Information Processing & Management Vol. 21, No. 5, pp. 433–442, 1985
Printed in Great Britain.

0306-4573/85 $3.00 + .00

A COMPARATIVE STUDY OF MULTIPLE ATTRIBUTE TREE AND INVERTED FILE STRUCTURES FOR LARGE BIBLIOGRAPHIC FILES

S. V. NAGESWARA RAO and S. SITHARAMA IYENGAR
Department of Computer Science, Coates Hall, Louisiana State University, Baton Rouge, LA 70803, U.S.A.

and

C. E. VENI MADHAVAN
School of Automation, Indian Institute of Science, Bangalore-560012, India

(*Received* 28 *February* 1985; *in revised form* 22 *March* 1985)

Abstract—A variety of data structures such as inverted file, multi-lists, quad tree, *k-d* tree, range tree, polygon tree, quintary tree, multidimensional tries, segment tree, doubly chained tree, the grid file, *d*-fold tree, super *B*-tree, Multiple Attribute Tree (MAT), etc. have been studied for multidimensional searching and related problems. Physical data base organization, which is an important application of multidimensional searching, is traditionally and mostly handled by employing inverted file. This study proposes MAT data structure for bibliographic file systems, by illustrating the superiority of MAT data structure over inverted file. Both the methods are compared in terms of preprocessing, storage, and query costs. Worst-case complexity analysis of both the methods, for a partial match query, is carried out in two cases: (a) when directory resides in main memory, (b) when directory resides in secondary memory. In both cases, MAT data structure is shown to be more efficient than the inverted file method. Arguments are given to illustrate the superiority of MAT data structure in an average case also. An efficient adaptation of MAT data structure, that exploits the special features of MAT structure and bibliographic files, is proposed for bibliographic file systems. In this adaptation, suitable techniques for fixing and ranking of the attributes for MAT data structure are proposed. Conclusions and proposals for future research are presented.

Keywords: Design and analysis of algorithms, multidimensional data structures, complexity analysis.

INTRODUCTION

The inverted file is one of the most popular and widely applied techniques for physical data base organization. This technique is based on the extension of 'the concept of lists' to multiple dimensions, and is also referred to as the inverted lists method. Knuth[1] presents a thorough treatment of inverted lists structure. In recent times, there has been a phenomenal growth in literature on multidimensional tree structures. A variety of data structures—*k-d* tree, quad tree, range tree, *d*-fold tree, quintary tree, multidimensional tries, segment tree, the grid file, polygon tree, super *B*-tree, Multiple Attribute Tree (MAT), etc.—have been studied. A comprehensive treatment on multidimensional tree data structures can be found in Bentley and Friedman[2], and Overmars[3]. A close look at the literature reveals that there does not exist a panacea for multidimensional search problems, but, each structure has its own merits when used in certain applications. Hence, given a problem, the nature of the problem and related operations should be investigated to arrive at a suitable data structure for the problem. This study is a attempt to illustrate the specific suitability of MAT to bibliographic file systems. In particular, MAT will be shown to be a better alternative than the inverted file.

Most of the multidimensional tree structures are based on the extension of 'the concept of binary trees' to multiple dimensions. But, MAT is based on a totally different

philosophy, as will be explained later, and seems to be particularly well suited for bibliographic files. However, the immense interest in these multidimensional tree structures can be attributed to their flexibility and adaptability in dynamic environments. The MAT structure was proposed for physical data base organization by Kashyap *et al.*[4], and was shown to be more efficient than inverted file-based data base organization in terms of access times. In [4], the superiority of MAT over the inverted lists is illustrated by taking several real-life databases. Gopalakrishna and Veni Madhavan[5] proposed and analyzed a more efficient and modified version of MAT structure. In [5], the effectiveness of a MAT-based data base organization over inverted file-based data base organization was demonstrated using six real-life databases and four types of query complexities. Bentley[6] proposed the *k-d* tree as an improvement over structures like inverted lists, quad trees, etc. In [6], the *k-d* tree is shown to be very effective for partial match queries. Veni Madhavan[7] illustrated that the MAT outperforms the *k-d* tree in many real-life query situations by taking into account the ranking of attributes and query probabilities. Nageswara Rao, Veni Madhavan and Sitharama Iyengar[8] developed an efficient adaptive range search algorithm, and a novel dynamization technique for MAT data structure. This adaptive range search algorithm dynamically exploits the nature of the query and the structure of the MAT. This dynamization technique allows intermixing of insertions, deletions and queries, and also rebuilds the structure at suitable points to ensure good response to queries.

