International Journal of Computer and Information Sciences, Vol. 2, No. 3, 1973

Bitmaps and Filters for Attribute-Oriented Searches

J. Michael Burke[1] and J. T. Rickman[1]

Revised March 1973

This paper presents inverted file structures for attribute searches in the form of bitmaps. It also presents techniques to reduce search lengths through the use of blocking and block-filter records.

1. INTRODUCTION

The basic justification for using an inverted file in a data retrieval system is to reduce the search lengths of specified data classes or individual records which satisfy user requests. The additional complexity inherent in inverted file structures is usually justified in on-line systems which require minimal search lengths, or in systems which can use relatively small inverted files to access large data bases.

The major drawback of inverted files is the storage required for the list of record addresses corresponding to each particular key. Since one record address can belong to many such lists, the size of an inverted file can grow quite large. For systems requiring extremely short response times for every request it may also become necessary to keep the inverted file in main core since auxiliary storage access would cause severe degradation in total response time.

It is the purpose of this paper to present alternative file structures employing strings and arrays of bits which help to reduce these storage requirements and at the same time facilitate rapid inverted file processing for Boolean searches. One alternative structure which is presented reduced search lengths by 78 %.

[1] Computer Science Department, The Pennsylvania State University, University Park, Pennsylvania.

2. GENERAL OVERVIEW

Some basic definitions are now given: A *generalized file structure* is one in which every record is assigned a unique address.[1] A collection of these unique addresses is commonly known as a *directory*. Associated with each record are *keys* or *attributes*. An inverted file differs from the generalized file directory in the way it is constructed. The *inverted file* is built using the keys or attributes rather than the record addresses. Each attribute or key has a list of addresses of all records that pertain to that distinct key. The word "inverted"[2] evolves from the idea that an inverted file is one organized (usually from an existing file) in such a manner as to facilitate the search and/or retrieval of data based on an attribute or groups of attributes different from those used to organize the original file.

In order to make the following more meaningful, it is necessary to explicitly define three terms which are often confused.

(a) *Key*—A data item that serves to uniquely identify a data record. The key may be embedded within the record, precede the record, or be separated from the record as in an inverted file.

(b) *Attribute*—A characteristic defining or specifying something; the range or characteristic that an item may acquire.

(c) *Value*—That quantity or quality assigned to an attribute.

Now it is possible to distinguish between a search conducted on a generalized file using keys and a search conducted on an inverted file using attribute keys constructed from data items in the records of the data set. In an inverted file organization the attribute keys can acquire a value in a range bounded by the possible values of the individual data items in the file.

The advantage of inverted file processing can be seen in the following example.

Example. Given a general file F, locate a set of records E, generally much smaller than F, having some common keys. The steps involved in this type of file processing are: (1) Get the address of every record $r(i)$ in F. (2) Search $r(i)$ for the common keys.

Now let r be the ratio of time and effort involved in obtaining E from F versus the time needed to process every record in F. It can readily be seen that $r = 1$ in a generalized file organization. Using an inverted file organization, the appropriate keys are looked up in the directory. The lists of records containing the desired attributes comprise the set E.

As long as the size of E remains relatively small with respect to the size of F the ratio r of time and effort to obtain E versus searching all records contained in F remains small.

The above example further demonstrates what is meant by "inverted" since using keys to obtain addresses is opposite from using addresses to find keys.

The retrieval of groups of records containing sets of like characteristics or qualities lends itself to a particular type of file organization which emphasizes retrieval through encoded attributes of records within an inverted file rather than actual record inspection. The concept of an inverted file allows multirecord retrieval at relatively low processing cost; however, the storage requirements of the inverted file may limit, or even nullify, the effectiveness of such a system.

Another method used to construct the inverted file is to encode each attribute or key. Encoding normally permits the keys to be compressed and thereby reduces storage requirements. An example of this would be the simple ordering of the keys and their mapping onto the integers. Still another variation uses the actual data of the record but codes it into a bit string in which each attribute value is represented by a position in the bit string. If a particular bit is set (equal to one), then the record contains that attribute value. When this method is used a one-to-one correspondence exists between a bit string in the inverted file and a record of the data set. At first glance this may not appear to save much; however, the pertinent attributes are now positional and in binary coded form. One needs only to perform logical operations (and, or, not) on either columns or rows of the inverted file to obtain a desired set E of the original file F. A simple algorithm can convert the directory entries into actual record addresses.

