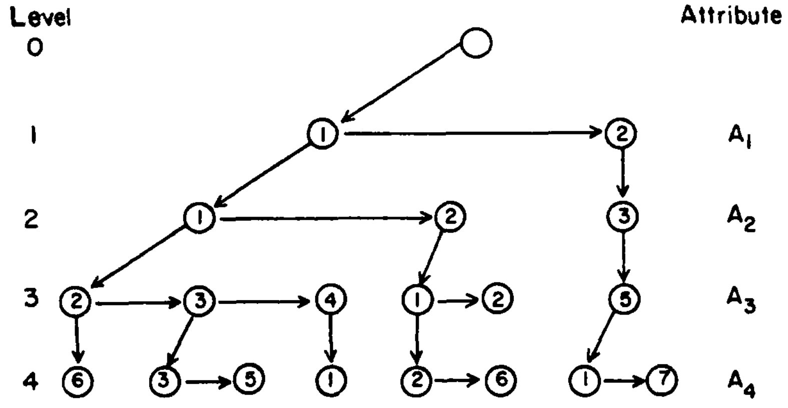

FIG. 7. Doubly chained tree for data of Fig. 6a.

BST-complex for a file of records is similar to the MAT. In the BST-complex, each filial set is maintained as a height-balanced binary tree or an AVL-tree. (In an AVL-tree the difference between the heights of the left and right subtrees of every node is at most one.) This guarantees logarithmic searching time for a complete match query. Fig. 8b shows the BST complex for the datafile of Fig. 8a. The leaves carry pointers to the page numbers. Thus, if the number of distinct key values at level i is $v(i)$, the time required for the BST-complex to respond to an exact match query is at most $\sum_{i=0}^{k-1} \log(v(i)) \leq k \log N$. Despite its good performance for complete match, the BST-complex does not efficiently support partial match and range queries.

The multidimensional B-tree (MDBT) of Ouskal and Scheuermann (1981) ensures efficient searching for various associative queries for database environments. The central idea is the use of B-trees to maintain the filial sets in a structure similar to MAT. This structure, called the MDBT, is proposed as a viable alternative for an environment with a fairly high degree of query complexity. Strategies to maintain the MDBT are also discussed in Ouksel and Scheuermann (1981). Analytical estimates for random MDBT are given by $O(\log N)$, $O(N \log N'/N')$, and $O(\log N)$ as complexities for the worst-case exact match, worst-case partial match and restricted average range query respectively. N' denotes the product of average filial set sizes of unspecified levels for a partial match. It is to be noted that the MDBT is assumed to be in an 'average' form. Again, the restricted range query is a simplified form of a generalized range query. Each attribute is assumed to specify no more than two attribute values. However, a generalized range query will have substantially higher complexity in the MDBT than $O(\log N)$.

Another tree based index structure which is based on the same concepts as the MDBT is the multidimensional B-tree (kB-tree) proposed by Gueting and

(a)

A_1	A_2	A_3	Page No.
x	1	b	1
y	1	b	2
x	2	a	3
y	2	a	4
x	2	c	5
x	3	a	6
z	2	b	7
z	1	a	8
z	3	c	9

(b)

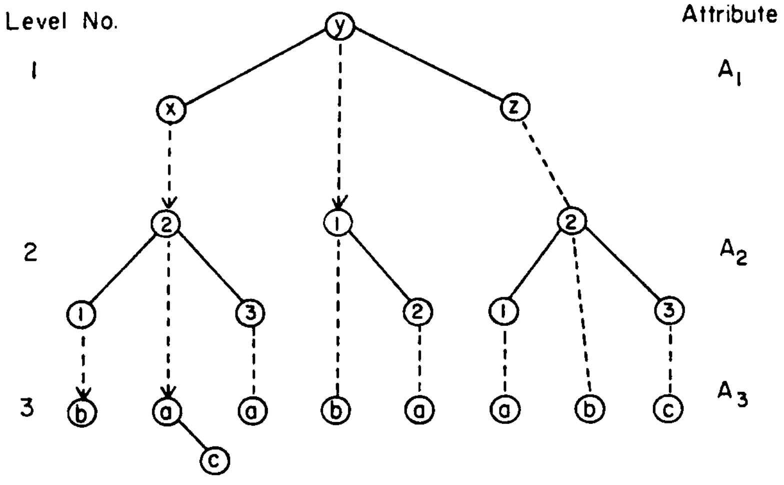

FIG. 8. (a) Input data file; (b) BST-complex for data in (a).

Kriegel (1980, 1981). Under the assumption of a uniform and independent distribution of the attributes the MDBT will have a maximal height of log $N+k$. But on the data that does not satisfy these properties, the search depth could be as much as $O(k \log N)$. The novel method of "balancing over all components" in the kB-tree ensures the depth to be $O(\log_{m+1} N + k)$, where m is the order of the kB-tree. Hence, the complete match query, insertion and deletions restricted to one path have logarithmic time complexity. We will present an example from Kriegel (1984) to illustrate the difference between the MDBT and kB-tree. Figure 9a shows two-dimensional records as points in a two-dimensional space. Figure 9b shows the corresponding MDBT. In the

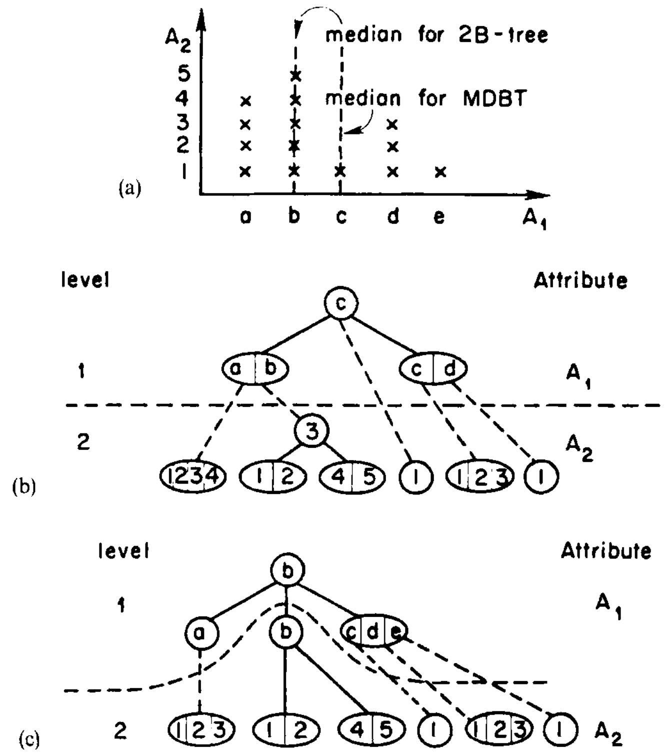

FIG. 9. (a) Input data points; (b) MDBT for data of (a); (c) kB-tree for data of (a).

MDBT each level corresponds to an attribute, and each filial-set is maintained as a B-tree at the same level. On level 1 the median of a, b, c, d, e, is c, as balancing is based on attribute 1 alone. Figure 9c shows the kB-tree. At level 1, b is selected as the median, as balancing is based on all the records. This notion of balance is shown to ensure the $O(\log N + k)$ depth for the kB-tree. Though the concept of balancing over all components can be found in Lee and Wong (1977), the main contribution of the kB-trees is their maintainability in dynamic environments. However, both the MDBT and the kB-tree have a shortcoming: neither supports sequential processing at each level. This property is very important for answering partial match and range queries. There are other refinements of these on the kB-tree such as the kB^+-tree, and $(k + 1)B^+$-tree, which are more efficient in certain ways. See Kriegel (1984) for details.