The remainder of this study is organized as follows: Section 1 gives the basic definitions and notations of MAT data structure. The special features of MAT are discussed in Section 2. The specific query properties of bibliographic files are discussed in Section 3. Section 4 presents a comparative study of MAT and inverted file structures. Both structures are analyzed for their worst-case complexities in two cases: (a) the directory resides on main memory, (b) the directory is in secondary memory. In both cases, MAT is shown to be more efficient than inverted file. An efficient adaptation of MAT, which exploits the special features of MAT and bibliographic files, is presented in Section 5. In Section 6, an example is provided to illustrate the process of answering a typical bibliographic query using MAT and inverted list methods. A discussion on further work and conclusions are presented in Section 7.

1. MULTIPLE ATTRIBUTE TREE (MAT)

Construction and properties of the multiple attribute tree data structure for a set of records are discussed in [4, 5]. Here, we give a formal definition for MAT.

Definition 1: A k-dimensional MAT on k attributes for a set (file) of records is defined as a tree of depth k, with the following properties:

(1) it has a root,
(2) each child of the root is a (k-1)-dimensional MAT, on second to kth attributes for the subset of records that have the same first attribute value. The first attribute value is the value of the root of the corresponding (k-1)-dimensional MAT, and
(3) the child nodes of the root are in ascending order of their values.

From the above definition, we observe that the root is at level zero and the k attributes correspond to the next k levels of MAT. The root node does not have a value associated with it. All other nodes of MAT are associated with the corresponding attribute values. The data structure MAT is constructed as follows: (a) the records are sorted, in ascending order, on all attributes, and (b) all elements of an attribute having the same value are recursively combined into a node, starting from the first attribute. One of the notable features of MAT is that each distinct combination of the values of k attributes (i.e. each record or point) is represented by a unique path from the root to a terminal node. Any terminal node contains, in addition to an attribute value, a physical record pointer (or page pointer). Figure 1 shows a sorted data file and the corresponding MAT data structure.

The properties of MAT structure can be characterized in terms of the following notations: Let N: number of records, M: number of nodes of MAT, k: number attri-

A_1	A_2	A_3	A_4	Record pointer
1	1	2	6	4
1	1	3	3	1
1	1	3	5	6
1	1	4	1	3
1	2	1	2	2
1	2	5	6	7
2	3	5	1	8
2	3	5	7	5

(a) Sorted data file

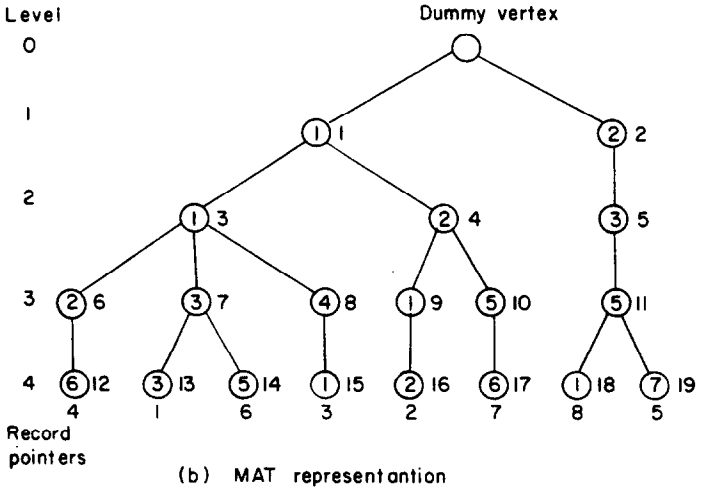


Fig. 1. Input data and corresponding MAT.

butes, *filial-set*: set of nodes, at the same level, which have the same parent, and s_j: average size of a filial-set at the same level j, for $j = 1, 2, \ldots, k$.

