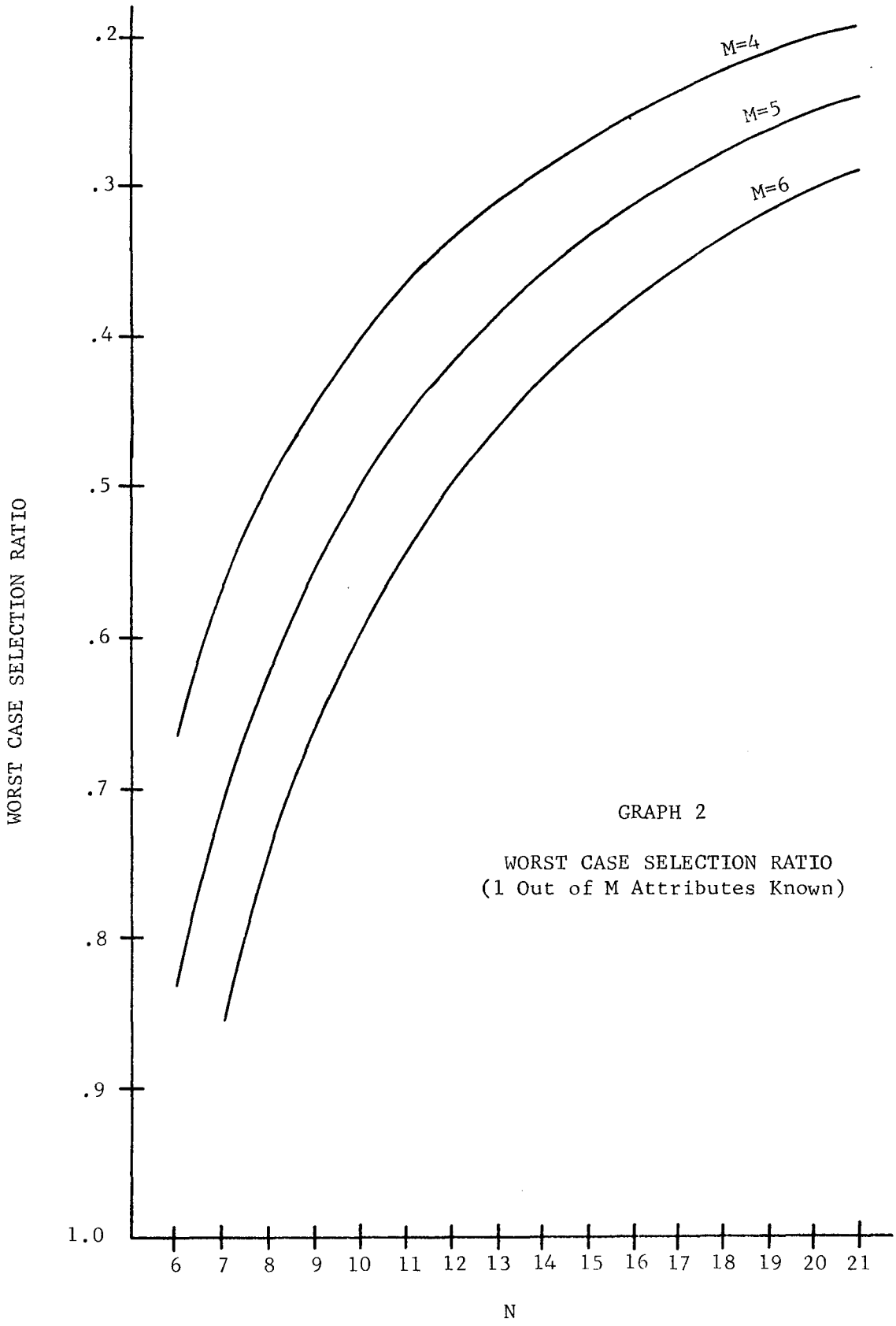
It comes as no surprise that the probability of duplicate codes increases as the number of specified request attributes increases. This increase, however, may be considered as a fairly small perturbation to retrieval efficiency in view of the far more rapid decrease in the selection ratio as the number of specified attributes increases (see Selection Ratios Table previously presented).