3. SEARCH TECHNIQUES

There are two related search algorithms that lend themselves to an inverted file: attribute-oriented searches and record-oriented searches. Both of these methods can employ coded binary strings which are generated from the attributes of each record in the data base. The logical collection of these bit strings into a matrix form will be called a bitmap. In the bitmap there is a one-to-one correspondence between the row entries of the inverted file and the records in the data base. The method used to process this inverted file now becomes dependent upon machine hardware, the length of the individual bit strings, and the number of entries in the inverted file.

When using an attribute-oriented search the records are arranged in a matrix in which each column represents a specific attribute $A(j)$ and each row represents the logical collection of attributes for a given record $r(i)$. This is shown in Fig. 1.

Attribute-oriented searches are extremely effective when the relative number of attributes in $r(i)$ is small. One has only to logically "AND" the

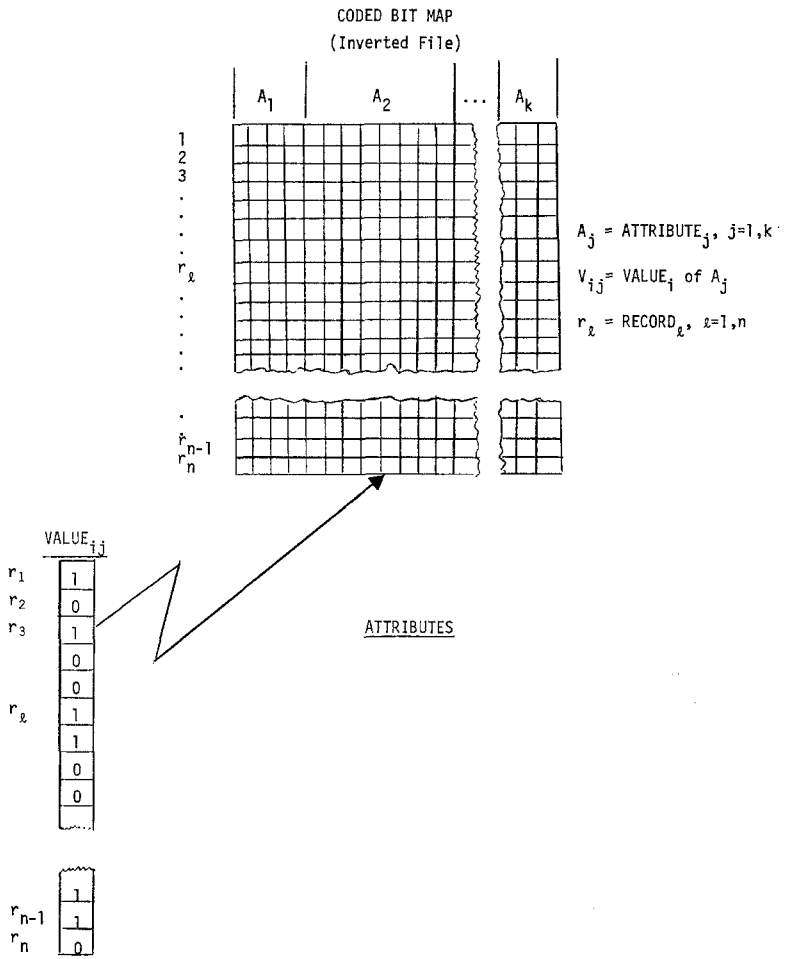


Fig. 1. Logical record of attribute-oriented search. Each 1 or "on" bit of information of a given record r_i signifies a given particular value V_{ij} of attribute A_j which is mutually exclusive of all other values of attribute A_j in (r_i).

columns corresponding to the desired attributes to obtain an answer vector. Since any "1's" propagated to the answer vector denote records that contain the desired attribute values, the only remaining task involves the simple transformation of "1's" in the answer vector to actual record addresses. Logical "ANDing" proceeds rapidly in most computers because of their ability to "AND" long bit strings in a few machine operations. However, some hardware may require that all, or at least a large segment, of the directory remain resident in core in order to operate efficiently.