Hence, on an average, we have

$$s_0 = 1, N = s_0 s_1 \ldots s_k = \prod_{j=0}^{k} s_j = \prod_{j=1}^{k} s_j, \text{ and } M = \sum_{j=0}^{k} \prod_{i=1}^{j} s_i.$$

For a *symmetric* MAT structure, all filial-sets are of the same size s, and we have, $s = s_1 = s_2 = \ldots s_k$, and, $N = s^k$. Hence, we assume that on an average, for any MAT,

$$s_j = O(N^{1/k}) \tag{1}$$

$$\prod_{j=1}^{k-1} s_j = N/s_k = O(N^{1-1/k}). \tag{2}$$

In the following sections, we use a symmetric MAT structure for analysis. Without loss of generality, the expressions (1) and (2) are used in estimating complexity measures. The analysis is carried out on the lines of Bentley and Friedman[2] and Overmars[3].

2. SPECIAL FEATURES OF MAT

Most of the tree structures like, *k-d* tree, quad tree, range tree, etc. are based on the concept of multidimensional 'divide and conquer.' In this approach, the problem

domain is recursively divided into smaller regions of same dimensionality, and the results from these smaller regions are combined to produce the answer to the problem. But MAT is based on a different concept, where the problem is solved by reducing the dimensionality of the problem domain by one in every step.

MAT has several properties which make it an attractive choice for data base applications. Unlike many other data structures, the query and data properties can be made use of to make MAT more efficient. The following are the two important properties:

(i) Ranking of attributes: The ranking of attributes in a decreasing order of the probability of their occurrence in a query, helps in 'pruning' the search process while answering a query. This results in faster responses to queries.

(ii) Clustering effect: The clustering of tree nodes having the same parent helps the search process in MAT. A breadth-first linearization for the MAT directory makes use of this property in minimizing the number of pages accessed from the secondary memory. This aspect is elaborated on in Sections 4 and 5.

3. BIBLIOGRAPHIC FILES

In this section, we present the properties of bibliographic files from the point of view of query specifications. Most queries encountered in bibliographic files fall into the generic class of partial match queries. Queries specifying author's name, title, etc. are frequently encountered. Full match and range queries are infrequent. It is logical to expect the user to specify a few keywords and phrases from the titles rather than 'full and exact' titles. Similarly, the authors'/editors' first name is most often specified rather than the full name. This type of specification in a query naturally supports partial match retrieval. Hence, hereafter, partial match retrieval is used as an important criterion to compare the performance of MAT with inverted file. Normally, the number of records retrieved is of the order of tens, and cases retrieving hundreds of records are uncommon. In interactive environments, the response time to a query has to be kept small. All these special features are exploited in proposing a new adaptation of MAT data structure for bibliographic files, in Section 5.

4. COMPARISON OF PERFORMANCE

The structures used for the multidimensional search and related problems are characterized by a data structure and corresponding search algorithms. The performance of a structure, A, is expressed in terms of three cost functions of N and k;

- $P_A(N, k)$, the cost of *preprocessing* N records into a data structure;
- $S_A(N, k)$, the *storage* required by the data structure;
- $Q_A(N, k)$, the *search cost* or *query cost*.

Bentley and Friedman[2] present these measures for sequential scan, projection (inverted file), cells (quad tree), k-d tree, range tree, and k-ranges. In the analysis, all the structures are assumed to reside in the main memory.

In this section, we develop these measures for inverted file and MAT. Two cases, when the directory (data structure) resides in the main memory or the secondary memory, are investigated. The preprocessing and query costs are estimated in terms of number of comparisons in the former case, and in terms of number of pages accessed from secondary memory in the latter case. The remainder of this section discusses a comparative study of MAT and inverted file.

A) *Preprocessing cost*

When the directory resides in the main memory, the preprocessing cost for inverted file is $P_{INV}(N, k) = 0(kN \log N)$, as given in [2]. The analysis of breadth-first top-down linearization of MAT to generate a directory shows that the preprocessing cost for MAT is, $P_{MAT}(N, k) = 0(kN \log N + kN)$, as given in [8]. It is to be noted that, in both the cases, the major cost incurred is due to the sorting of the input file. When

the directory resides in secondary memory, comparison between any two records takes, at most, two page accesses. Hence we have, when directory resides in memory, $P_{INV}(N, k) = 0(N \log N)$, and $P_{MAT}(N, k) = 0(N \log N + kN)$. For MAT factor kN is the cost incurred in filling up the appropriate fields in the directory. Refer to [8] for the details. We note that the preprocessing cost is almost the same in both cases.