We note that the main difference between the data structures discussed above lies in the way in which the filial-sets are organized. In the MAT, the filial-sets are maintained as ordered lists which support binary search. In main memory this feature is naturally supported if the members of filial-sets are stored in consecutive locations. Consequently, searching a filial-set of size s, for a member costs almost $O(\log(s))$. The exact implications of this aspect are discussed in the next section.

6.3 Directory Generation

The data structure MAT is linearized to form a directory. The directory defines the way various levels are organized and the way the filial sets are maintained. The data structures such as the MAT, DCT, MDBT, kB-tree, $k\mathrm{B}^+$-tree, etc. can be implemented using pointers. Answering a query calls for the traversal of pointers. In general, no assumption is made on the way the pointers are organized. This type of implementation using pointers is efficient when the data structure resides in the main memory. But when the directory resides in secondary memory, where the transfer of information is in the form of pages, the randomness of the pointer organization results in the retrieval of many unnecessary pages. For example, let us consider the case in which the nodes of a filial set reside on separate pages in secondary memory. This situation might arise when the filial set is generated by dynamic insertions of records after the static structure is constructed. In this case, the searching for a value in a filial set of size s is likely to result in $O(\log N)$ retrievals of secondary memory pages. But instead, if a filial set is maintained on a single page or on a constant number of pages, then the search involves $O(1)$ retrievals of secondary pages. Once a secondary page is loaded into main memory, search can be carried out on the filial set for the required value. This situation arises in physical database organizations where the files physically reside on secondary storage. The other applications may include graphics, computational geometry, etc., where the large sizes of data necessitate storage of information on secondary memory.

The main driving force for the MAT-based organization is secondary memory storage. Though the abstract structure MAT can be implemented using pointers, we consider an organization where the nodes corresponding to a filial set are kept together. The main idea is to cluster the nodes that are likely to be accessed in answering a query. In a complete match query, the nodes of a filial set are searched for a value. In a partial match query, either the filial set is searched for a value, or the whole of the filial set is retrieved. In a range query, the nodes within the specified range are retrieved. In order to answer the above queries, we introduce the concept of linearization, which enables us to store the MAT data structure in the form of a table. There are two basic types of linearization as proposed by Subas and Kashyap (1975) and Kashyap *et al.* (1977): depth-first linearization and breadth-first linearization. Either can be done in a top-down or bottom-up manner. The top-down method is more efficient than the bottom-up method. This is because, in answering a query, the top-down method results in reducing the number of nodes to be searched. In the depth-first organization, the nodes corresponding to the fields of a record are kept together, as illustrated in Fig. 10a. The numbers by the side of the node represent the entry number of the nodes in the directory. This is an efficient technique for full match queries, as in the best case the answer can be produced in a single access to secondary storage (assuming k nodes can be stored in a single page). However, in the worst case, k accesses are needed. But this does not naturally support the partial match and range queries, as members of a filial set are not kept together. In the breadth-first linearization, the nodes corresponding to a filial set are kept together as shown in Fig. 10b. This method naturally supports the partial match and range queries.

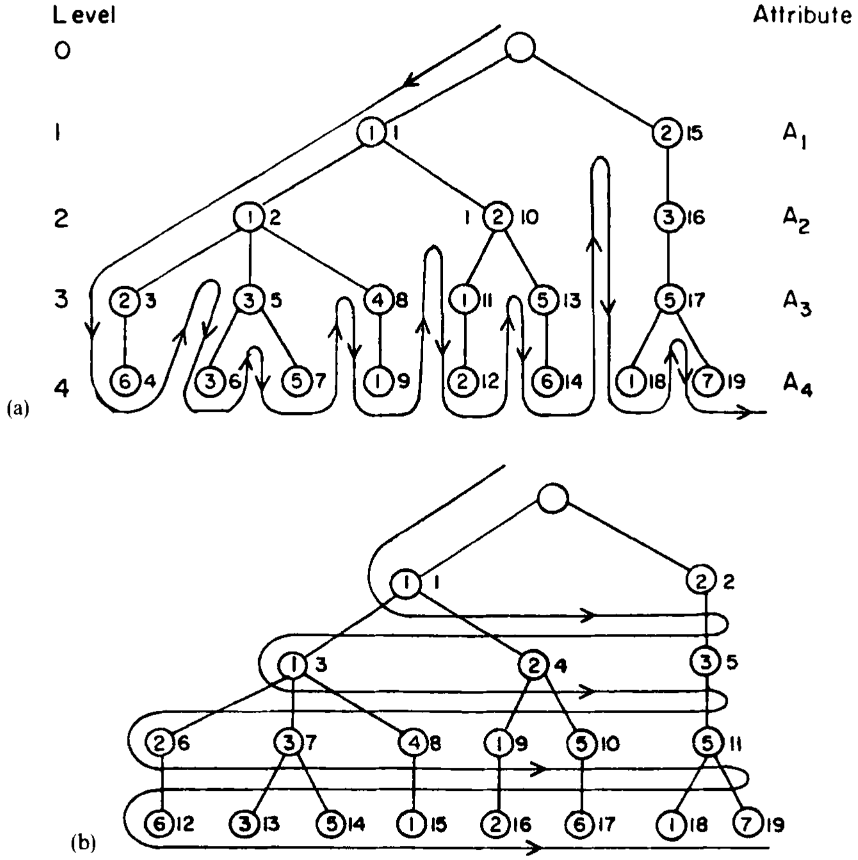

FIG. 10. (a) Depth-first linearization; (b) breadth first linearization.

Figures 11a and 11b show the directories corresponding to the depth-first and breadth-first linearizations, where each directory element is of the following form:

```
            directory element = record
                  node-num: 1..M;
                     value: 1..M;
      first-child/brother: 1..M;
        last-child/cousin: 1..M;
                        end;
```

where the fields corresponding to a node T, at level j, and numbered n are defined as follows:

```
       value: value of node T;
 first-child: node number of the first element
              of the child set of node T;     for j = 1, 2, ..., k
              record pointer                  for j = k;
  last-child: node number of the last element
              of the child set of node T;     for j = 1, 2, ..., k
            : 0                               for j = k;
     brother: node number of the node next to
              node T in the order;            for j = 1, 2, ..., k;
      cousin: node number of the first node of
              the filial set next to that of
              node T;                         for j = 1, 2, ..., k;
```

The fields *first-child* and *last-child* are used in the breadth-first linearization. The fields *brother* and *cousin* are used in the depth-first linearization. The algorithms to generate a depth-first directory can be obtained from Kashyap *et al.* (1977), and Gopalakrishana and Veni Madhavan (1980). We give an algorithm to generate a breadth-first directory.

Step 2 of DIR-GENERATION computes base[i], $i = 1, 2, \ldots, (k+1)$, such that any directory node at level i lies between base[i] and base[$i+1$] $- 1$. These values are needed to store the value, first-child and last-child fields in algorithm DIR-GENERATION. This algorithm is given below.

The main cost of algorithm DIR-GENERATION is due to sorting. After the imput records and sorted, two passes are made on the sorted records, once

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

(a)

FIG. 11. (a) Depth-first directory; (b) breadth-first directory.

for algorithm COMPUTE-BASE and once for lines 3 to 11 of algorithm DIR-GENERATION. If the input resides on the main memory, then the sorting phase has $O(kN \log(kN))$ comparisons. Algorithm COMPUTE-BASE takes $O(kN)$ comparisons, or memory accesses. This is because line 6 of COMPUTE-BASE and line 6 of COMPUTE-BASE and line 6 of DIR-GENERATION each involve at most k comparisons for each record. Again, line 8 of COMPUTE-BASE is executed at most k times for each record. Similarly, we can show that line 8 and line 9 of DIR-GENERATION each can be executed at most k times for each record. Hence the complexity of

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	—

(b)

FIG. 11. *(Continued)*

DIR-GENERATION is $O((kN)\log(kN) + 2kN) = O((kN)\log(kN))$. If each record comparison is taken to be of unit time then the complexity is $O(N \log N)$.