Implementation and Results

The combinatorial file structure simulator, comprised for the most part of Fortran IV routines, was implemented on an IBM 7040 computer and storage and retrieval tests made. These test runs were performed by first utilizing a storage mode programming package to read a prepared tape of 4000 descriptor described items which were then stored in a manner consistent with the combinatorial file structure; and secondly, by utilizing a retrieval mode package to read card input requests consisting of various numbers of requestors and to retrieve all items in the combinatorial file which were fully or partially described by each request.

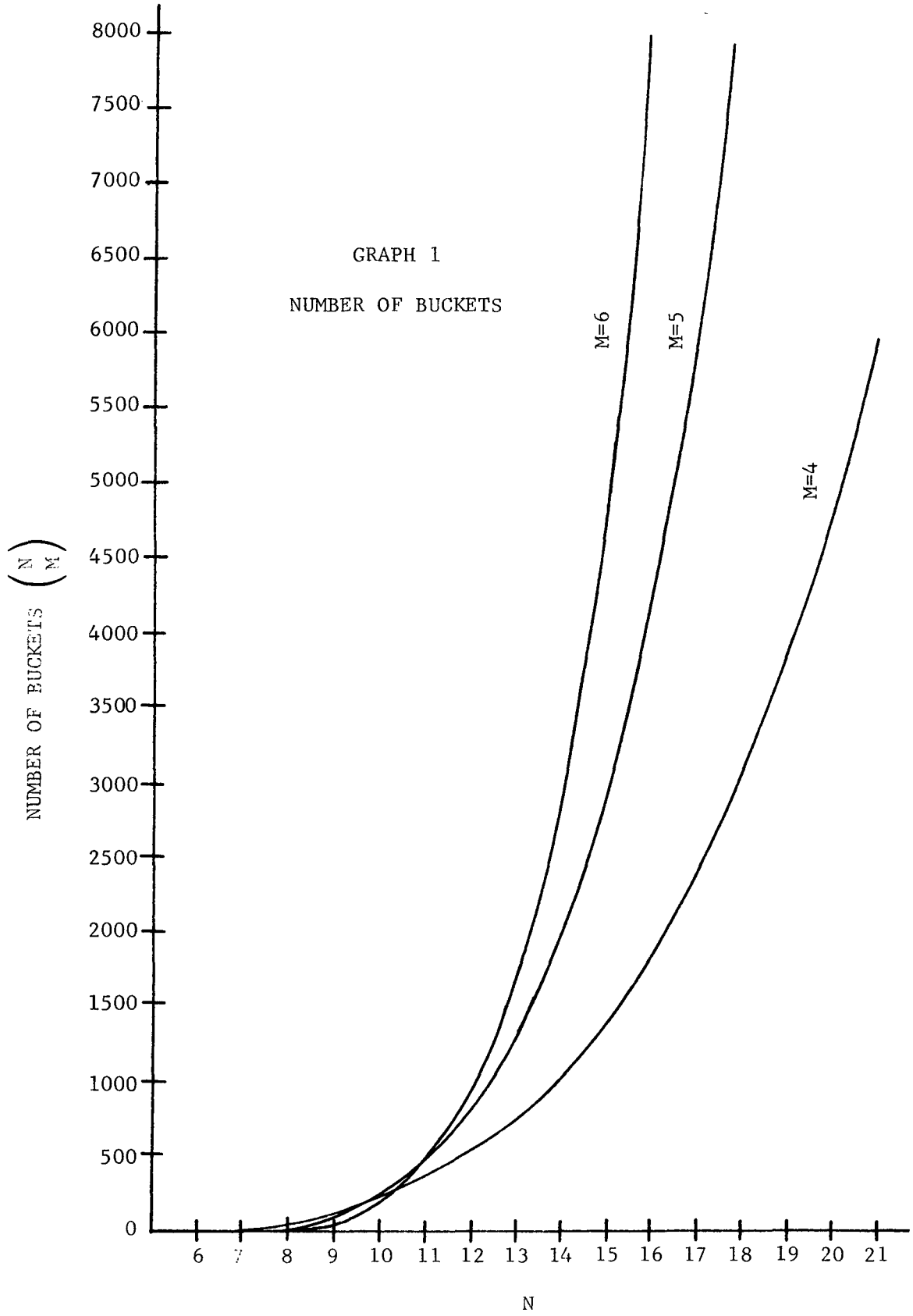
The retrieval times for a typical retrieval test are presented in the following table. This test consisted of five requests with the requests ranging from five to one requestor attributes respectively. Each request is listed with the number of request terms, time per bucket addressed, and the total time used to satisfy the request.