B) *Storage cost*

From [2, 8], we note that, when the directory resides in main memory, $S_{INV}(N, k) = 0(kN)$ and $S_{MAT}(N, k) = 0(kNc) = 0(kN)$, where c is a constant and gives number of fields (indices) needed to represent a MAT node (normally $c = 3$). When the directory is stored in secondary storage the number of secondary pages needed are given by $S_{INV}(N, k) = 0(kN/P)$, and $S_{MAT}(N, k) = 0(kN/P)$, where P is the page size. Here, we note that the storage cost is the same in both cases.

C) *Query cost*

Partial match query is taken to be yardstick for comparison of performances. However, a thorough analysis calls for considering all possible generic query types. The worst-case complexity measures are estimated in the following lemmas, which establish the superior features of MAT. Arguments are given to illustrate the advantages of MAT over inverted file in an average case.

In the *breadth-first* top-down linearization of MAT, the nodes are numbered in a breadth-first manner at any level. The members of any filial set are ordered, and are consecutive in the memory. A *partial match* query specifies attribute values for some levels of MAT, and these levels are called *specified-levels*. The other levels are called *unspecified-levels*. A node is called a *qualified-node* (a) it belongs to a specified-level and has the same value as specified in the query, or (b) belongs to an unspecified level, and has an ancestor in the nearest specified level which is a qualified-node at that level. The filial-set at level j, whose parent is a qualified-node at level $(j\text{-}1)$ is called a qualified-filial-set at level j. The dummy root is always taken as a qualified-node and level 0 as a specified-level.

The search algorithm descends down the MAT, level by level, collecting qualified-nodes at each level. For any specified-level j, binary search is carried out on all the qualified-filial-sets of level j, and the qualified nodes are collected and stored. For any unspecified-level j, all the members of all the qualified-filial-sets are collected and stored. At the final level the record pointers are retrieved.

For an inverted file, the specified attribute values are searched in the corresponding inverted list, and the qualified record pointers are retrieved. The intersection of the qualified record pointers is carried out to find out the record pointers that satisfy the query specifications.

Now, we present a worst-case analysis of these methods, and a discussion on the average case analysis.

1. *Worst-case analysis.* We denote the number of attributes specified in a partial match query by t, and the set of attributes specified by $A(|A| = t)$. The query times for inverted file and MAT are denoted by $Q_{INV}(N, k)$ and $Q_{MAT}(N, k)$ respectively. When the directory resides in main memory, these quantities denote the number of comparisons, and when the directory resides on secondary memory these quantities denote the number of pages accessed. The following lemmas establish query complexities of inverted file and MAT.

LEMMA 1:

$Q_{INV}(N, k) = 0(t \log tN^{1-1/k})$, when the directory resides in main memory.

Proof: For attribute $j \in A$, the cost of searching the inverted list is $\log(s_j)$, and the number of qualified record pointers is $0(N/s_j)$. The cost of finding intersection of t lists, each of size (N/s_j) is $\log(t) \sum_{j \in A} (N/s_j)$. Hence, the total search cost is $\sum_{j \in A} \log(s_j) + \log(t) \sum_{j \in A} (N/s_j) = \log(\prod_{j \in A} s_j) + \log(t)N \sum_{j \in A} (1/s_j)$. Using expressions (1) and (2), we conclude that $Q_{INV}(N, k) = 0(t \log tN^{1-1/k})$. □

LEMMA 2:

$$Q_{MAT}(N, k) = 0(N^{1-t/k} \log(N^{t/k}))$$

when the directory resides in main memory.

Proof: For a partial match query, in the worst-case, the first $(k\text{-}t)$ levels are unspecified, and each qualified-filial-set, for all the specified levels, contains are qualified-node. So, the number of qualified-filial-sets to be searched in any specified level is at most $s_1 s_2 \ldots s_{k-t}$. The cost of searching a filial set, at level j, is $\log(s_j)$. Hence, the query cost is given by

$$Q_{MAT}(N, k) \leq (s_1 s_2 \ldots s_{k-t})[\log(s_{k-t+1}) + \cdots + \log(s_k)].$$

Using expressions (1) and (2), we have, $Q_{MAT}(N, k) = 0(N^{1-t/k} \log(N^{t/k}))$. □

Lemmas 1 and 2 indicate that, as more and more attributes are specified in the query, $Q_{MAT}(N, k)$ approaches $0(\log(N))$, whereas $Q_{INV}(N, k)$ increases slightly, for large N. The comparison of both methods is summarized in the following theorem.