If the input records reside on secondary storage, then the cost is due to the page accesses during sorting. The other parts of the algorithm make two sequential passes over the sorted data, and $O(m)$ page accesses are made during these passes, where m is the number of pages in which data is stored. Hence, using standard external sorting techniques, the directory generation can be done in $O(m \log m)$ page accesses as in Horowitz and Sahni (1976).

```
algorithm DIR-GENERATION;
begin
1.  sort the input data file on all $k$ attributes
2.  COMPUTE-BASE;
3.  read first record;
4.  repeat
5.     read next record;
6.     compare it with previous record to get $L$,
       the highest level at which the attribute
       values are different;
7.  for $j := L$ to $k$ do
8.     assign value field;
9.  for $j := (L + 1)$ to $k$ do
10.    assign first-child and last-child fields;
11. until all records are processed;
end;
```

```
algorithm COMPUTE-BASE;
begin
1.  for $i := 1$ to $(k + 1)$ do
2.     base[$i$] := $i$;
3.  read first record;
4.  repeat
5.     read next record;
6.     compare it with previous record to get $L$,
       the highest level at which the attribute
       values are different;
7.  for $j := L$ to $(k + 1)$ do
8.     base[$j$] = base[$j$] + 1;
9.  until all records are processed;
10. for $j := 2$ to $(k + 1)$ do
11.    base[$j$] = base[$j$] + base[$j - 1$];
end;
```

6.4 Search Algorithms

In this section, we dicuss MAT-based search algorithms for complete match, partial match and range queries. The concept of "clustering" of data is exploited to improve the average case performance of various search algorithms. This is achieved by keeping together the nodes that are likely to be accessed in answering a given query. The analysis of the performance is carried in the following two cases:

(i) directory resides on the main memory,
(ii) directory resides on the secondary storage.

The breadth-first linearization assures that the members of a filial set are consecutive in main memory and a binary search can be carried out on them. When the directory resides on secondary memory, this clustering helps in reducing the number of pages accessed. We assume that each filial set occupies no more than a constant number of pages on secondary memory. We also assume that a filial set can be completely loaded into the main memory when queries are being answered. We show in this section that the MAT fares almost as well as other structures, such as the *k*B-tree, MDBT, etc., when the directory resides on main memory. Our main result is showing that the MAT is a better choice when the directory resides on secondary memory.

We define a query Q, as follows:

$$Q = \bigcap_{i=1}^{k} q_i$$

where q_i is the qualifier for atrribute A_i. For a complete match query, q_i specifies a value for A_i, $i = 1, 2, \dots, k$. For a range query, q_i specifies a range $[l_i, h_i]$ for attribute A_i, $i = 1, 2, \dots, k$, where l_i and h_i specify the lower and upper limits of the range respectively. For a partial match, each level i. $i = 1, 2, \dots, k$, could be specified or unspecified. For a specified level i, q_i specifies a value for A_i. For unspecified levels, q_i does not specify any restriction on A_i. A full match can be visualized as a range query that specifies the range of a single value for each attribute. Similarly, a partial match query can be visualized as a range query that specifies a range of size one for specified levels, and range of the domain of the attribute for unspecified levels. Normally, a complete match and partial match are easier to answer than a range query. We also discuss a restricted range query, which specifies no more than a values for each attribute. Answering a query can be visualized using geometric notions: the set of records of k attributes describes a k-dimensional space. A range query describes a rectilinearly oriented hyperrectangle in k dimensions. Answering a range query corresponds to the retrieval of the points contained in the hyperrectangle specified by the range query. In the MAT-based organization, while answering a complete match, partial match or range query, the processing of each attribute reduces the dimensionality of the space to be searched by one.

The basic notion of a MAT-based search is that of "hierarchical descent", where the query is processed level by level starting from the root node. At each level $j, j = 1, 2, \ldots, k$, nodes that satisfy the 'partial' query $\bigcap_{i=1}^{j} q_i$ are aggregated. These nodes are called *qualified nodes*. The qualified nodes at level k contain pointers to the physical records that satisfy the query. We restrict our discussion to the breadth-first linearization. More details on this can be found from Rao *et al.* (1984). The discussion on depth-first linearization can be found in Gopalakrishana and Veni Madhavan(1980).

In the algorithm MAT-SEARCH, line 10 corresponds to the initial call. When the directory resides on secondary storage, the execution of lines 2 and 3 is very critical in deciding the number of page accesses. The set qnodes of MAT-SEARCH is sorted and accessed in order. This results in accessing the consecutive filial sets from left to right in the MAT structure. This type of access is efficient when the sets are of smaller sizes such that a number of them can be accommodated in a single page. This scenario occurs in answering a range query. In fact, using breadth-first ordering and a sorted qnodes set, sequential accesses of the directory can also be supported efficiently.

```
algorithm MAT-SEARCH(level, qnodes);
begin
1.  tempset := $\phi$;
2.  for each $n \in$ qnodes do
3.      add to tempset all children of $n$ that
        satisfy $q_{\text{level}}$;
4.  if level $< k$
5.  then MAT-SEARCH(level + 1, tempset)
6.  else
    begin
7.      for each $n \in$ tempset do
8.  retrieve record pointer;
9.  end;
end;
10. MAT-SEARCH(1, {root});
```

a) Complete Match Query. For a complete match query, at any level j, there is only one node that satisfies the partial query $\bigcap_{i=1}^{j} q_i$ since $\bigcap_{i=1}^{j} q_i$ specifies a unique combination of attribute values in the 'partial' MAT curtailed to the first j levels. Hence line 3 of algorithm MAT-SEARCH is executed only once for each level. For each execution of line 3, a binary search is carried out on the filial set for the specified attribute value. Hence the worst-case complexity of the complete match query is given by

$$O(\log(v_1) + \log(v_2) + \cdots + \log(v_k)) = O(k \log N).$$

If there is "uniformity" in the profile of MAT, such that the average size of the filial set of level j is s_j, then the average complete-match query cost becomes

$$O(\log(s_1) + \log(s_2) + \cdots + \log(s_k)) = O(\log N).$$

This estimate is appropriate for random data satisfying certain distributions. The complexity of $O(\log N)$ for a complete match for multidimensional data can be easily achieved by treating it as an equivalent single-dimensional problem. The key corresponding to any record in the equivalent problem is obtained by concatenating the corresponding attribute values. A balanced binary tree can be easily constructed to ensure $O(\log N)$ complexity for searching any key corresponding to the complete match query. However, such a strategy is not suited for handling partial match and range queries.

If the directory is maintained in secondary storage, at each level we retrieve no more than c pages corresponding to the filial set of the qualified node at the previous level. Hence the number of page accesses for a complete match is $O(ck) = O(k)$. In breadth-first organization ck pages are always retrieved for any complete match query. For depth-first linearization, in the best case, one page access will be sufficient to answer a complete match query. In this case all the qualified nodes corresponding to a complete match query lie on a single page. But in the worst case $O(s_1 + s_2 + \cdots + s_k)$ pages may be retrieved. In this case, the qualified nodes are the last ones in their filial sets. Using the average characterizations of the MAT, the worst-case number of page accesses is $O(kN^{1/k})$ for a complete match query. It is interesting to note that this many page accesses will be needed when the MAT is implemented using pointers and no clustering is assumed on the nodes.

b) Partial Match Query. The maximum number of records that satisfy a partial match query that specifies t levels is given by

$$t_1 t_2 \ldots t_k$$

where

$$t_i = \begin{cases} 1 & \text{if } A_i \text{ is specified} \\ v_i & \text{if } A_i \text{ is not specified} \end{cases}$$

Using the average characterizations of Section 6.1, we can conclude that the number of records that satisfy a given partial match query is $O(N^{1-1/k})$ for uniformly distributed data. In line 3 of algorithm MAT-SEARCH, for specified levels, we carry out binary search on the filial sets. For unspecified levels, we qualify the entire filial set. In the worst-case situation for our algorithm the first $(k - t)$ attributes are unspecified, and each qualified node at level $(k - t)$ gives rise to one qualified node at each subsequent level. Hence for each level j, $(k - t + 1) < j < k$, we have $O(N^{1-t/k})$ sets to be searched, and at each level there can be $O(N^{1-t/k})$ qualified nodes. Hence the cost of answering a partial match query is given by

$$O(s_1 s_2 \cdots s_{k-1}(\log(s_{k-t+1}) + \cdots + \log(s_k))) = O(tN^{1-t/k}\log(N^{t/k}))$$

We note that the above expression approaches $O(\log N)$ as more and more attributes are specified. But, as we leave some of the attributes unspecified, the complexity increases. This is because more and more nodes could get qualified for a partial match as fewer and fewer attributes are specified.

