Theoretical Computer Science 62 (1988) 251-266
North-Holland

AN AVERAGE-CASE ANALYSIS OF MAT AND INVERTED FILE

Nageswara S.V. RAO* and S.S. IYENGAR

Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, U.S.A.

R.L. KASHYAP

Department of Electrical Engineering, Purdue University, West Lafayette, IN 47907, U.S.A.

Communicated by M. Nivat
Received February 1987
Revised September 1987

Abstract. It is shown in the literature that the multiple attribute tree outperforms inverted files in terms of worst-case complexities for partial-match queries. In this paper, we estimate the expected values for the complexities of both complete match query and range query on multiple attribute tree and inverted files. We use a uniform probabilistic model for the input data space. We show that the multiple attribute tree is more efficient than the inverted file in terms of these expected value measures.

1. Introduction

Traditionally, inverted files have been applied in many information retrieval systems. Recent advances in multidimensional data structures and related algorithms have resulted in efficient methods to store and retrieve information. But the inverted files continue to dominate in the commercial information retrieval systems [11]. This could be attributed, at least in parts, to the lack of direct rigorous studies about the relative performance of the tree structures and inverted files.

The multiple attribute tree has been often shown to perform better than the inverted files in many situations [4, 6, 10]. Kashyap et al. [6] show that MAT fares better than inverted file in terms of total access time in physical database environments. Gopalakrishna and Veni Madhavan [4] compare the performance of MAT and inverted file organizations using six real-life databases; they establish careful tradeoffs in terms of storage and access times for directory and data, query complexities, and database characteristics. Rao et al. [10] have proven that the worst-case complexity of a partial match query on MAT is exponentially better than that using the inverted file. These results motivate the comparison of the MAT with inverted file in terms of *average-case* performance. However, such an analysis is a very difficult task. This is partly because of the difficulty involved in specifying the average-case measures (of MAT structure and query). Furthermore, the mathematics

* Present affiliation: Department of Computer Science, Old Dominion University, Norfolk, VA 23529-0162, U.S.A.

involved in a complete rigorous characterization could be too complicated to give rise to very conclusive results.

In this paper, we assume a simple uniform probabilistic model for the input. We then establish the superiority of MAT over inverted file in terms of expected complexity values for complete match and range queries. We estimate the expected cost of complete match and range queries on MAT and inverted file organizations. Then, we prove that the MAT based method is more efficient in terms of these cost estimates. In terms of the approach, our work is similar to that of Flajolet and Puech [3]. They estimate the expected values for complexities of partial match queries in data structures such as *k-d* trees, quad trees, multiple attribute trees, etc. Here, we focus on the range query and also on the relative performance of the MAT and inverted files.

The organization of this paper is as follows: We develop the basic concepts of MAT in Section 2. The linearization is discussed in Section 3. A uniform probabilistic model for the input data space is introduced in Section 4, and the corresponding MAT parameters are evaluated in Section 5. The costs of range and complete match queries on MAT are estimated in Sections 6 and 7 respectively. These cost estimates for inverted file are obtained in Section 8. In Section 9, the MAT is shown to outperform inverted file in terms of expected cost values for the complete match and range queries.

2. Multiple Attribute Tree

In this section, we discuss the definition and structural properties of MAT. The Multiple Attribute Tree was first proposed by Kashyap et al. [6]. The MAT is formally defined as follows [10].

Definition. A k-dimensional MAT on k attributes $A_1, A_2, \ldots, A_k$ for a set of k-dimensional points (records), called the *input-set*, is a tree of depth k, with the following properties:

(i) there is a root at level 0;

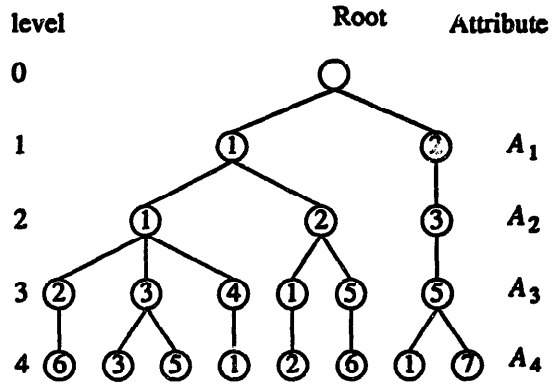
(ii) each child of the root is a $(k-1)$-dimensional MAT on the $(k-1)$ attributes $A_2, A_3, \ldots, A_k$ for the subset of records that have the same A_1 value. This value of A_1 is the value for the root of the corresponding sub-MAT, and

(iii) the child nodes of the root are in the ascending order of their values. This set is called the *filial-set.*

Figure 1(b) shows the MAT for the set of records of Fig. 1(a). We observe that there is a root at level 0 which does not have a value. The level i of the MAT corresponds to the attribute A_i. Thus the attributes $A_1, \ldots, A_k$ form the hierarchy of the levels 1 through k of the tree. There are many data structures that are based on this notion. The Doubly Chained Tree of Sussenguth [12], and Cardenas and Sagamang [2], the BST-complex of Lien et al. [8], the MDBT of Ouskal and

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

(a) Input data-set



(b) MAT data structure for the data-set of (a)

Fig. 1. Sample data and the corresponding MAT.