THEOREM 1:

For a partial match query the search cost in MAT is much smaller than the search cost in inverted file, for large N and $t > 1$, when the directory resides in main memory.

Proof: For $t > 1$, we have, from lemmas 1 and 2

$$Q_{MAT}(N, k) = 0(N^{1-1/k} \log(N^{t/k})/N^{(t-1)/k}),$$

$$Q_{INV}(N, k) = 0(t \log tN^{1-1/k}, \text{ and}$$

$$Q_{MAT}(N, k)/Q_{INV}(N, k) = 0(\log(N^{t/k})/(t \log tN^{(t-1)/k})).$$

We note that as N increases, this ratio $Q_{MAT}(N, k)/Q_{INV}(N, k)$ approaches zero quite rapidly, and hence, the theorem. □

Theorem 1 illustrates the superiority of MAT structure in terms of worst-case analysis. The following lemmas estimate the complexity when the directory resides on secondary memory. In the analysis the page size, P, is assumed to be greater than $c \max_j (s_j)$, where c is number of indices (integers) needed to represent a MAT node. Hence, searching for an attribute value in any filial set takes at most two page accesses.

LEMMA 3:

$Q_{INV}(N, k) = 0(tN^{1-2/k})$, when the directory resides in secondary memory.

Proof: The number of pages for each inverted list is $N/P \cdot s = 0(N/s) = (N^{1-1/k})$. Each specified attribute will involve at most $0(N/s^2) = 0(N^{1-2/k})$ pages accesses. Hence, we have $Q_{INV} = 0(tN^{1-2/k})$. □

LEMMA 4:

$Q_{MAT}(N, k) = 0(tN^{1-t/k})$, when the directory resides in secondary memory.

Proof: The number of filial sets at any level j is $\prod_{i=1}^{j-1} s_i$. For a partial match query, in the worst-case, the collection of qualified-nodes at the top $(k\text{-}t)$ levels involves $\sum_{j=1}^{k-t} \prod_{i=1}^{j} s_i$ page access. For level j, $j \in A$, the number of qualified-filial-sets to be searched is $\prod_{i=1}^{k-t} s_i$. Hence, the total number of page accesses is given by, $Q_{MAT}(N, k) = 2(\sum_{j=1}^{k-t} \prod_{i=1}^{j} s_i + t \prod_{i=1}^{k-t} s_i) = 0(t \prod_{i=1}^{k-t} s_i) = 0(tN^{1-1/k})$. □

Lemmas 3 and 4 indicate, again, that as more and more attributes are specified, $Q_{MAT}(N, k)$ decreases rapidly, whereas $Q_{INV}(N, k)$ increases slightly, for large N.

THEOREM 2:

For a partial match query the number of pages accessed in MAT is much smaller than the number of pages accessed in inverted file, for large N and $t > 2$, when the directory resides on secondary memory.

Proof: For $t > 2$, we have, from lemmas 3 and 4,

$$Q_{MAT}(N, k) = 0(tN^{1-2/k}/N^{(t-2)/k}),$$
$$Q_{INV}(N, k) = 0(tN^{1-2/k}), \text{ and}$$
$$Q_{MAT}(N, k)/Q_{INV}(N, k) = 0(1/N^{(t-2)/k}).$$

We note that as N increases, the ratio, $Q_{MAT}(N, k)/Q_{INV}(N, k)$ approaches zero quite rapidly, and hence, the theorem. □

The essence of the theorems 1 and 2 is that the MAT structure is superior to inverted file structure, when more than two attributes are specified in a query, in terms of worst-case performance.