If the directory is maintained on secondary storage the number of page accesses corresponding to the first $(k - t)$ unspecified levels is $O(N^{1-(t-1)/k}$— one page access for each accessed filial set. The number of pages accessed in the subsequent t specified levels is $O(tN^{1-(t-1)/k})$. Hence the total number of page accesses is $O(tN^{1-(t-1)/k})$. As more and more attributes are specified, this cost approaches $O(kN^{1/k})$. If no linearization is assumed, each qualified node might give rise to a page access, and thus the number of page accesses is $O(tN^{1-t/k}\log(N^{t/k}))$. If depth-first ordering is used for linearization, the number of page accesses could be shown to be $O(tN^{1-t/k})$. The performance of breadth-first linearization is marginally better than that of depth-first linearization. However, both linearizations are more efficient than the random implementation. It is to be noted that there exists a certain amount of clustering of nodes on the secondary pages, and the assumption of complete randomness on the nodes allocation is not valid. However, it provides a theoretical upper bound on the number of page accesses.

c) Range Query. The performance analysis of a range query based on the worst case is mainly of theoretical significance rather than of great practical value. Specifically, in the worst case, a range query may retrieve all the N records, which is not very realistic, especially in large database environments. Normally, the number of records retrieved is small, and the queries that retrieve a large number of records are infrequent. Hence an approximate estimate for the number of retrieved records that is close to the average value is more appropriate to describe the performance of the system. However, the average case analysis is difficult because the exact average case patterns of a query are not known and also the average case varies from problem to problem. In this section we first consider generalized range queries and then discuss restricted range queries.

Normally, the time complexity of a range query is expressed in the form

$$O(f(N) + K_1),$$

where k_1 is the number of records retrieved and $f(N)$ gives the cost involved in searching for the limits that delimit the qualified record pointers. The main efforts in designing efficient algorithms for range search are directed towards decreasing $f(N)$ without considerable increase in storage and preprocessing costs. For a range query, step 3 of the algorithm MAT-SEARCH is modified to search for the upper and lower limit of the specified range for that level. The nodes that lie within the limits are retrieved and added to tempset. This feature is well supported by the breadth-first linearization.

In the worst cast of the range query, all the nodes of the searched fiial sets are qualified. Hence for a filial set of size s, the cost of searching for limits is $O(\log(s))$ and the retrieval of the qualifid node costs $O(s)$. Using the average characterization of the MAT, the search cost at level j can be computed as $((\prod_{i=1}^{j-1} s_i)\log(s_j))$. Since, all the nodes of a level are qualified in the worst case, the cost of processing a range query for level j is $O((\prod_{i=1}^{j-1} s_i)\log(s_j) + \prod_{i=1}^{j} s_i)$. Hence, the time complexity of a range query is given by $\sum_{j=1}^{k-1}(\prod_{i=1}^{j} s_i) + (\prod_{i=1}^{k-1}(s_i)\log(s_k) + K_1$ where the first term corresponds to the cost of processing the first $(k-1)$ levels. The second term corresponds to the cost of searching for the range limits for the final level. The third term, namely K_1, corresponds to the number of nodes qualified in the final level. The cost of a range query can be simplified and given as follows:

$$O(N^{1-1/k}\log(N^{1/k}) + K_1).$$

If the directory resides on secondary storage, arguments similar to those above can be used to show that the number of page accesses is $O(N^{1-1/k})$. If no linearization is used, and random location of nodes among secondary storage pages is assumed, then the number of page accesses is $O(N^{1-1/k} + K_1)$. However, as remarked earlier, such an assumption is not very practical. It is again interesting to note that depth-first ordering also has a range search complexity of $O(N^{1-1/k} + K_1)$ page accesses.

So far we have considered only the worst-case estimates for a range query. The assumption that all members of filial sets satisfy the query is not realistic. If we impose a restriction that for a range query no more than a nodes get qualified in each filial set searched, then the complexity of a range query can be given as:

$$\sum_{j=1}^{k-1}\prod_{i=1}^{j} a^i + \left(\prod_{j=1}^{k-1} a^j\right)\log(s_k) + K_1$$

which can be approximated (on the average) by

$$O(a^{k-1}\log(N^{1/k}) + K_1).$$

Let us define p, called the *qualified fraction*, to be

$$p = \frac{a + a^2 + \cdots + a^k}{N}$$

$$= \frac{a(a^k - 1)}{(a - 1)N}$$

$$= \frac{a^k}{N}, \quad \text{for reasonably large } a.$$

The qualified fraction gives an average estimate of the maximum value of the fraction of the nodes that get qualified from a filial set. Now the cost of the average range query is given by

$$O((p(N)^{(k-1)/k} \log(N^{1/k}) + K_1) \approx O(pN^{1-1/k} \log(N^{1/k}) + K_1)$$

Hence, the value of p determines the cost of the range query. When the directory resides on secondary storage, the number of page accesses is given by $O(pN^{1-1/k})$.

The intuitive feel for this average profile of a range query given by Rao and Iyengar (1986) is that, on the average, a range query defines a "sub-MAT" that has a filial-set size of ps_i at level i, $i = 1, 2, \ldots, k$. This visualization gives more insight into the process of answering a range query on MAT. Table I summarizes the order of the cost of processing a range query for various values of p.

The performance of the MAT becomes better as the number of records retrieved by a range query is reduced. This is generally the case for applications

TABLE I

AVERAGE COST ANALYSIS FOR THE RANGE QUERY OF TYPE $Q = \bigcap_{i=1}^{k} q_i$

(K_1 denotes the number of qualified records)

Qualified fraction p	Query cost main memory	Query cost secondary memory
$\left(\frac{N}{N}\right)$	$O(N^{1-1/k} \log(N^{1/k}) + K_1)$	$O(N^{1-1/k})$
$\left(\frac{\log N}{N}\right)$	$O(\log N + K_1)$	$O(\log N)$
$\left(\frac{\log \log N}{N}\right)$	$O(\log(N^{1/k} + \log N) + K_1)$	$O(\log(N^{1/k} + \log N))$
$\left(\frac{c}{N}\right)$, c = constant	$O(\log(N^{1/k}) + K_1)$	$O(\log(N^{1/k}))$

involving large amounts of data, such as large databases, image processing systems, etc.

We conclude this section by noting that when secondary storage is used for the directory the appropriateness of MAT is enhanced as a result of keeping the nodes of the filial sets together in secondary storage pages.

6.5 Comparison of Performance

In this section, we compare the performance of the MAT data structure with other multidimensional data structures. The basis for the comparison is the query, preprocessing, and storage costs when the directory resides on the main memory. The comparison of the MAT with the data structures such as the MDBT, kB-tree, BST-complex, etc. was carried out earlier. They all have nearly the same performance measures, as observed in the last sections. Here, we consider the data structures that do not resemble the MAT very closely in the structural aspects. These include the sequential scan, projection, quadtree, k-d tree, range tree, non-overlapping k-ranges, and overlapping k-ranges. An excellent survey of these data structures for the range search can be found in Bentley (1980).