Scheuermann [9], kB-tree of Gutting and Kriegal [5], and kB^+-tree of Kriegal [7] are some examples. These data structures differ from one another in the way the filial-sets are represented.

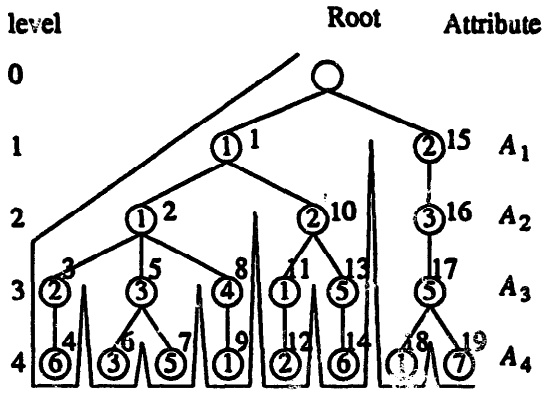
3. Structural properties and linearization of MAT

In the definition of MAT the attribute values are assumed to be chosen from a totally ordered set. One of the most important properties of MAT is the size of a filial-set. In general, this size may vary from 1 to the cardinality of the input-set. We characterize the structural properties of MAT using the following notation:

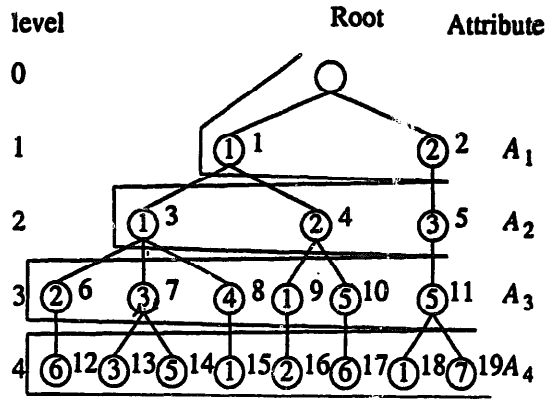
- N: number of records or equivalently the number of terminal nodes in the MAT;
- M: total number of nodes in the MAT;
- k: number of attributes or fields;
- s_j: random variable representing the size of a filial-set at level j of MAT.

The exact profile of the MAT depends on the set of input records. In general, s_j varies along the breadth of the MAT and also along the levels of MAT, apart from varying with the input data.

The MAT data structure is represented in a linearized form to facilitate the process of answering queries [6]. Let the set of all child nodes of a nonterminal node of MAT be called the *child-set*. The process of linearization corresponds to representing the MAT in an array form so that the node accesses can be implemented as array operations. Such a representation is shown to be more efficient than the pointer implementation in Section 6. There are two types of linearizations—depth-first and breadth-first. In depth-first linearization the nodes of the MAT are numbered in a depth-first manner, and in breadth-first linearization the nodes are numbered in a breadth-first manner. Figure 2 illustrates these concepts. A *breadth-first linearized*



(a) Depth-first linearization



(b) Breadth-first linearization

Fig. 2. The idea of linearization.

MAT is an array of M records, each MAT node T is represented as a record of the following type:

```
directory-element = record
                      value: value type;
                      first-child: 1..M;
                      last-child: 1..M;
                    end;
```

where the fields are defined as follows:

- *value*: value of the node T;
- *first-child*: the node number of the first node of the child-set of T;
- *last-child*: the node number of the last node of the child-set of T.

Table 1 shows the breadth-first linearization of the MAT of Fig. 1(b). The most important aspect of linearization is the ability to carry out a binary search for any value in a filial-set. The complexity of this process in $O(\log s)$ as opposed to the complexity of $O(s)$ in a pointer based implementation, where s is the size of the filial-set. This is particularly advantageous in answering a range query. However, dynamic insertions and deletions, which can be very easily accommodated in a pointer based MAT, cause severe modification in a linearized MAT. Thus, linearized MAT is best suited for static environments.

Table 1. Breadth-first linearized MAT represented in array form.