2. *Average case analysis.* The analysis of average case performance of either structure is difficult. This is due to two factors: (a) characterization of 'an average query' is difficult, (b) mathematics involved in estimation process tend to be inconclusively complicated. However, here, we present intuitive arguments to establish the superiority of MAT structure. The main point of focus is the growth of the search part of MAT as the search progresses. It can be easily seen that, as more and more attributes are specified, the part of the tree searched gets pruned, in the average case also, giving rise to a fewer number of filial sets to be searched. The specification, in a query, of the attributes at higher levels of MAT will, on an average, prune the number of nodes to be searched. Hence, placing the frequently-occurring attributes at higher levels of the tree will ensure good average case performance. This feature makes MAT structure more effective than inverted file structure for the bibliographic files. This is because, in an inverted file-based organization, all the attributes are treated equally, and the probabilistic properties of attributes are not used to any advantage. In fact, this sort of 'probabilistic ordering' of attributes makes MAT superior to the *k-d* tree, as was shown by Veni Madhavan[7]. Hence, the ranking of attributes is guaranteed to, even in the average case, make MAT more efficient than the inverted file organization. Thus, it can be concluded, based on the above discussion, that MAT is better than inverted file organization for bibliographic file systems.

5. AN EFFICIENT ADAPTATION OF MAT

The abstract MAT structure is linearized to be stored in the form of a directory for ease of access. This idea is very similar to the array representation of a binary tree. There are two basic methods of linearization: the *breadth-first* and the *depth-first*. In the *breadth-first* method, the nodes are numbered in a breadth-first manner starting from the root, and traversing level by level. In the *depth-first* method, the nodes are numbered in depth-first manner starting from root, and traversing from left to right. The method of linearization influences, to a large extent, the average response time to the queries. Figure 2 depicts both these methods of linearization: the numbers by the side of the nodes in Fig. 2b and Fig. 2c indicate the node number in the directory. Depth-first linearization is adopted in [4, 5]. In [8], a depth-first linearization is employed and an efficient range search algorithm and a dynamization technique are developed. The search algorithm in Section 4, for partial match, is in the lines of [8].

The specific properties of bibliographic files can be made use of to arrive at an efficient adaptation of MAT structure.

A. *Fixing the attributes*

Fixing of attributes decides the structure of the MAT and hence the performance of the search algorithms. The author name is split into three (or more) parts as first name, second name, and family name, and each is treated as an attribute. The important and frequently encountered words and phrases are chosen out of the titles to be used as attributes. Others, like the journal name, volume number, month, year, etc. will form the other attributes.