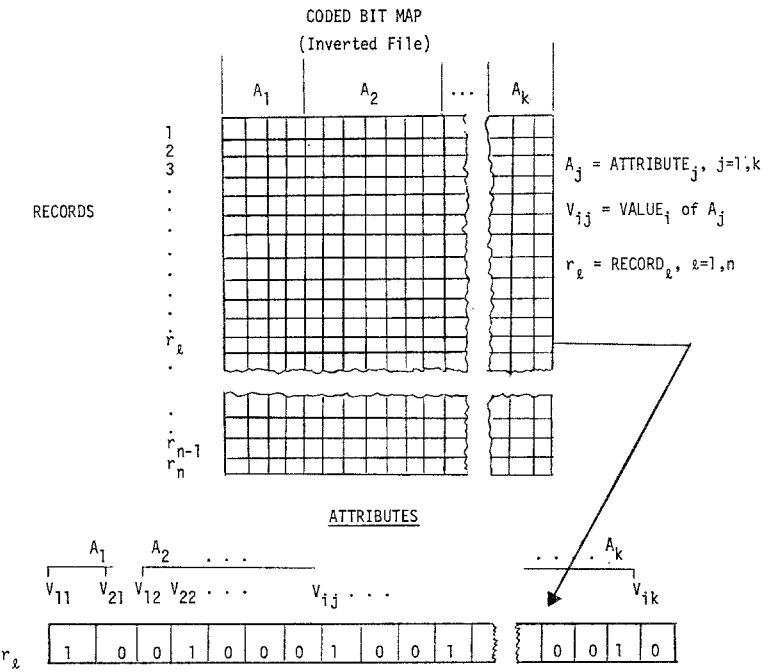


Fig. 2. Logical record of record-oriented search. Each 1 or "on" bit signifies that record r_l contains the attribute values denoted by V_{ij} of attribute $A_j, j = 1, k$.

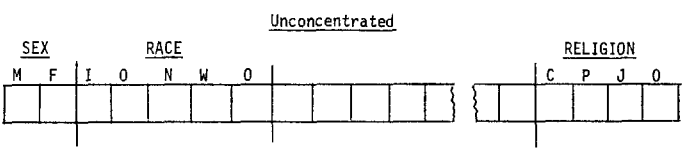
When a record-oriented search is used the records are again arranged in matrix form, but logical operations are performed on each row (Fig. 2) rather than each column. Record-oriented searches are most effective when the number of records in a data base is not extremely large and when there is a sufficient number of coded attributes contained in a record to permit efficient machine operations on the bit strings. One has only to logically "AND" record $r(i)$ of the directory with a target or request vector to immediately determine if record $r(i)$ contains the desired attribute values. The two main advantages that result from this method are the immediate determination of a request match in the file and the ability to segment the directory, keeping most of it in secondary storage without noticeably degrading performance.

Because of the above-mentioned advantages and the additional versatility inherent in a record-oriented search, experiments were conducted to explore and define advanced techniques to improve the overall performance and adaptability of these types of inverted files. Sequential record-oriented searches within a bitmap provide the basis of this study.

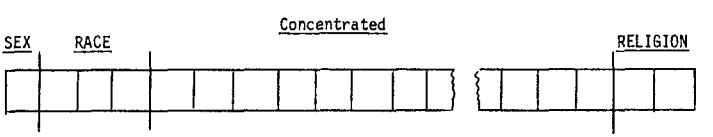
4. EXPERIMENTAL MODEL

In order to investigate the type of data retrieval that results in multiple record matches generated from a single request, the following data base was defined. This data base contained records about humans and their physical characteristics. In order to have the added ability of rapidly changing the base, records of ten major attributes (each containing up to seven values) were built, using a random number generator to formulate the values of the individual attributes.

Each row in the bitmap was 48 bits long, one bit for each value. For example, bit positions 1 and 2 are used for the SEX attribute, bit positions 3–7 for RACE, etc., as is shown in Fig. 3. No attempt was made to produce minimum-length codes for the individual records (i.e., up to eight different values could be coded in only three bits using binary coding). The space saved by minimal coding would not have been appreciable since the attributes



Standard coding--one bit per attribute value.



Bit Position	Attribute	Values	Binary Code	Decimal Equivalent
1	Sex	Female	0	0
		Male	1	1
2→4	Race	Indian	000	0
		Oriental	001	1
		Negro	010	2
		White	011	3
		Other	100	4
n-1→n	Religion	Catholic	00	0
		Protestant	01	1
		Jewish	10	2
		Other	11	3

Fig. 3. Minimization of code length.

of an entire record required only six bytes. Each record of the bitmap contains exactly ten "1's" that completely describe the individual human corresponding to that entry in the inverted file.