Node-number	Value	First-child	Last-child
1	1	3	4
2	2	5	5
3	1	6	8
4	2	9	10
5	3	11	11
6	2	12	12
7	3	13	14
8	4	15	15
9	1	16	16
10	5	17	17
11	5	18	19
12	6	4	-
13	3	1	-
14	5	6	-
15	1	2	-
16	2	2	-
17	6	7	-
18	1	8	-
19	7	5	-

4. Probability distribution of data

The data are a set of records (called the *input-set*) and each record is specified by k fields or attributes. Each attribute assumes values from a "discrete strict order" set of cardinality d. Such a set can be mapped onto the interval $[1, d]$ consisting of only integer values. To simplify our discussion, we assume that the attribute values are integers from $[1, d]$ (without loss of generality). Hence, any record can be viewed as a point in the k-dimensional *input-space* formed by $[1, d]\times\cdots\times[1, d]$. The input-set is a set of N points of the input-space.

The MAT is constructed for the records of the input-set, and the exact profile of the MAT depends on the number and the locations of these points in the input-space. Let p be the probability with which any point of input-space appears in input-set. In all there are d^k points in the input-space. Hence, the probability that there are i points in the input-set is given by

$$p^i(1-p)^{d^k-i}. \tag{1}$$

There are $C_i^{d^k}$ such sets. Now, we state a result which is repeatedly used in our discussion.

Lemma 4.1. $\sum_{i=1}^{R} C_i^R i p^i (1-p)^{R-i} = pR$ *for a positive integer* R *and a fraction* p $(\leqslant 1)$.

Proof. See Appendix. □

Now, the expected value for the cardinality of the input-set is

$$\sum_{i=1}^{d^k} C_i^{d^k} i p^i (1-p)^{d^k-i} = pd^k$$

by Lemma 4.1.

Another important aspect of multidimensional data is the notion of hyperplanes. A single attribute value $A_i = a_i$ defines a k-dimensional *hyperplane* consisting of all the records with a_i as the value for A_i. Note that the dimensionality of the space formed by this hyperplane is $k-1$. Similarly, a $(k-j)$-dimensional hyperplane can be defined by the sequence of attribute values $A_1 = a_1$, $A_2 = a_2, \ldots, A_j = a_j$. Let us denote this hyperplane by $\langle a_1, a_2, \ldots, a_j\rangle$.

In the next sections, we estimate the expected values for range and complete match queries on MAT and inverted files.

5. Performance parameters of MAT

The complete profile of the MAT for a given input-set of records is defined by the sizes of various filial-sets. In general, the size of any filial-set is random, and can vary from 1 to d. Thus a complete characterization of the exact sizes of filial-sets

is very involved. However, we restrict our discussion to the expected value of a filial-set of any level.

Consider a node at level 1 with a value a_1. The probability that such a node exists in MAT is equal to the probability that any point on the hyperplane $\langle a_1 \rangle$ appears in input-set. This probability is given by

$$\begin{aligned} q_1 &= 1 - \text{probability that no point of } \langle a_1 \rangle \text{ appears in input-set} \\ &= 1 - (1-p)^{d^{k-1}}. \end{aligned} \tag{2}$$

The attribute may assume an integer value a_1 from the interval $[1, d]$ with a probability of q_1. Thus, the probability that there are exactly i nodes in the filial-set at level 1 is $(q_1)^i(1-q_1)^{d-i}$. There are C_i^d possible filial-sets of size i. The expected value for the size of filial-set at level 1 is given by

$$\mathrm{E}[s_1] = \sum_{i=1}^{d} C_i^d i q_1^i (1-q_1)^{d-i}.$$

Using Lemma 4.1 we have

$$= q_1 d$$

and using equation (2)

$$= d[1-(1-p)^{d^{k-1}}]. \tag{3}$$

Now, consider a node at level j, $j = 1, 2, \ldots, k$. Any such node is defined by the attribute values given by $A_1 = a_1$, $A_2 = a_2, \ldots, A_j = a_j$. The probability q_j of occurrence of any such node is equal to the probability that any node on the hyperplane $\langle a_1, a_2, \ldots, a_j \rangle$ appears in the input-set. Hence,

$$\begin{aligned} q_j &= 1 - \text{probability that no point of } \langle a_1, a_2, \ldots, a_j \rangle \text{ appears in input-set} \\ &= 1 - (1-p)^{d^{k-}}. \end{aligned}$$

Using the same arguments as in the case of level 1, we obtain the expected value for the size of filial-set of level j as

$$\mathrm{E}[s_j] = d[1-(1-p)^{d^{k-j}}]. \tag{4}$$

The expected values for the filial-set sizes developed in this section are used in the next sections to evaluate the complexities of complete match and range queries.