Although the time used to satisfy each request should not be considered as an evaluation of retrieval efficiency (e.g., no attempt was made to isolate input/output operations from the total time), the times are of interest when considered from a "relative time" standpoint. If the opera-



GRAPH 2

WORST CASE SELECTION RATIO
(1 Out of M Attributes Known)


GRAPH 1
NUMBER OF BUCKETS
NUMBER OF BUCKETS $\binom{N}{M}$
8000
7500
7000
6500
6000
5500
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
N
M=6
M=5
M=4

tion of retrieval mode simulator was as predicted, the "time to retrieve" should have been in direct proportion to the predicted number of buckets each request should address. This simply indicates that "time per bucket" should be relatively constant. This is indeed the case, and is illustrated in the table.

Request Terms	Time to Retrieve (Seconds)	Addressed Buckets (Predicted)	Time Per Bucket (Milliseconds)
5	.07	1	70
4	.28	10	28
3	1.42	55	26
2	5.50	202	27
1	17.85	715	25

As can be seen, all of the requests (with the exception of the first) exhibited an average "time per bucket" of 25 to 28 milliseconds. The discrepancy in the time of the first request may be attributed to input/output overhead as only one bucket is addressed and the time presented is near the resolution of the computer interval timer (16.67 milliseconds).

V. Conclusion

The randomized combinatorial file structure achieves storage and retrieval mapping by direct combinatorial address decoding. This direct mapping property negates the necessity for the storage of large file directories or list structures (e.g., an inverted file directory) while retaining the random access, rapid retrieval characteristics of list structured files (4, 8, 9, 18, 23, 27). The combinatorial mapping is conceptually similar to key-to-address transformation techniques (16, 22, 28, 31, 34, 35) but the transformation maps not as a function of a single key but as a function of a combination of keys.

Although the application of the file structure presented in this paper was made on a limited scale, implementation of the structure at a significantly greater level can be envisioned. In order to accommodate a large library of data, auxiliary random access storage such as disk files or auxiliary large capacity core storage could be utilized for the storage of items of information. The system design techniques presented in the application section are applicable to a large-scale system and should be easily integrated in an information storage and retrieval system.

BIBLIOGRAPHY

(1) ARON, J. Information systems in perspective. Computing Surveys, 1:4 (December, 1969) 213-236.

(2) BECKER, J.; HAYES, R. Information storage and retrieval: tools, elements, theories. Wiley, New York, 1963.

(3) BOURNE, C. Methods of information handling. Wiley, New York, 1963.

(4) CANTER, J.D.; DONAGHEY, C. E. UPLIFTS-University of Pittsburgh linear file tandem system. Communications of the ACM, 8:9 (September, 1965) 579-581.

(5) CHERRY, C.; ed. Information theory. Butterworth, London, 1961.

(6) CHEYDLEUR, B.; ed. Colloquium on technical preconditions for retrieval center operations. Spartan Books, Washington, D. C., 1965.

(7) CHEYDLEUR, B. F. SHIEF: a realizable form of associative memory. American Documentation, 14:1 (January, 1963) 56-57.

(8) DODD, G. Elements of data management systems. Computing Surveys, 1:2 (June, 1969) 117-133.