In order to examine the search lengths within the model, 41 requests were generated manually. In these experiments a request was merely a string of attribute and value pairs. As shown in Fig. 4, the requests in the experimental data had each attribute separated from its value by a period: Individual pairs were separated by commas and/or one or more blanks. Individual requests were delimited by semicolons. All individual pairs of a request were automatically coded into a 48-bit request vector. When the complete request string was coded the search lengths were measured for each search of the inverted file.

First a request vector was passed over each record in the file. If the

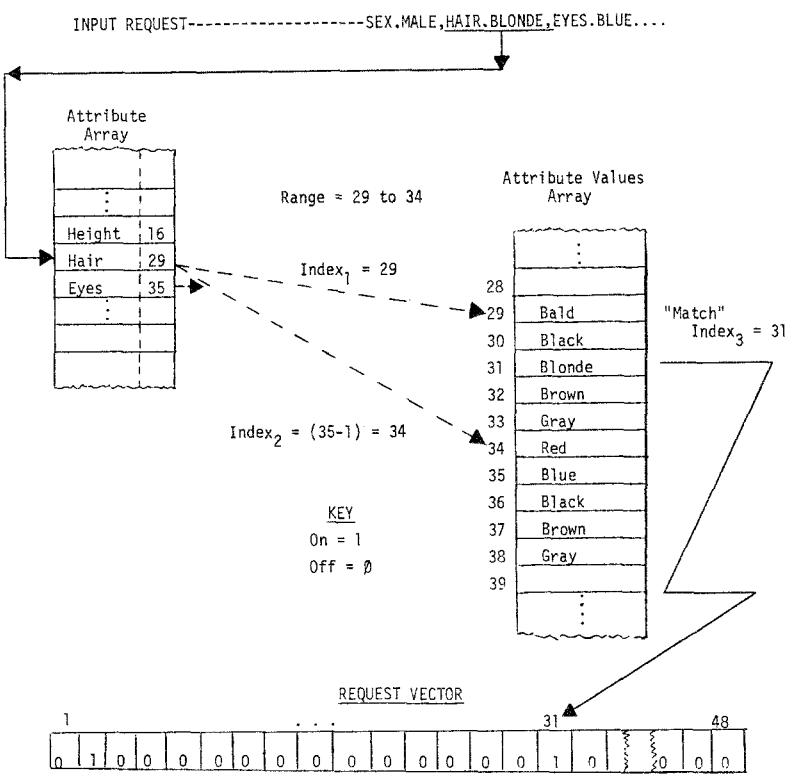


Fig. 4. Input request coding. The major attribute array gives an entry point and a range of the search; the attribute values array then gives the actual index into the request vector. When a match is found the corresponding bit in the request vector is set.

result obtained by logically "ANDing" the request vector and an inverted file record $r(i)$ matched the request vector, then $r(i)$ was a member of the set of records requested, as shown in Fig. 5. Note that all the attributes of an individual record $r(i)$ are compared in *one* logical operation regardless of how many attributes it contained. Once a match was found the actual data record's address was computed using i.

The next step in the study was to investigate the effect of blocking the inverted file's records. The records were combined into logical blocks of from 2 to 30 records. Each block contained a block-filter record which was merely an additional record having an "on" bit for any value "on" in any