6. Analysis of range query on MAT

A *range query* on a k-dimensional data is given by $Q_R = \bigcap_{i=1}^{k} q_i$, where $q_i = [l_i, h_i]$ specifies the range of values for the attribute A_i. Geometrically speaking a range query specifies a rectilinearly oriented hyperrectangle. We term this hyperrectangle

as a *qucry-rectangle.* The points enclosed by this rectangle belong to $[l_1, h_1] \times \cdots \times [l_k, h_k]$. Answering a range query calls for the retrieval of the points enclosed by the query-rectangle. The records or points that satisfy a given range query depend on

(i) the dimensions of the query-rectangle,
(ii) its location inside the input-space,
(iii) the profile and distribution of the input-set.

Since a range query specifies a portion of input space, the points inside the query-rectangle inherit their properties from the input-space. As we assumed uniform probabilistic model for the input-space, the probabilistic nature of the query-rectangle is uniform across the input-space. Thus, as a consequence of this result, the probabilistic nature of the query depends only on the dimension of the query-rectangle and is independent of the location of the query-rectangle in the input-space. In our analysis, we consider a "uniform" range query that specifies an interval of size a ($\leq d$) for all attributes.

The answers to a range query are produced by descending down the MAT level by level starting from the root. At any level j, all the nodes that satisfy the "partial" query $\bigcap_{i=1}^{j} q_i$ are retrieved. These nodes are called the *qualified nodes* of level j. In the next level $(j+1)$, the child nodes of the qualified nodes at level j are checked for inclusion in the interval $[l_{j+1}, h_{j+1}]$. The details are presented in algorithm RANGE-SEARCH.

Algorithm RANGE-SEARCH(*level, qset*);
begin
(1) *tempest* $\leftarrow \emptyset$;
(2) **for** each $n \in$ *qset* **do**
    begin
(3) **for** each child node n_1 of n **do**
(4) **if** value of n_1 lies in the range $[l_{level}, h_{level}]$
(5) **then** add n_1 to *tempest*;
    end;
(6) **if** ($level \neq k$)
(7) **then** RANGE-SEARCH($level+1$, *tempest*);
end;

The algorithm RANGE-SEARCH is initialized with $level = 1$ and $qset = \{\text{root}\}$. At any level *tempest* collects the qualified nodes of that level. Using the breadth-first linearized MAT, the process of retrieving qualified-nodes of any level is carried out as follows: Let n be a node qualified at level j. Let s be the size of the child-set of n at level $(j+1)$. On the child-set of n, two binary searches are carried out for the range limits l_{j+1} and h_{j+1}. All the nodes that lie within these limits are retrieved as qualified nodes. In the final level the information about the records that satisfy the given query is retrieved.

In the remainder of this section, first we establish that the linearized MAT fares better than the MAT implemented with points. Later we estimate the expected number of nodes accesses incurred in answering a range query on a linearized MAT.

In answering a range query, the number of filial-sets searched at level $j+1$ is equal to the number of nodes qualified at level j. Consider a node n qualified at level j. Let the ordered child-set of n at level $(j+1)$ be $\{v_1, v_2, \ldots v_s\}$. Each child node v_i is checked for inclusion in the interval $[l_{j+1}, h_{j+1}]$. The nodes that lie in this interval are consecutive in the child-set as a result of the ordering imposed on any filial-set. Let r ($\leq s$) be the number of qualified nodes. In MAT, the processing of a child-set of a qualified node is carried out as follows: The nodes $\{v_1, v_2, \ldots, v_s\}$ are searched sequentially starting with v_1. The search ends when a node with its value greater than h_{j+1} is encountered. Complexity of such a sequential search is $O(s)$. In a linearized MAT, the complexity of processing a child-set is $O(\log(s)+r)$. The first term corresponds to the searching for the range limits, and the second term corresponds to the retrieval of the qualified records. Comparing these two approaches it is easy to see that in a general case the linearized MAT fares better.

The complexity of a range search on the MAT is estimated in terms of the nodes accessed in answering a range query. There are two types of costs involved in answering a range query on a MAT:

(a) The number of nodes accessed in searching for range limits in various filial-sets. This factor depends on the structure of the MAT, i.e., the exact sizes of the filial-sets which are searched.

(b) The number of nodes that lie within the specified range limits in any filial-set. This factor depends on the nature of the query, i.e. the exact range of limits specified by the query.