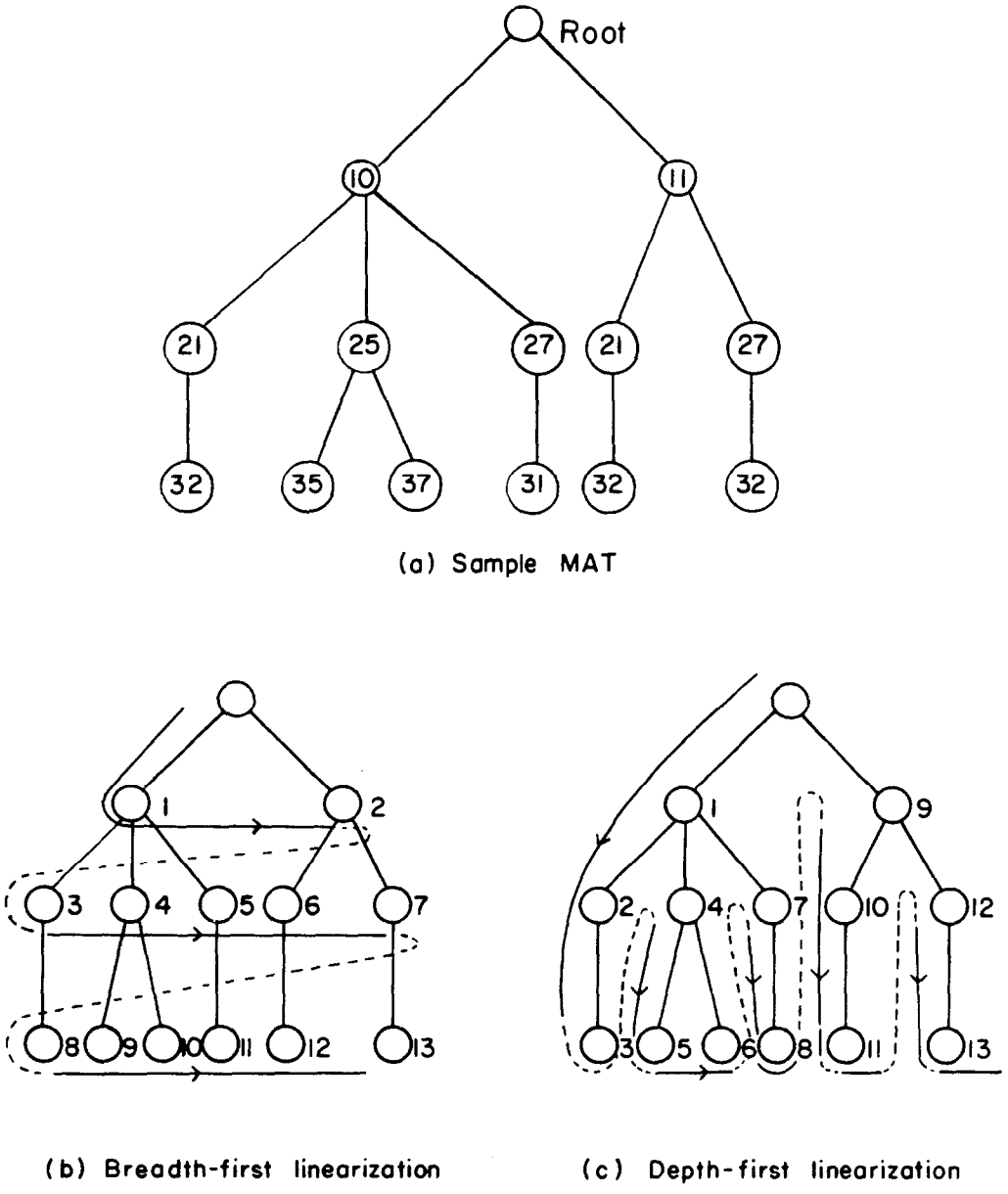


Fig. 2. Linearization.

B. *Ranking the attributes*

The attributes are ranked as follows:

(a) The authors'/editors' first name, which is most frequently specified in a query, is taken to be the first level attribute.

(b) The key-words and phrases from the titles can be ranked according to their frequency of occurrence in the queries. They rank next to authors' first name.

(c) Other attributes like journal name, publisher name, authors' other names, etc. can be ranked next to attributes chosen as in (b).

C. *Clustering effect*

The clustering effect is exploited in the breadth-first linearization, where the tree nodes that have the same parent are kept ordered in consecutive locations in the memory. In many real-life data bases, the directory and the data both reside in the secondary memory, and the access is carried out in terms of pages. The search for the records

proceeds by descending down the tree by collecting all the qualified nodes. Maintaining all the tree nodes of the same parent in consecutive locations facilitates the search for qualified nodes, when the search is carried out within nodes of the accessed pages.

Search algorithms for other types of queries that occur in bibliographic file systems can be arrived at on the lines of [8]. The dynamization technique of [8] can be employed with suitable modifications to ensure good response times in volatile file environments.

6. AN ILLUSTRATIVE EXAMPLE

The data file corresponding to the MAT of Fig. 2a is shown in Fig. 3a. Figure 3b shows the corresponding inverted file organization. The first attribute may correspond to the authors name, and the second and third may correspond to the title (or a few keywords) of the article, and journal name, respectively. In a typical bibliographic query, the author name and the journal name may be most often specified, and long titles (or keywords) may not be completely remembered. For example, query specifies the first attribute to be 10, and the third query to be 32. The search process in MAT, using breadth-first linearization, is as follows:

Finally, node 8 corresponding to first record gets qualified. The total number of nodes traversed is 1 + 3 + 1 = 5.

level	nodes qualified	nodes searched
1	1	–
2	3	3, 4, 5
3	8	8

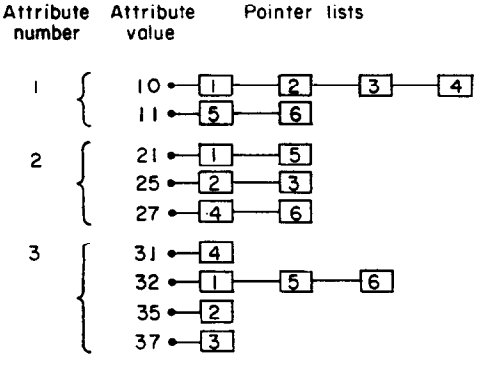
The same is achieved using inverted lists as follows:

attribute number	qualified pointers
1	1, 2, 3, 4
2	all
3	1, 5, 6

The intersection of the first and third lists gives the answer to be 1, which corresponds to the first record. The total number of pointers traversed is 4 + 3 = 7, and the steps involved in finding the intersection is 4 + 3 = 7. These examples illustrate how the tree structure prunes the search process.