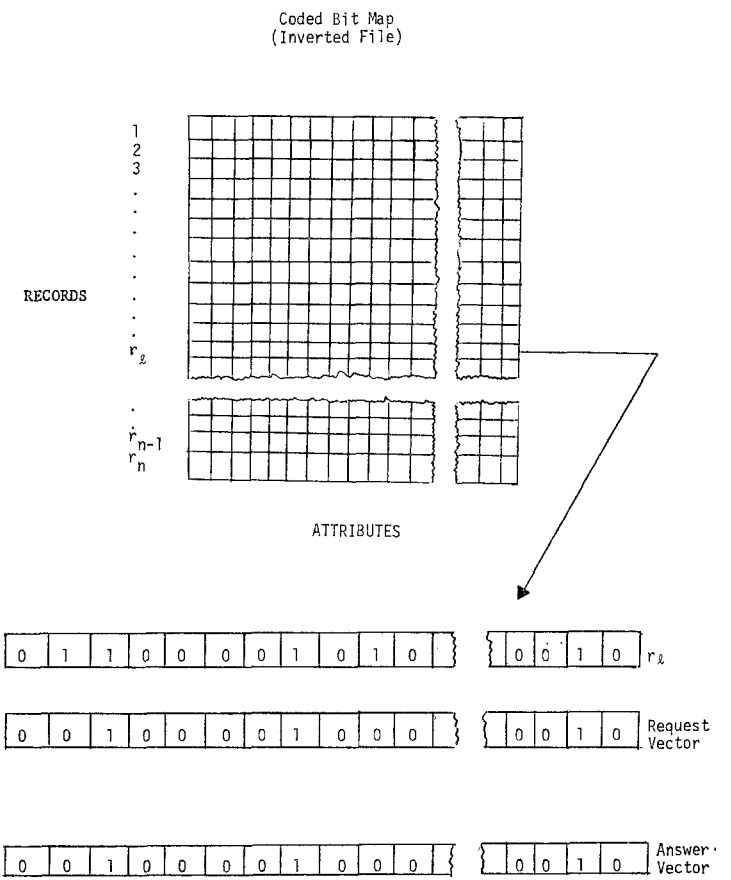


Fig. 5. Inverted file processing. The "Answer" vector is the logical product of r_l ANDed with the request vector. Note that a match occurred because the answer vector is a duplicate of the request vector.

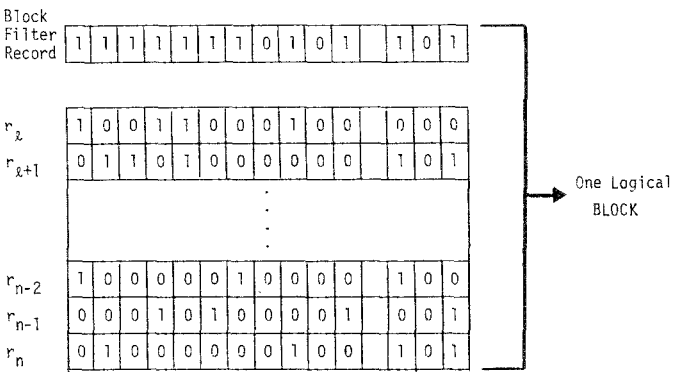


Fig. 6. Block-filter record generation. By the process of "Oring" records r_l through r_n all attribute characteristics of the block are propagated in the Block-Filter Record. Note, however, that a match between the Block-Filter Record and a request vector does not guarantee that a record(s) contained in the block will match the request. The Block-Filter Record only eliminates the those blocks that could not contain a desired record.

of the records of that block. The block-filter record is generated by logically "ORing" all of the records in that block, as is shown in Figure 6.

Block-filters should enhance the search procedure because any time a block-filter record fails to match the request vector the search length is reduced by the number of records in the block by excluding them from the search. If, however, a block-filter record gave a positive match, then each record of that block had to be checked as in normal sequential processing.

The technique of blocks was again modified through the use of sorting. All the records in the inverted file were sorted before blocking. Here the ordering of the physical attributes played an important role. In the experiments an attempt was made to place the most discriminating and most queried physical attributes (major attributes) in the high-order sorting positions of the bit strings so that after sorting these attributes would be least likely to change within a block. In other words, it would enable the filter to represent a larger number of records in a block and still retain its ability to discriminate. For any given data set the choice and positioning of the more important characteristics will greatly affect the expected search lengths. For the same reason it was thought unwise to use minimum-length binary codes for values because most filters would become all "1's."

The technique of blocking the inverted file's records and creating a block-filter record for every block was also used to separate the directory into easily managed segments. It also allowed the entire inverted file to reside on a secondary storage medium and to be sequentially processed in

terms of blocks. Since the use of blocks and filterrecords decreases search lengths, the inverted file may no longer need to remain in core. With this technique almost any small computing system with tape storage would have the ability to process a relatively large inverted file data base.

This type of blocking also offers a simple solution to the problem of updating the inverted file with new record insertions and old record deletions. A block can be constructed to contain more record locations than necessary; in the event an insertion is required the new record's attribute bit string is "ORed" with the block-filter record and placed in the first available position. Deletion of an existing entry requires only that the record's values be set to "off" since no request will match a null set of attributes. When the blocks become laced with these null records a simple update which deletes all null entries and compresses the remaining entries of each block could be performed. When the effectiveness of the block filters is sufficiently degraded through many deletions the updating routine should reconstruct blocks and block-filter records from the remaining entries.