Let S_j be the expected value for the total number of nodes accessed during the search operation for the limits at level j. Let N_j be the expected value for the total number of nodes qualified at level j. The expected value for the size of filial-set at level j is $d[1-(1-p)^{d^{k-j}}]$ as per equation (4). Hence, we have

$$S_j = N_{j-1}(2\log(d[1-(1-p)^{d^{k-j}}])).$$

Consider the range specified for the attribute A_1. As stated earlier in the section, the range query specifies a range of a values for each attribute. The probability that a node at level 1 with a value $b \in [l_1, h_1]$ appears in MAT is equal to the probability that a point on the hyperplane $A_1 = b$ appears in the input-set. This probability is equal to q_1 given by (3). Using the same arguments as in Section 5, the expected value for the number of nodes that fall into the specified range is given by $N_1 = a[1-(1-p)^{d^{k-1}}]$.

Now, consider a filial-set at level j. By using the same argument as above, we obtain the expected value for the number of nodes that lie within the specified range limits to be $a[1-(1-p)^{d^{k-j}}]$. We define the *qualify-fraction* of a range query with respect to a filial-set to be the ratio of the expected number of nodes that satisfy the query to the expected number of nodes of the filial-set. The qualify-fraction for

a filial-set at level j is

$$f=\frac{a[1-(1-p)^{d^{k-j}}]}{d[1-(1-p)^{d^{k-j}}]}=\frac{a}{d}.$$

Note that the qualify-fraction is the same for all levels of MAT. The expected number of nodes qualified at level j is given by

$$N_j=(a[1-(1-p)^{d^{k-j}}])N_{j-1}=(fd[1-(1-p)^{d^{k-j}}])N_{j-1}. \tag{5,6}$$

We have

$$N_1=a[1-(1-p)^{d^{k-1}}]=fd[1-(1-p)^{d^{k-1}}]$$

and solving equation (6)

$$N_j=f^jd^j\prod_{i=1}^{j}[1-(1-p)^{d^{k-i}}], \tag{7}$$

$$N_k=f^kd^k\prod_{i=1}^{k}[1-(1-p)^{d^{k-i}}]=f^kd^kp\prod_{i=1}^{k-1}[1-(1-p)^{d^{k-i}}]. \tag{8,9}$$

The product $\prod_{i=1}^{j}[1-(1-p)^{d^{k-i}}]$ is very important for our discussion and the following lemma gives some of its useful properties.

Lemma 6.1

(1) $[1-(1-p)^{d^{k-(i+1)}}]\leqslant[1-(1-p)^{d^{k-i}}]$.

(2) $\prod_{i=1}^{j}[1-(1-p)^{d^{k-i}}]\geqslant\prod_{i=1}^{j+1}[1-(1-p)^{d^{k-i}}]$.

(3) $[1-(1-p)^{d^{k-j}}]\geqslant\prod_{i=1}^{j}[1-(1-p)^{d^{k-i}}]$.

(4) $q\geqslant[1-(1-p)^{d^{k-j}}]$ *for* $j=1,2,\ldots,k$, *where* $q=[1-(1-p)^{d^k}]$.

(5) $p\geqslant\prod_{i=1}^{k}[1-(1-p)^{d^{k-i}}]$.

Proof. See Appendix. □

Using part (5) of Lemma 6.1 in equation (8) we have,

$$N_k\leqslant pf^kd^k. \tag{10}$$

Again,

$$S_j=N_{j-1}2\log(d[1-(1-p)^{d^{k-j}}]),\qquad S_k=N_{k-1}2\log(pd). \tag{11,12}$$

Using these equations we develop the expected value for the complexity of the range query in the following theorem.

Theorem 6.2. *The expected number of node accesses in answering a range query is given by*

$$N_{\mathrm{MAT}}(Q_{\mathrm{R}})=\begin{cases}O(k(fqd)^{k-1}[fqd+\log(qd)]) & \text{if } (fqd)\geqslant 1,\\ O(k\log(qd)) & \text{if } (fqd)<1.\end{cases}$$

Proof. The expected value for the number of nodes that are accessed in answering a range query Q_R is given by

$$N(Q_R) = \sum_{j=1}^{k} (S_j + N_j). \tag{13}$$

Using (4) of Lemma 6.1, and equations (7) and (11), we have

$$\leqslant \sum_{j=1}^{k} (f^j q^j d^j + f^{j-1} q^{j-1} d^{j-1} 2 \log(qd))$$

$$\leqslant \sum_{j=1}^{k} (f^{j-1} q^{j-1} d^{j-1} [fdq + 2 \log(qd)])$$

$$\leqslant [fqd + 2 \log(qd)] \sum_{j=1}^{k} (f^{j-1} q^{j-1} d^{j-1}).$$

Now, we have

$$\sum_{j=1}^{k} (fqd)^{j-1} \leqslant \begin{cases} k & \text{if } (fqd) < 1, \\ k(fqd)^{k-1} & \text{if } (fqd) \geqslant 1. \end{cases}$$

Thus if $(fqd) < 1$, we have

$$N_{MAT}(Q_R) = O(k \log(qd)). \tag{14}$$

If $(fqd) \geqslant 1$, then the complexity of range query is given by

$$N_{MAT}(Q_R) \leqslant [fqd + 2 \log(qd)](fqd)^{k-1} k \tag{15}$$

$$= O(k(fqd)^{k-1} [fqd + \log(qd)]). \tag{16}$$

Hence, the theorem. □

For the condition $(fqd) \geqslant 1$ we have the following two cases.