A complete match query is of complexity $O(\log N)$ in the MAT data structure, quadtree, k-d tree, etc. Similarly, a partial match query can be answered in $O(N^{1-1/k}\log(N^{1/k}))$ in the MAT. The sequential scan has complexity of $O(N)$ for both the complete and partial match query. The projection data structure has the complexities of $O(t \log tN^{1-1/k})$, and $O(kN^{1-1/k})$ for partial match and complete query respectively. The k-d tree and quadtree have the time complexities of $O(\log N)$ and $O(N^{1-1/k})$ for complete and partial match queries. The other data structures such as range trees, d-fold tree, etc. are specifically designed for range queries and, in general, are not well suited for complete and partial match queries.

The comparison of the performance of various data structures is depicted in Table II. Basically, there are four types of data structures from the range search point of view. The first type has complexity of $O(N^{1-1/k} + K_1)$, and the second type has complexity of $O(\log^k + K_1)$. The third and fourth have the range search complexities of $O(\log N + K_1)$, and $O(N^{\epsilon})$ respectively. The MAT data structure belongs to the first type. It is to be noted that a reduction in one cost results in an increase in the other costs for any data structure. In the second type of data structure, the reduction in the query cost to $O(\log^k N + K_1)$ is a result of the increases in the preprocessing and storage costs to $O(N \log^{k-1} N)$ and $O(N \log^{k-1} N)$ respectively. This effect is more prominent in the overlapping k-ranges. In this data structure the further reduction in the query to $O(\log N + K_1)$ results in the high preprocessing and query costs given by $O(N^{1+\epsilon})$, where ϵ is a positive number. The inverse effect is observed

TABLE II

COMPARISON OF PERFORMANCE OF SOME MULTIDIMENSIONAL DATA STRUCTURES

Data structure	Preprocessing cost	Storage cost	Range query cost
Sequential scan	$O(N)$	$O(N)$	$O(N)$
Projection	$O(N \log N)$	$O(N)$	$O(N^{1-1/k} + K_1)$
Quadtree *k*-d tree	$O(N \log N)$	$O(N)$	$O(N^{1-1/k} + K_1)$
MAT	$O(N \log N)$	$O(N)$	$O(N^{1-1/k} \log(N^{1/k}) + K_1)$
Range tree d-fold tree Super *B*-tree Quintanary tree	$O(N \log^{k-1} N)$	$O(N \log^{k-1} N)$	$O(\log^k N + K_1)$
k-fold tree	$O(N \log^{k-1} N)$	$O(N \log^{k-1} N)$	$O(\log^{k-1} N + K_1)$
Overlapping *k*-ranges	$O(N^{1+\epsilon})$	$O(N^{1+\epsilon})$	$O(\log N + K_1)$
Non-overlapping *k*-ranges	$O(N \log N)$	$O(N)$	$O(N^\epsilon + K_1)$

in non-overlapping k-ranges which have the query cost of $O(N^\epsilon + K_1)$ and have preprocessing and storage costs $O(N \log N)$, and $O(N)$ respectively. This seems to suggest an intimate relationship between the query, preprocessing, and storage costs.

When the MAT is organized on secondary storage, its performance should be compared with that of structures such as the grid file, extended hashing, etc. However, the analysis of most of these structures is not carried out in the manner that is done for the MAT. This makes the comparison very difficult.

We close this section by noting that the MAT belongs to the family of data structures that contains k-d trees, quadtrees, etc. Despite their similarity in the worst-case complexity, each fares better than the other in the average case of certain aplications. The properties of the problem have to be studied extensively to choose one data structure over the other. The aspects that make the MAT data structure efficient in certain applications are discussed in the next section.

7. Paradigms for the MAT Data Structure

Although the MAT belongs to the class of quadtrees and k-d-trees, its worst-case performance can be made better than others in certain applications. The properties of the data and queries can be used to improve the performance of the MAT in many real-life applications. The data properties such as the sizes of various filial sets, the ordering of attributes, etc, decide the profile of the MAT. This in turn determines the complexities of various queries. The query properties such as the query type, the probability of occurrence of different attributes in a query, the number of records that satisfy the query, etc., decide the performance of different queries on the MAT. This section provides a brief discussion of these two aspects in the form of paradigms.

Paradigm I. For the same number of records, a MAT with a profile that has fewer large filial sets fares better than a MAT with more smaller filial sets. Figures 12b and 13b give the MAT for the data of Fig. 12a and 13a respectively. In general, data with "large clusters" has a more efficient MAT structure than a MAT with "dispersed" data. For any given data, the profile of

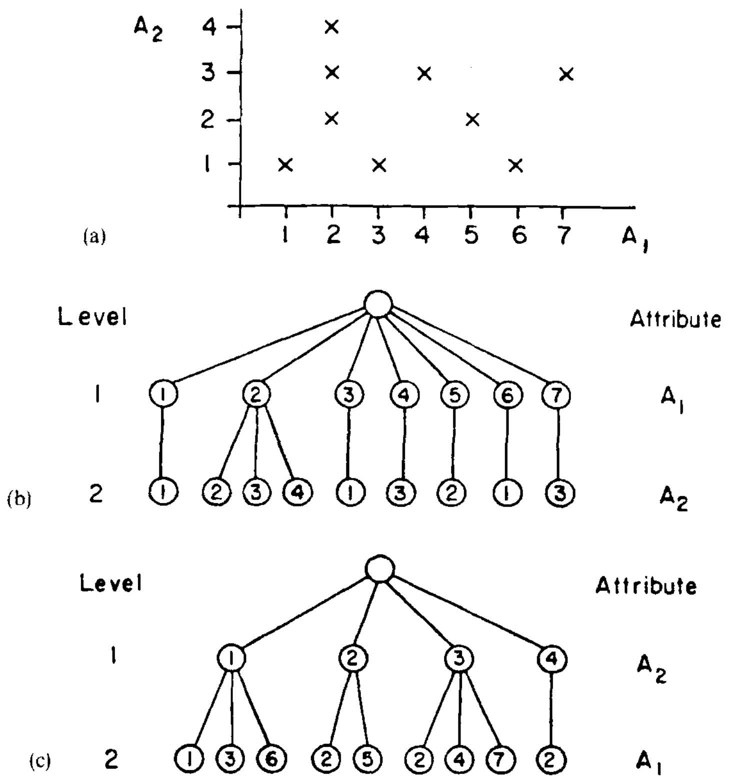

FIG. 12. (a) Sample data; (b) MAT with attribute ordering A_1, A_2; (c) MAT with attribute ordering A_2, A_1.