5. ANALYSIS

As was the intent of this study, it has been shown that some storage and processing techniques exist which can lower the normally high storage requirements and search lengths of an inverted file. If inverted files are required, it should be obvious that the bitmap structure presented here reduces storage requirements. Therefore this analysis will concentrate on search lengths.

Since the experimental data base of human physical attributes was generated using random numbers, it does not reflect the true nature of an actual identification file. It does, however, illustrate that a data base with similar qualities can be processed efficiently with minimum storage allocation.

For a more detailed analysis the requests were categorized as follows, where "major" implies an attribute which is more often used in a request (i.e., sex, race, etc.), and is more discriminating (i.e., fewer degrees of freedom):

(1) Many descriptors (7–10 attributes).
(2) Average number of discriptors (4–7 attributes).
(3) Few descriptors (1–3 attributes).
(4) Many major descriptors (four or more attributes).
(5) Average number of major descriptors (3–4 attributes).
(6) Few major descriptors (1–2 attributes).
(7) Typical requests for the given data set.

The purpose of these groupings was to show the various behaviors of the search algorithm under different request conditions and blocking factors.

Since the data used in this experiment were of human physical attributes, it is obvious that major attributes such as sex, race, weight, etc., are more discriminatory than others. With this in mind the attributes were ordered so that attributes of decreasing discriminating power were placed at the low-order end of the request vector.[3]

In the real world it is true that some attributes are known or requested more often than others. For example, it is much easier to determine an individual's sex than his age. Therefore, sex would be placed in a higher-ordered sort position than age. The construction of the inverted file should be dictated by the environment of its usage.

The request vectors for the seven groups were passed over the file several times while varying the number of records per block from 2 through 30. The search ratio r of the number of comparisons required for a request to be satisfied versus the total number of comparisons required to satisfy the request if no blocking was used was calculated for each request. Search lengths with reductions of up to 50% were measured. An average for each group was also calculated and this is shown in Table I.

Although the effect of blocking the inverted file records resulted in a marked increase in processing effectiveness, even better results were achieved by sorting the inverted file records. After sorting the directory all the records in a sufficiently small contiguous group contained similar higher-order attributes. This produced an additional advantage since the groups of contiguous records were similar, a block could become much larger without degrading its filter.

Table I. Average Search Ratios per Request Group (Without Sorting)[a]

Group	Range of search ratio	Best blocking factor[b]
Many random attributes	0.2450–1.0005	5
Average number of attributes	0.3560–1.0260	5
Few random attributes	0.7876–1.0454	2
Many major attributes	0.3264–1.0083	5
Average number of attributes	0.4905–1.0283	5
Few number of attributes	0.4406–1.0333	4
Typical requests	0.5706–1.0290	6

[a] When all the requests were taken as a single group the best ratio resulted with a blocking factor of five records per block.

[b] Records per block.

The results of using blocked records with sorting were analyzed in a manner analogous to the unsorted blocks. The individual requests, as well as the request groupings, remained constant and the blocking factor again varied from 2 to 30.

The effect of blocking and sorting resulted in a further marked increase in processing effectiveness, bringing the average percentage of compares per request versus straight sequential processing to 22%. A comparison of the relative effectiveness of blocking with and without sorting can be seen in the graph in Fig. 7 and in Table II.

It must be stressed that overall effectiveness of blocking an inverted file directory, with or without a sort, is completely dependent upon the data set under investigation; its organization, the attributes, the inherent hierarchical qualities of these attributes, and most of all, the request environment in which it will be used.