Theorem 6.3. *For* $(fqd) \geqslant 1$,

$$N_{MAT}(Q_R) = \begin{cases} O(k(fqd)^k) & \text{if } f \geqslant \dfrac{\log(qd)}{qd}, \\ O(\kappa \log^k(qd)) & \text{if } f \leqslant \dfrac{\log(qd)}{qd}. \end{cases}$$

Proof. The proof directly follows from Theorem 6.2. □

7. Analysis of complete match query on MAT

The complete match query is a special case of the range query in which the range for each attribute specifies a single value. Stated equivalently, the query-rectangle specifies a single point in the input space. As a consequence, $N_j = 1$ in equation (13), and there is a single filial-set to be searched at any level j. The expected value for the complexity of complete match query is given in Theorem 7.1.

Theorem 7.1. *The expected value for the number of nodes accessed in answering a complete match query, Q_C, is given by*

$$N_{MAT}(Q_C) = O(\log(pd^k)).$$

Proof. The expected value for the number of nodes accessed in answering a complete match query Q_C is

$$\begin{aligned} N_{MAT}(Q_C) &= \sum_{j=1}^{k} S_j = \sum_{j=1}^{k} 2\log(d[1-(1-p)^{d^{k-j}}]) \\ &= 2\log\left(d^k \prod_{j=1}^{k} [1-(1-p)^{d^{k-j}}]\right) \\ &\leq 2\log(pd^k) \quad \text{by using Lemma 6.1(5)} \\ &= O(\log(pd^k)). \end{aligned}$$

Hence, the theorem. □

In the next section, we deal with the expected values for the complexities of complete match and range queries on inverted file based organizations.

8. Inverted file organization

In this section, we compute the expected value for the complexities of complete match and range queries for the data described in Section 4. The inverted file organization consists of k inverted lists; one inverted list for each attribute A_i. The inverted list for A_i contains a header whose entries correspond to the values of the attribute A_i. Each entry corresponding to a value $A_i = a_i$ points to the list of all record numbers which have a_i as the value of A_i. This list is sorted with respect to the record number. See [1] for more details on inverted files.

There are k inverted files. The expected size h_i for the header is computed as follows: The probability that a value a_i appears in the header of A_i is equal to the probability that any point on the hyperplane $\langle a_i \rangle$ occurs in the input-set. Thus, using the same arguments as in Section 5, we obtain

$$h_i = d[1-(1-p)^{d^{k-1}}]. \tag{17}$$

Consider a value a_i in the header. The expected number of records that have this value as the value for A_i is equal to the expected number of points in the input-set that belong to the hyperplane $A_i = a_i$. There are d^{k-1} points on this hyperplane and each has a probability p of occurring in the input-set. Thus the expected length of the list of records for any value a_i in the header of A_i (by using the arguments of Section 4)

$$pd^{k-1}. \tag{18}$$

Theorem 8.1. *The expected number of operations (either a comparison or a memory access) carried out in answering a complete match query Q_C using inverted file structure is given by*

$$N_{INV}(Q_C) = O(kpd^{k-1}).$$

Proof. Let $Q_C = \bigcap_{j=1}^{k}(A_j = b_j)$ be the given complete match query. The answer is produced by searching for the specified attribute value $A_j = b_j$ in the header of the inverted list for A_j. The corresponding list of records is retrieved. The intersection of all such lists for all attributes is computed. Since all lists are ordered, the lists can be merged to obtain the intersection. The expected value for the cost of searching in all k headers is $k \log(d[1-(1-p)^{d^{k-1}}])$. Now there are k sorted lists each having an expected length of pd^{k-1}. Hence, the expected cost of finding the intersection is $O(kpd^{k-1})$. The expected complexity of complete match query is given by

$$N_{INV}(Q_C) = O(k \log(d[1-(1-p)^{d^{k-1}}]) + kpd^{k-1}) = O(kpd^{k-1}). \tag{19}$$