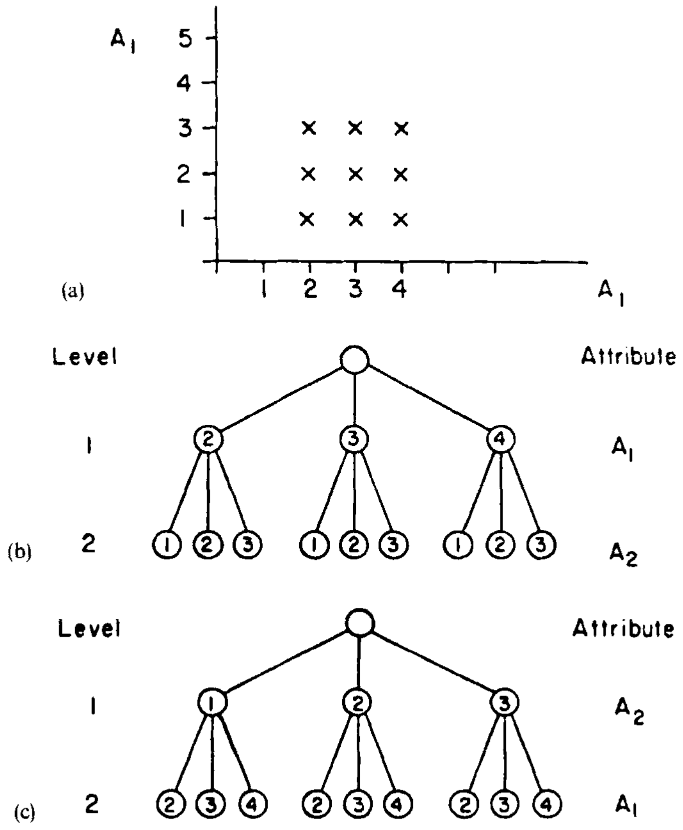

FIG. 13. (a) Sample data; (b) MAT with attribute ordering A_1, A_2; (c) MAT with attribute ordering A_2, A_1.

the MAT depends on the attributes ordering also. Figure 12a illustrates that interchanging the attributes A_1 and A_2 results in a more compact MAT. The ordering of attributes such that the average filial set size decreases towards the bottom, is shown to be optimal for the average case response of a range query in Rao and Iyengar (1986). Another criterion for ordering the attributes is to minimize the average size of filial sets in all levels. These two requirements may not be met simultaneously. A tradeoff is needed if both the requirements are conflicting.

Paradigm 2. If the more frequently appearing attributes are placed at higher levels, more screening of the nodes will be done at higher levels. This results in fewer nodes to be searched at lower levels. This approach is appropriate for partial match queries. For a range query, the attributes that are likely to result in fewer qualified nodes may be placed higher in the MAT structure. The growth in the number of nodes to be searched will be restricted as a consequence of this ordering.

The above paradigms suggest MAT as a promising indexed structure for information retrieval applications.

8. Conclusions

The design and analysis of multidimensional data structures and algorithms has gained increasing importance in recent years, especially in the areas of computational geometry, robotics, information retrieval and physical database organization. As a result, a large number of multidimensional data structures have been proposed and studied in various application areas. A brief survey of these existing data structures along with some new paradigms for balancing multidimensional search trees has been presented in this paper. In addition we have explored the potential of Multiple Attribute Trees as a promising multidimensional data structure for application in range query problems. This survey is by no means exhaustive, and omission of any data structure is unintentional.

We have attempted here to illustrate and emphasize the following aspects:

1. The need to develop and analyze simple, efficient balanced self-organizing search tree structures that are useful for real-time applications.
2. A relative analysis of the existing techniques for extension of balanced search trees to multidimensional domains.
3. A model of balanced multidimensional and weighted trees by means of well-defined paradigms.
4. An introductory description of Multiple Attribute Trees and the analysis of their performance for complete match query, partial match query and range query.

It is hoped that the survey will result in a better understanding of the working as well as the development of new performance characteristics of multi-dimensional data structures. In view of the ongoing Fifth Generation Computer Project and the increasing emphasis on the use of high speed computers, a detailed study of the techniques for the design and analysis of balanced self-organizing multidimensional search tree structures becomes quite important.

The notions of time and space complexity are indeed helpful in making an appropriate choice of a data structure in high speed computation. But it is difficult to analyze the data structure of the computation because of the close interleaving and indeterminacy of mapping costs.

Vaishnavi's [1984, 1987] work described in Section 3 presents a framework for implementations of balanced, self-organizing multidimensional and weighted trees. Such structures need to have efficient worst-case time performance per operation. Multiple attribute trees is a multidimensional data structure with special relevance physical database organization. A major theme of our paper is the exploration of new paradigms for the above structures.

We can view the paradigms discussed here at two different levels: an abstract level and an implementation level.

At the abstract level, the paradigms discussed can be split up into two categories:

1. Strategies to support "structure violations" in the "generalized balanced multidimensional weighted trees".
2. Strategies to exploit structural properties of multiple attribute trees.

At the implementation level, the paradigms for balanced multidimensional weighted trees can be split up into four categories:

1. Structure violations must be "supported" by each of the (next lower dimensional) "sibling" subtrees of the node. We refer to this paradigm as the *local-support strategy* paradigm.
2. In the next strategy, called *global-support strategy*, the structure violations must be supported by each of the "adjoining" subtrees of the nodes causing the structure violation.
3. In the *neighbor-support strategy*, the structure violations must be supported by at least one of the "adjoining" subtrees of the nodes causing the structure violation.
4. By exploiting computational properties of the data structure.

These paradigms have been substantiated with suitable examples. We would like to emphasize here that there is no universal method for designing efficient multidimensional data structures. The trick lies in identifying the right formulation of the computational problem followed by an appropriate decomposition for efficient implementation. The techniques presented in this paper could be used for achieving the above decomposition.

Acknowledgements

The authors would like to acknowledge the assistance received from Prof. D. Kraft and Prof. Moitra in terms of editorial comments.

References

Adel'son-Velskij, G. M., and Landis, Y. M. (1962). An algorithm for the Organization of Information. *Soviet Mathematic Doklady* **3,** 1259–1263.

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1974). "The Design and Analysis of Computer Algorithms." Addison-Wesley, Reading, MA.

Alt, H., Mehlhorn, K., and Munro, J. I. (1981). Partial match retrieval in implicit data structures. *Proc. 10th Symp. Math. Foundations of Comp. Sci.*, pp. 156–161.

Avid, Z., and Shamir, E. (1981). A direct solution to range search and related problems for product regions. *Proc. 22nd Annual Symp. on Foundations of Comp. Sci.*, pp. 123–126.

Barnes, M. C., and Collens, D. S. (1973). "Storing Hierarchical Database Structures in Transposed Form." Datafair.

Batory, D. S. (1979). On searching transposed files. *ACM Trans. Database Syst.* **4,** 531–544.

Bayer, R. and McCreight, E. (1972). Organization and maintenance of large ordered indexes. *Acta Informatica* **1,** 173–189.

Bent, S. W. (1982). Dynamic Weighted Data Structures. Ph.D. dissertation, Stanford University, Report No. STAN-CS-82-916.

Bent, S. W., Sleator, D. D., and Tarjan, R. E. (1980). Biased 2-3 Trees. *Proc. 21st Annu. IEEE Symp. on Foundations of Comp. Sci.*, 248–254.

Bent, S. W., Sleator, D. D., and Tarjan, R. E. (1985). Biased Search Trees. *SIAM J. Computing* **14,** 545–568.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM* **18** (9), 509–517.

Bentley, J. L. (1977). Solutions to Klee's rectangle problems. Carnegie-Mellon University, Department of Computer Science, typescript.

Bentley, J. L. (1979a). Decomposable searching problems. *Inf. Process. Lett.* **8** (5), 133–136.

Bentley, J. L. (1979b). Multidimensional binary search trees in database applications. *IEEE Trans. Softw. Eng.* **SE-5** (4), 333–340.

Bentley, J. L. (1980). Multidimensional divide-and-conquer. *Commun. ACM* **23** (4), 214–229.

Bentley, J. L., and Burkhard, W. A. (1976). Heuristics for partial match retrieval data base design. *Inf. Process Lett.* **4** (5), 132–135.

Bentley, J. L., and Friedman, J. H. (1979). Data structures for range searching. *ACM Comput. Surveys* **11** (4), 397–409.

Bently, J. L., and McGeoch, C. C. (1985). Amortized analysis of self-organizing sequential search heuristics. *Commun. ACM* **28,** 404–411.

Bentley, J. L., and Ottmann, T. H. (1979). Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comp.* **C-28,** 643–647.

Bentley, J. L., and Saxe, J. B. (1979). Algorithms on vector sets. *SIGACT News* **11,** 36–39.