A_1	A_2	A_3	Record Pointer
10	21	32	1
10	25	35	2
10	25	37	3
10	27	31	4
11	21	32	5
11	27	32	6

(a) Data file for MAT of Fig. 3(a)



(b) Inverted file for data in (a)

Fig. 3. Inverted file organization.

7. CONCLUSIONS

Various properties of MAT and inverted file structures are discussed. The specific suitability of the MAT data structure for bibliographic files is established. In particular, MAT is shown to be an attractive choice compared to an inverted file. The superiority of MAT over inverted files is established using worst-case performance measures. Arguments are provided to establish average case superiority. An adaptation of MAT, using a certain manner of selecting and ranking attributes, is proposed to exploit the special features of MAT and bibliographic files.

The study of MAT data structure is of both theoretical and practical interest. Future work can be carried out on the following lines:

(1) Query processing: A query specified by the user has to be processed to get the required attribute values, in the manner required by the search procedures.
(2) Study of attribute ranking: The ranking of attributes can be studied, both theoretically and empirically, to arrive at an optimal ranking pattern.
(3) Query answering: Efficient algorithms, in terms of worst-case and average case performance, are designed to handle various queries. Special emphasis is to be laid on the response times in the case of interactive environments.
(4) Dynamization: The process of maintaining a volatile file, by suitably accomodating insertions and deletions, is called dynamization. Dynamization is achieved in [8] by keeping the newly-inserted records in a overflow region, and marking the deleted nodes. The structure is rebuilt at certain points. The complexities of the corresponding insertion and deletion algorithms are estimated in [8]. However, the criteria for arriving at the points of rebuilding can still be investigated. Also, totally new dynamization schemes can be looked into.
(5) Study of the effect of parameters: Performance of various search and dynamization algorithms with respect to various parameters like average filial set size, frequency of occurrence of attribute values, region size, other dispersion measures of overflow nodes, etc. can be studied. This aids the understanding of the MAT data structure.
(6) Analysis of MAT: Average case analysis of various search and dynamization algorithms, taking into account various query and data properties, will provide a deeper understanding of the MAT data structure.

Acknowledgements—The authors wish to thank the referees for their comments on the earlier version of the study.

REFERENCES

[1] D. E. KNUTH, *The Art of Computer Programming*, Vol. 3: *Sorting and Searching*, Addison-Wesley, Reading, MA (1973).
[2] J. L. BENTLEY and J. H. FRIEDMAN, Data structures for range searching, *ACM Comput. Surveys* **11**(4), pp. 397–409 (1979).
[3] M. H. OVERMARS, *The Design of Dynamic Data Structures, Lect. Notes in Compt. Sci.*, Vol. 156, Springer-Verlag, Berlin (1984).
[4] R. L. KASHYAP, S. K. C. SUBAS, and S. B. YAO, Analysis of multiple attribute tree database organization, *IEEE Trans. Softw. Eng.* **SE-2**(6), pp. 451–467 (1977).
[5] V. GOPALAKRISHMA and C. E. VENI MADHAVAN, Performance evaluation of attribute-based tree organization, *ACM Trans. Database Syst.* **6**(1), pp. 69–87 (1980).
[6] J. L. BENTLEY, Multidimensional binary search trees used for associative searching, *Commun. ACM* **18**(9), pp. 509–517 (1975).
[7] C. E. VENI MADHAVAN, Secondary attribute retrieval using tree data structures, *Theor. Comput. Sci.*, in press.
[8] S. V. NAGESWARA RAO, C. E. VENI MADHAVAN, and S. SITHARAMA IYENGAR, Range search and dynamization algorithms in multiple attribute tree, Tech. Report, School of Automation, Indian Institute of Science, Bangalore, India, July 1984.