Hence, the theorem. □

Theorem 8.2. *The expected number of operations (either comparisons or memory accesses) carried out in answering a range query Q_R is given by*

$$N_{INV}(Q_R) = O(kfq^2d^k).$$

Proof. Let $Q_R = \bigcap_{j=1}^{k} q_j$ be the given range query, where q_j specifies the range $[l_j, h_j]$ for the attribute A_j. The answer to the range query is produced by searching the header of A_j for the limits l_j and h_j, and retrieving lists corresponding to the values lying in the range. The cost of searching for the range limits for all attributes is $O(k \log(d[1-(1-p)^{d^{k-1}}]))$. For each attribute A_i, the expected number of values that qualify the range constraint is $fd[1-(1-p)^{d^{k-1}}] \leq qfd$ (using Lemma 6.1(4)). In all, there are $kqfd$ sorted lists each of expected size $pd^{k-1} \leq qd^{k-1}$ (since $p \leq q$ by Lemma 6.1(4)). The expected cost of finding the intersection of these lists is $O(kfq^2d^k)$. Thus the expected cost of the range query is given by

$$\begin{aligned} N_{INV}(Q_R) &= O(k \log(d[1-(1-p)^{d^{k-1}}]) + kfq^2d^k) \\ &= O(kfq^2d^k). \end{aligned}$$

Hence, the theorem. □

In the next section, we carry out a comparative study of MAT and inverted file, and establish that the former is superior.

9. Comparison of performance

There has been many efforts to evaluate the relative performances of MAT and inverted files; the former being proven efficient in many situations. Kashyap et al.

[6] and Gopalakrishna and Veni Madhaven [4] have carried out a comparative study of inverted files and MAT. Rao et al. [10] prove that MAT outperforms inverted files in terms of worst-case complexity of a partial match query. The MAT and inverted files have the same complexity of preprocessing [10]. In this section, we establish the superiority of MAT over inverted files in terms of expected value complexities for complete match and range queries.

Theorem 9.1. *The performance of MAT is better than inverted file in terms of expected value for the complexity of a complete match query for large input-sets.*

Proof. We prove this theorem by proving that the $N_{\mathrm{MAT}}(Q_{\mathrm{C}})/N_{\mathrm{INV}}(Q_{\mathrm{C}})$ approaches zero as the expected value for the input-set increases. Consider

$$\frac{N_{\mathrm{MAT}}(Q_{\mathrm{C}})}{N_{\mathrm{INV}}(Q_{\mathrm{C}})} = \mathrm{O}\left(\frac{\log(pd^k)}{kpd^{k-1}}\right) = \mathrm{O}\left(\frac{\log(pd^{k-1})}{kpd^{k-1}} + \frac{\log(d)}{kpd^{k-1}}\right). \tag{20}$$

For input-sets of large expected sizes, pd^k is large. Consequently, pd^{k-1} is also large. Thus, both terms in the above equation (20) tend to zero as we consider input-sets with larger expected sizes. Speaking in terms of exponential complexities, the log terms in the numerator increase at a slower rate than the terms in the denominator. Hence, the theorem. □

Theorem 9.2. *The number of operations needed in MAT is a fraction of the number of operations needed in inverted file in terms of the expected value for the complexity of a range query.*

Proof. We prove this claim by explicitly computing the fraction of the number of operations needed in MAT to the number of operations needed in inverted file.

Case (a): Consider the case $f \geq (\log(qd))/qd$. We have, from Theorems 6.3 and 8.2,

$$\frac{N_{\mathrm{MAT}}(Q_{\mathrm{R}})}{N_{\mathrm{INV}}(Q_{\mathrm{R}})} = \mathrm{O}\left(\frac{k(fqd)^k}{kfq^2d^k}\right) = \mathrm{O}(f^{k-1}q^{k-2}).$$

Case (b): Consider the case $f \leq (\log(qd))/qd$. We have, from Theorem 8.2,

$$N_{\mathrm{INV}}(Q_{\mathrm{R}}) = \mathrm{O}(kfq^2d^k) = \mathrm{O}\left(\frac{k\log^k(qd)}{f^{k-1}q^{k-2}}\right).$$

Thus we have

$$\frac{N_{\mathrm{MAT}}(Q_{\mathrm{R}})}{N_{\mathrm{INV}}(Q_{\mathrm{R}})} = \mathrm{O}(f^{k-1}q^{k-2}).$$

In both cases, the ratio of the expected number of operations needed in a MAT to the number of operations needed in inverted file is given by $f^{k-1}q^{k-2}$. Clearly, this is a fraction. Since both f and q are fractions, the product $f^{k-1}q^{k-2}$ is very small. Another important factor is the dimensionality of the input-space. As k increases,

the fraction $f^{k-1}q^{i-2}$ decreases rapidly. Thus the performance of the MAT data structure is increasingly efficient as the dimensionality of the input-space increases. Hence, the theorem. □

Thus, we have established the superiority of MAT over inverted file in terms of the expected value for the complexities of complete match and range queries.