Bentley, J. L., and Saxe, J. B. (1980). Decomposable searching problems I: static-to-dynamic transformations. *Journal of Algorithms* **1,** 571–577.

Bentley, J. L., and Shamos, M. I. (1977). A problem in multivariate statistics: algorithm, data structure, and application. *Proc. 15th Allerton Conf. on Commun., Control and Comput.*, 193–201.

Bentley, J. L., and Stanat, D. F. (1975). Analysis of range searches in quad trees. *Inf. Process. Lett.* **4** (5), 170–173.

Bolour, A. (1981). Optimal retrieval algorithms for small region queries. *SIAM J. Comput.* **10,** 721–741.

Burkhard, W. A. (1983). Interpolation based index maintenance. *Proc. ACM Symp. Principles of Database Systems*, 76–89.

Cardenas, A. F., and Sagamang, J. P. (1974). Double-chained tree data base organization–analysis and strategies. IBM research report RJ 1374.

Casey, R. G. (1973). Design of tree structures for efficient querying. *Commun. ACM* **16** (9), 546–556.

Chang, J. M., and Fu, K. S. (1981). Extended *k*-d tree database organization—a dynamic multiattribute clustering method. *IEEE Trans. Softw. Eng.* **SE-7** (3), 284–290.

Dobkin, D. P., and Lipton, R. (1976). Multidimensional searching problems. *SIAM J. Comput.* **5**, 181–186.

Edelsbrunner, H. (1980a). Dynamic rectangle intersection searching. Technische Universität Graz, Institut für Informationsverarbeitung, Report F47.

Edelsbrunner, H. (1980b). Dynamic data structure for orthogonal intersection queries. Technische Universität Graz Report #59.

Edelsbrunner, H. (1981). A note on dynamic range searching. *Bulletin of EATCS* **15**, 34–40.

Edelsbrunner, H. (1983). A new approach to rectangle intersections. *Int. J. Comput. Math.* **13** (3 & 4), 209–229.

Edelsbrunner, H. (1984). Key-problems and key-methods in computational geometry. *STACS84 Symp. Thero. Aspects of Comp. Sci.*, Paris, April 1984, Lect. Notes Comp. Sci. Springer-Verlag, 1–13.

Edelsbrunner, H., and Maurer, H. A. (1981). On the intersection of orthogonal objects. *Inf. Process. Lett.* **13**, 177–181.

Edelsbrunner, H., and van Leeuwen, J. (1983). Multidimensional data structures and algorithms: Bibliography. IIG, Technische Universität Graz, Austria, Rep. 104.

Fagin, R., Nievergelt, J., Pippenger, N., and Strong, H. R. (1979). Extendible hashing—a fast method for dynamic files. *ACM Trans. Database Syst.* **4** (3), 315–344.

Feigenbaum, J. and Tarjan, R. E. (1983). Two new kinds of biased search trees. *Bell Syst. Tech. Journal* **62**, 3139–3158.

Finkel, R. A., and Bentley, J. L. (1974). Quad trees—a data structure for retrieval on composite keys. *Acta Informatica* **4** (1), 1–9.

Fredkin, E. (1960). Trie Memory. *Commun. ACM* **3**, 490–499.

Fredman, M. L. (1981a). The spanning lower bound on the complexity of orthogonal range queries. *J. Algorithms* **1**, 77–87.

Fredman, M. L. (1981b). A lower bound on the complexity of orthogonal range queries. *J. ACM* **28**, 696–705.

Fredman, M. L. (1981). Lower bounds on the complexity of orthogonal range queries. *SIAM J. Comput.* **10**, 1–10.

Gopalakrishna, V., and Veni Madhavan, C. E. (1980). Performance evaluation attribute-based tree organization *ACM Trans. Database. Syst.* **6** (1), 69–87.

Gotlib, C. C., and Gotlib, L. R. (1978). Data types and structures. Prentice-Hall, Englewood Cliffs, N. J.

Gueting R. H. and Kriegel, H. P. (1980). Multidimensional B-tree: an efficient dynamic file structure for exact match queries. *Proc. 10th GI Annu. Conf.*, Informatik Fachberichte, Springer Verlag, 375–388.

Gueting R. H. and Kriegel H. P. (1981). Dynamic *k*-dimensional multiway search under time-varying access frequencies. *Proc. 5th GI Conf. on Theoretical Comp. Sci.*, Lecture Notes in Computer Science 104. Springer-Verlag, New York, 135–145.

Hart, J. H. (1981). Optimal two dimensional range using binary range lists. Tech. Report 76-81, Dept. Computer Science, Univ. Kentucky.

Hirschberg, D. S. (1980). On the complexity of searching a set of vectors. *SIAM J. Comput.* **9**, 126–129.

Horowitz, E., and Sahni, S. (1976). "Fundamentals of data structures." Computer Science Press.

Huddleston, S., and Mehlhorn, K. (1982). A new data structure for representing sorted lists. *Acta Informatica* **17**, 157–184.

Iyengar, S. S., and Raman, V. (1983). Properties and applictions of forest of quadtrees for pictorial information. *BIT* **23,** 472–486.

Jones, L., and Iyengar, S. S. (1984). Space and time efficient virtual quadtrees. *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-6** (2), 244–247.

Kashyap, R. L., and Subas, S. K. C., and Yao, S. B. (1977). Analysis of multiattribute tree database organization. *IEEE Trans. Softw. Eng.* **SE-2** (6), 451–467.

Kirkpatrick, D. G. (1983). Optimal search in planar subdivisions. *SIAM J. Comput.* **12,** 28–35.

Knuth, D. E. (1974). "The art of computer programming, Vol. 3: Sorting and searching." Addison-Wesley, Reading, Mass.

Kosaraju, R. (1981). Localized search in sorted lists. *Proc. 14th Symp. on Theory of Computing,* 62–69.

Kriegel, H. P. (1981). Variants of multidimensional B-trees as dynamic index structure for associative retrieval database systems. *Proc. 7th Conf. Graph Theoretic Concepts in Comput. Sci.,* Linz, Austria.

Kriegel, H. P. (1984). Performance comparison of index structures for multi-key retrieval. *Proc. Annu. Meeting of SIGMOD,* 186–195.

Kriegel, H. P., and Vaishnavi, V. K. (1981a). Multidimensional B^+-trees. unpublished manuscript.

Kriegel, H. P., and Vaishnavi, V. K. (1981b). Weighted multidimensional B-trees used as nearly optimal dynamic dictionaries. *Proc. 10th Int. Symp. on Math. Found. of Comput. Sci.* **118,** 410–417.

Kriegel, H. P., and Vaishnavi, V. K. (1981c). A nearly optimal dynamic tree structure for partial-match queries with timevarying frequencies. *Proc. Conf. on Information Sciences and Systems,* 193–197.

Lee, D. T., and Preperata, F. P. (1984). Computational geometry—a survey. *IEEE Trans. Computers* **C-33** (12), 1072–1101.

Lee, D. T., and Wong, C. K. (1977). Worst case analysis of region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica* **9,** 23–29.

Lee, D. T., and Wong, C. K. (1980). Quintary trees: a file structure for multidimensional database systems. *ACM Trans. Database Syst.* **5** (3), 339–353.

Lee, D. T., and Wong, C. K. (1981). Finding intersections of rectangles by range search. *J. Algorithms* **2,** 337–347.

Lien, E. Y., Taylor, E. C., Driscoll, R. J., and Reynolds, L. H. (1975). Binary search tree complex-towards the implementation, *Proc, Very Large Data Base Conf.* Framingham, Mass, 540–542.

Liou, J. H., and Yao, S. B. (1977). Multi-dimensional clustering for data base organization. *Inf. Syst,* **2,** 187–198.