It should also be noted that an alternate inverted file organization is a tree structure. A tree can be designed so that every progressive step down from the root defines a larger set of attributes which the records below the current node have in common. Thus in the extreme case one only has to trace

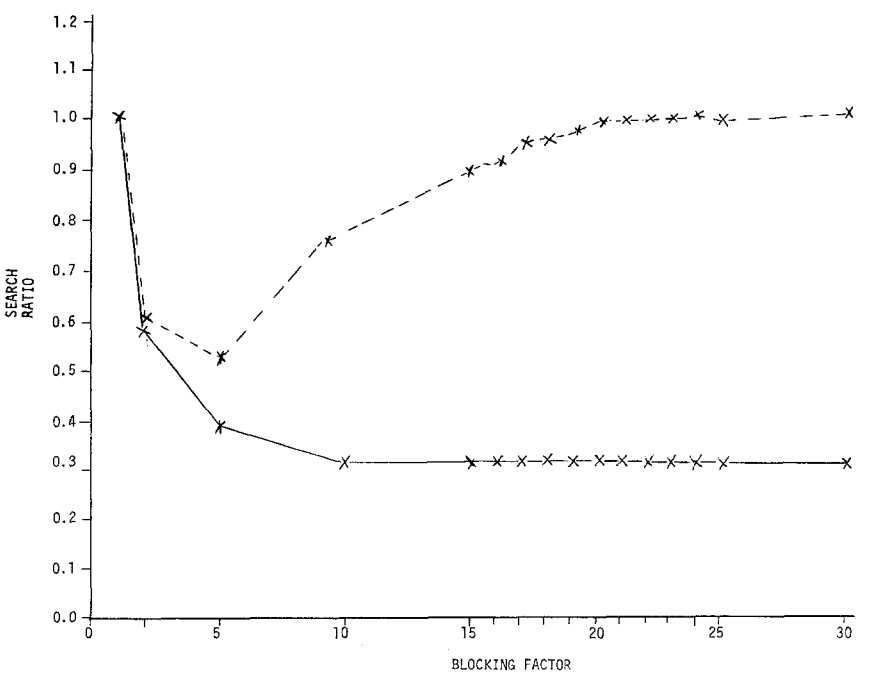


Fig. 7. Search ratios of sorted (——) vs. unsorted (– – –) bitmap. The search ratio is the search length with block filters divided by the sequential search length.

Table II. Average Search Ratios per Request Group (with Sorting)[a]

Group	Range of search ratio	Best blocking factor
Many random attributes	0.1078–0.5001	17
Average number of attributes	0.2368–0.5041	19
Few number of attributes	0.5840–0.7198	15
Many major attributes	0.0700–0.5014	23
Average number of major attributes	0.1192–0.5113	19
Few number of major attributes	0.2089–0.5160	17
Typical request	0.2199–0.5402	22

[a] When all the requests were taken as a single group the best ratio resulted with a blocking factor of 17 records per block.

down the proper nodes (attributes) in order to obtain the record, or records, with all the required attributes. The nodes crossed by the path to the desired records are merely a collection of all the attributes which those records contain. It can be seen that for any request the path to the desired records is quite short, thereby ensuring that the amount of time and effort expended to find the set of records (E) that satisfy the request versus the time and effort required to search the entire file (F) is very small.

Hence the inverted file structure studied does not have minimal search length but it can be easily implemented using common sequential data structures, and it can even be used on tape files where the directory shares a physical record with its corresponding data records. Such a tape file would have additional advantages when output is required of records which satisfy search requests.

6. CONCLUSIONS

This work has investigated several advanced techniques in inverted file organization which greatly reduce the normally high storage requirements. Blocking of directory records for record-oriented searches offers a greater range of versatility in processing and at the same time it is an easy structure to implement. The simple blocking of the directory records provided up to a 50 % reduction in search lengths, while blocking with sorting reduced search lengths to a mere 22–27 % of normal sequential directory processing. A possible improvement is now under study which uses multilevel (tree structured) block-filters which assume a different sorting order on specific attributes within a block.

REFERENCES

1. D. Hsiao and F. Harary, "A formal system for information retrieval from files," *Commun. ACM* **13**(2):69 (1970).
2. Philip B. Jordian, *Condensed Computer Encyclopedia* (McGraw–Hill, New York, 1969).
3. D. A. Huffman, A method for the construction of minimum redundancy codes," *Proc. IRE* **40**(10):1098–1101 (1952).
4. Ivan Flores, *Data Structures and Management* (Prentice-Hall, 1970), Chapter 14, pp. 318–335.
5. R. C. Brill, "The TAXIR Primer, "Occasional Paper No. 1, Institute of Artic and Alpine Research, University of Colorado, Boulder, Colorado, 1971.
6. L. Hudson, R. Dutton, M. Reynolds, and W. E. Walden, "TAXIR, A biologically oriented information retrieval system as an aid to plant introduction," *Econ. Botany* **25**(4):401–406 (1971).