(9) DZUBAK, B. J.; WARBURTON, B. J. The organization of structured files. Communications of the ACM, 8:7 (July 1965) 446-452.

(10) FELLER, W. An introduction to probability theory and its applications. Wiley, New York, 1963.

(11) GARVIN, P.; ed. Natural language and the computer. McGraw-Hill, New York, 1963.

(12) GHOSH, S.; ABRAHAM C. Application of finite geometry in file organization for records with multiple-valued attributes. IBM Journal of Research and Development, 12:2 (March, 1968) 180-187.

(13) GOLOMB, S.; BAUMERT, L. Backtrack programming. Journal of the ACM, 12:4 (October, 1965) 516-524.

(14) GUSTAFSON, R. A randomized combinatorial file structure for storage and retrieval systems. Ph.D. thesis, University of South Carolina, Columbia (December, 1969).

(15) GUTENMAKHER, L. Electronic information-logic machines. Interscience, New York, 1963.

(16) HANAN, M.; DALERMO, F. P. An application of coding theory to a file address problem. IBM Journal of Research and Development, 7:2 (April, 1963) 127-129.

(17) HAYS, D. Introduction to computational linguistics. Elsevier, Amsterdam, 1967.

(18) HSIAO, D.; HARVARY F. A formal system for information retrieval from files. Communications of the ACM, 13:2 (February, 1970) 67-73.

(19) KENT, A. Textbook on mechanized information retrieval. Interscience, New York, 1962.

(20) KENT, A.; TAULBEE, O.; eds. Electronic information handling. Spartan Books, New York, 1965.

(21) KISEDA, J. R.; PETERSEN, H. E.; SEELBACH, W. C.; TEIG, M. A magnetic associative memory. IBM Journal of Research and Development, 5:2 (April, 1961) 106-121.

(22) KOHNEIM, A. G.; WEISS, B. An occupancy discipline and applications. SIAM Journal on Applied Mathematics, 14:6 (November, 1966) 1266-1274.

(23) LEFKOVITZ, D. File structures for on-line systems. Lecture notes for ACM Seminar on File Structures. Detroit, 1967.

(24) LEHMER, D. The machine tools of combinatories. In: E. Beckenbach; ed. Applied Combinatorial Mathematics. Wiley, New York, 1964.

(25) LUM, V. Y. Multi-attribute retrieval with combined indexes. Communications of the ACM, 13:11 (November, 1970) 660-665.

(26) MEADOW, C. Man-machine communication. Wiley, New York, 1970.
(27) MEADOW, C. The analysis of information systems. Wiley, New York, 1967.
(28) PETERSON, W. W. Addressing for random-access storage. IBM Journal of Research and Development, 1:2 (April, 1957) 130-146.
(29) SALTON, G. Automatic information organization and retrieval. McGraw-Hill, New York, 1968.
(30) SALTON, G. Data manipulation and programming problems in automatic information retrieval. Communications of the ACM, 9:3 (March, 1966).
(31) SCHAY, G.; RAVER, N. A method for key-to-address transformation. IBM Journal of Research and Development, 7:2 (April, 1963) 121-126.
(32) SCHECTER, G.; ed. Information retrieval: a critical view. Thompson, Washington, D. C., 1967.
(33) SPIEGEL, J.; WALKER, D.; eds. Information systems sciences: proceeding of the second congress. Spartan Books, Washington, D. C., 1965.
(34) TARTER, M.; KRONMAL R. Non-uniform key distribution and address calculation sorting. In: Proceedings of 21st National Conference, ACM. Thompson, Washington, D. C., 1966.
(35) TOYODA, J.; TEZUKA, Y.; KASHARA, Y. Analysis of the address assignment problem for clustered keys. Journal of the ACM, 13:4 (October, 1966) 526-532.
(36) VICKERY, B. On retrieval system theory. Butterworth, London, 1965.
(37) WALSTON, C. E. Information retrieval. In: F. Alt; M. Rubinoff; eds. Advances in Computers. Academic Press, New York, 1965.