Litwin, W. (1980). Linear hashing: a new tool for file and table addressing. *Proc. 6th Int. Conf. on Very Large Data Bases,* 212–223.

Lueker, G. S. (1978). A data structures for orthogonal range queries. *Proc. 19th Annu. Symp. Foundations of Comput. Sci.,* 28–34.

Lueker, G. S. (1979). A transformation for adding range restriction capability to dynamic data structures for decomposable searching problems. Tech. Report #129, Dept. Information and Computer Science, Univ. California, Irvine.

Lueker, G. S. (1982). A data structure for dynamic range queries. *Inf. Proc. Lett.* **15,** 209–213.

Lum, V. Y. (1970). Multi-attribute retrieval with combined indexes. *Commun. ACM* **13,** 660–665.

McCreight, E. M. (1980). Efficient algorithms for enumerating intersecting intervals and rectangles. Xerox Parc Report, CSL-80-09.

Mehlhorn, K. (1979). Dynamic binary search. *SIAM J. Computing* **8,** 175–198.

Mehlhorn, K. (1981). Lower bounds on the efficiency of transforming static data structures to dynamic data structures. *Math. Sys. Theo.* **15,** 1–11.

Mehlhorn, K. (1984a). "Data Structures and Algorithms 1: Sorting and Searching." Springer-Verlag.

Mehlhorn, K. (1984b). "Data structures and algorithms 3: Multidimensional tree structures and computational geometry." Springer-Verlag.

Mehlhorn, K., and Overmars, M. H. (1981). Optimal dynamization of decomposable problems. *Inf. Process. Lett.* **12,** 93–98.

Nievergelt, J., and Reingold, E. M. (1973). Binary search trees of bounded balance. *SIAM J. Comput.* **2,** 33–43.

Nievergelt, J., Hinterberger, H., and Sevcik, K. C. (1984). The grid-file: an adaptable, symmetric multi-key file structures. *ACM Trans. on Database Sys.* **9,** 38–71.

Orenstein. J. A. (1982). Multidimensional TRIEs used for associative searching. *Inform. Proc. Lett.* **4** (4), 150–158.

Ouskel, M., and Scheuermann, P. (1981). Multidimensional B-trees: analysis of dynamic behaviour. *BIT* **21,** 401–418.

Overmars, M. H. (1981). Dynamization of order decomposable set problems. *J. Algorithms* **2,** 245–260.

Overmars, M. H. (1984). "The design of dynamic data structures." *Lect. Notes in Comput. Sci.* **156,** Springer-Verlag, Berlin.

Overmars, M. H., and van Leeuwen, J. (1982). Dynamic multidimensional data structures based on quad- and *k*-d trees. *Acta Informatica* **17,** 267–285.

Pealtz, J. L., Berman, W. J., and Cagley, E. M. (1980). Partial match retrieval using indexed descriptor files. *Commun. ACM* **23,** (9), 522–528.

Preparata, F. P., and Shamos, M. I. (1985). "Computational Geometry." Springer-Verlag, New York.

Rao, N. S. V. (1984). Multiple attribute tree based search and dynamization algorithms. M. E. Thesis, School of Automation, Indian Institute of Science, Bangalore, India.

Rao, N. S. V., and Iyengar, S. S. (1986). Optimal attribute ranking problem in MAT data structure. *Int. J. of Comput. Math.* **21,** 1–12.

Rao, N. S. V., Vaishnavi, V. K. and Iyengar, S. S. (1987). On the dynamization of data structures. Tech. Report #85-031, Dept. of Computer Science, Louisiana State Univ., to appear in *BIT*.

Rao, N. S. V., Veni Madhavan, C. E., and Iyengar, S. S. (1984). Range search and dynamization algorithms in multiple attribute tree. Tech. Report, School of Automation, Indian Institute of Science, Bangalore, India.

Rao, N. S. V., Veni Madhavan, C. E., and Iyengar, S. S. (1985). A comparative study of multiple attribute tree and inverted file structures for large bibliographic files. *Inf. Process. and Mgmt.* **18** (5) 200–215.

Rivest, R. L. (1976). Partial-match retrieval algorithms. *SIAM J. Comput.* **5,** 19–50.

Rothnie, J. B., and Lozano, T. (1974). Attribute based file organization in a paged memory environment. *Commun. ACM* **17** (2). 63–69.

Samet, H. (1983). A quadtree medial axis transform. *Commun. ACM* **26** (9), 680–693.

Samet, H. (1984). The quadtree and related hierarchical data structures. *Comput. Surveys* **16** (2), 187–260.

Scheuermann, P. and Ouksel, M. (1982). Multidimensional B-trees for Associative Searching in Database Systems. *Inf. Syst.* **7,** 123–137.

Shamos, M. I. (1978). Computational Geometry. Ph.D. Thesis, Dept. Comput. Sci., Yale Univ.

Shamos, M. I., and Hoey, D. (1975). Closest point problems. *Proc. 16th Annu. IEEE Symp. Foundations of Comput. Sci.*, 151–162.

Sleator, D. D., and Tarjan, R. E. (1983a). Self-adjusting binary search trees. *Proc. 15th Annu. Symp. on Theory of Computing*, 235–245.

Sleator, D. D., and Tarjan, R. E. (1983b). A data structure for dynamic trees. *J. Comput. and Syst. Sci.* **26,** 362–391.

Sleator, D. D., and Tarjan, R. E. (1985). Self-adjusting binary search trees. J. ACM 32(3), 652–686.

Subas, S. K. C. and Kashyap, R. L. (1975). Database organization—an access and storage method. Tech. Report #TREE 75-32, School of Engineering, Purdue Univ.

Sussenguth, E. H., Jr. (1963). Use of tree structures for processing files. *Commun. ACM* **6** (5) 272–279.

Tamminen, M. (1982). The extendible cell method for closest point problems. *BIT* **22,** 27–41.

Tsakalidis, A. K. (1984). An optimal implementation for localized search. Technical Report A 84/06, University of Sarlandes, West Germany.

Vaishnavi, V. K. (1982). Computing point enclosures. *IEEE Trans. Comput.* **C-31** (1), 22–29.

Vaishnavi, V. K. (1983a). On the worst-case efficient implementation of weighted dynamic dictionaries. *Proc. 21st Annu. Allerton Conf. on Communication, Control, and Computing,* 647–655.

Vaishnavi, V. K. (1983b). How to balance a multidimensional search tree. *Proc. 3rd Conf. on Foundations of Software Technology and Theoretical Computer Science,* Bangalore, India.

Vaishnavi, V. K. (1984). Multidimensional height-balanced trees. *IEEE Trans. Comput.* **C-33** (4), 334–343.

Vaishnavi, V. K. (1986). On the height of multidimensional height-balanced trees. *IEEE Trans. Comput.* **C-35,** 773–780.

Vaishnavi, V. K. (1987). Weighted leaf AVL-trees. *SIAM J. Comput.,* 16 (3) 503–537.

Vaishnavi, V. K., and Wood, D. (1980). Data structures for rectangle containment and enclosure problems. *Computer Graphics and Image Process.* **13,** 372–384.

Veni Madhavan, C. E. (1984). Secondary retrieval using tree data structures. *Theor. Comput. Sci.* **33,** 107–116.

Willard, D. E. (1979a). The super B-tree algorithm. Tech. Report TR-03-79, Aiken Computation Lab., Harvard Univ.

Willard, D. E. (1979b). "Predicate oriented database search algorithms." Garland Pub. Company, New York.

Willard, D. E. (1982). Polygon retrieval. *SIAM J. Comput.* **11** (1), 149–165.

Willard, D. E. (1985). New data structure for orthogonal queries. *SIAM J. Comput.* **14** (1) 232–253.

Willard, D. E., and Lueker, G. S. (1985). Adding range restriction capability to dynamic data structures. *J. ACM* 32 (3), 597–617.