10. Conclusions

In literature the MAT is proposed as a viable and efficient alternative to the inverted file for storing and retrieving multidimensional data. In this paper, we computed the expected values for the complexities of complete match and range queries on the MAT and the inverted file. We theoretically proved that the MAT is more efficient than the inverted file in terms of these performance measures. Here, we considered a uniform probabilistic model for the input space. In general, the points could be arbitrarily clustered in some regions of the input space. The studies of such cases would further clarify the relative performance of the MAT and the inverted file.

Appendix. Proofs for Lemma 4.1 and Lemma 6.1.

Proof of Lemma 4.1. Consider

$$iC_i^R = \frac{R!}{i!(R-i)!} = \frac{R!}{(i-1)!(R-1-(i-1))!} = RC_{i-1}^{R-1}.$$

Now, we have

$$\sum_{i=1}^{R} C_i^R i p^i (1-p)^{R-i}$$

$$= pR \sum_{i=1}^{R} C_{i-1}^{R-1} p^{i-1} (1-p)^{(R-1)-(i-1)} = pR(p+1-p)^{R-1} = pR.$$

Hence, the lemma. □

Proof of Lemma 6.1. Since p is a fraction, we have $p \leq 1$, and $(1-p) \leq 1$.

(1): The claim directly follows from the following inequality

$$(1-p)^{d^{k-i}} \leq (1-p)^{d^{k-(i+1)}}.$$

(2): This property follows from the fact that the product $\prod_{i=1}^{j+1}[1-(1-p)^{d^{k-i}}]$ is obtained by multiplying $\prod_{i=1}^{j}[1-(1-p)^{d^{k-i}}]$ by the fraction $1-(1-p)^{d^{k-(j+1)}}$.

(3): This property follows from the fact that the product term $\prod_{i=1}^{j}[1-(1-p)^{d^{k-i}}]$ is obtained by multiplying $[1-(1-p)^{d^{k-j}}]$ by a fraction.

(4): This property follows from part (1).

(5): This property is obtained by substituting $j=k$ in (3).

Hence, the lemma. □

References

[1] A.F. Cardinas, Analysis and performance of inverted data base structures, *Comm. ACM* **18** (1975) 253-263.

[2] A.F. Cardinas and J.P. Sagamang, Modeling and analysis of database organization—The doubly chained tree structure, *Inform. Systems* **1** (1975) 57-67.

[3] P. Flajolet and C. Puech, Tree structures for partial match retrieval, in: *Proc. 24th Ann. Symp. on Foundations of Computer Science* (1983) 282-288.

[4] V. Gopalakrishna and C.E. Veni Madhavan, Performance evaluation of attribute-based tree organization, *ACM Trans. Database Systems* **6** (1980) 69-87.

[5] H. Guting and H.P. Kriegal, Dynamic k-dimensional multiway search under time-varying access frequencies, in: *Proc. 5th GI Conf. on Theoretical Computer Science* (1981) 135-145.

[6] R.L. Kashyap, S.K.C. Subas and S.B. Yao, Analysis of multiple attribute tree database organization, *IEEE Trans. Software Engng* **SE-2** (1977) 451-467.

[7] H.P. Kriegal, Variants of multidimensional B-trees as dynamic index structure for associative retrieval in database systems, in: *Proc. 7th Conf. on Graph Theoretic Concepts in Computer Science*, Linz, Austria (1981) 109-128.

[8] Y.E. Lien, C.E. Taylor and J.R. Driscoll, Binary search tree complex—towards the implementation, in: *Proc. 1st Conf. on Very Large Data Bases* (1975) 540-542.

[9] M. Ouskal and P. Scheuermann, Multidimensional B-trees: analysis and dynamic behavior, *BIT* **21** (1981) 401-418.

[10] S.V.N. Rao, S. Sitharama Iyengar and C.E. Veni Madhavan, A comparative study of multiple attribute tree and inverted file structures for large bibliographic files, *Inform. Process. and Management* **12** (1985) 433-442.

[11] G. Salton and M. McGill, *Introduction to Modern Information Retrieval* (McGraw-Hill, New York, 1983).

[12] E.H. Sussenguth, The use of tree structures for processing files, *Comm. ACM* **6** (1963) 272-279.

[13] C.E. Veni Madhavan, Secondary attribute retrieval using tree data structures, *Theoret. Comput. Sci.* **33** (1984) 107-116.