We propose an efficient data structure called the HG-tree (or Hilbert Grid tree) for organizing multidimensional data which are typically points in some domain space. The data structure is developed by extending the concept of the B*-tree to the multidimensional data, so that it guarantees the worst-case storage utilization to have more than 66.7% (2/3) of full capacity. It also maintains compactly the area covered by the directory regions of the data structure, so that it reduces the empty space of the directory region; therefore, the search cost is reduced. We present the design of our data structure and associated algorithms and demonstrate that it is possible to run various range queries and nearest neighbor queries efficiently using this structure. A comparison with the buddy-tree, which is one of the most successful data structures for multidimensional space, is carried out and the advantages of the HG-tree over it are discussed.

Recently, many applications require specialized data structures to handle multidimensional data. Such applications include multimedia databases, CAD/CAM, geographic information systems, and so on. B-trees (Comer, 1979) are an efficient data structure for indexing one-dimensional data. However, they are not suitable for indexing multidimensional data (Kumar, 1994). In recent years, various data structures for multidimensional data have been proposed. However, they are not always satisfactory with respect to the storage utilization. Moreover, their performance is degenerated in the skewed data distribution. We present a new dynamic data structure for indexing multidimensional data, which provides not only the theoretical worst-case storage utilization more than 66.7% (2/3), but also the empirical average storage utilization more than 80% of full capacity. We call it the HG-tree (or Hilbert Grid tree). The HG-tree is an n-dimensional extension of the one-dimensional B*-tree (Comer, 1979). A B*-tree is a B-tree in which each node is at least 2/3 full (instead of just 1/2 full). B*-tree insertion employs a local redistribution scheme to delay node splitting until two sibling nodes are full. Then the two nodes are divided into three, each 2/3 full. This scheme guarantees that storage utilization is at least 66.7% (2/3). It should be pointed out that increasing storage utilization speeds up the search since the height of the resulting tree is smaller.

We need some terminology to be used throughout the paper. Multidimensional data consist of a set of attributes where the number of attributes is more than one. A domain of an attribute is a set of values from which a value for the attribute can be drawn. A domain space is the Cartesian product of domains of all attributes. Any subset of the domain space is called a region (Whang, 1994). Data with n attributes can be represented by points in an n-dimensional domain space.

The basic principle of multidimensional data structures is to divide the n-dimensional domain space into several regions, each containing not more than a fixed number of entries. Each region corresponds to one disk page and, upon becoming full, is split into two. Since all of the multidimensional data structures are characterized by the way they divide the domain...
space and the way they represent the divided regions, the multidimensional data structures can be classified according to the following two properties: whether the domain space is divided into rectangles or not, and whether the division into regions is complete or not, i.e., the union of all divided regions spans the complete domain space or not (Seeger, 1990). Some data structures attempt to represent only the area actually covered by the data in the region rather than the whole area when they represent the region. In such data structures, the union of all regions does not span the complete domain space. According to this classification all known multidimensional data structures including the HG-tree are classified into four classes in Table 1.

In Table 1, the property rectangular means that the data structure divides the domain space into rectangles and the property complete means that the union of all divided regions spans the complete domain space.

The data structures in class C1 perform rather efficiently for uniform and uncorrelated data. However, for highly correlated data their performance degenerates (Seeger, 1990). For example, the directory of the grid file grows exponentially with the dimensionality if a strong correlation among attributes exists. The K-D-B-tree suffers the cascade splitting problem, that is, the split of one directory node causes descendant nodes to be split as well.

The data structures in class C2 adapt to the clustering of objects in the data space by allowing non-rectangular shapes of directory region. However, since they represent the region by the whole area covered by it, they suffer a performance loss in range queries because the number of regions overlapping the query region is increased.

The approach adopted by the data structures in class C3 is to maintain the directory region compactly. This is good especially in the distributions where large portions of empty data space occur. However, it abandons all control over the node occupancy because its region split policy is severely restricted. This reduces the storage utilization of the data structure, and therefore the search performance is degenerated.

The HG-tree in class C4 attempts to solve above problems by allowing more general shape of regions like the data structures in class C2 and by representing the region compactly like the data structures in class C3. In the following section, we describe the HG-tree and discuss algorithms for insertion, deletion, and searches for nearest neighbors and ranges. Section 3 gives the experimental results and analysis of the HG-tree. Final section contains our conclusions.

### HG-Tree

We now discuss the motivations and essential properties of the HG-tree. First of all, let us consider the performance factors that determine the query performance. From the analyses of Kamel and Faloutsos (1993) and Pagel et al. (1993), the numbers of nodes, $D(q)$, accessed by the range query $q$ can be estimated as follows, where the domain space is assumed to be normalized to the unit hypercube:

$$D(\vec{q}) = \sum_{j=1}^{d} \prod_{i=1}^{n} (x_{ij} + q_i)$$

The symbols and their definitions used in Equation (1) are summarized in Table 2.

The equation (1) says that the efficiency of the data structure depends on two parameters, $N$ and $x_{ij}$. It is the storage utilization and the directory coverage of the data structure that determine the values of the parameters $N$ and $x_{ij}$, respectively. As the storage utilization is higher, $N$ becomes smaller and as the directory coverage is smaller, $x_{ij}$ becomes smaller.

We define the storage utilization and the directory coverage as follows.

**Definition 1.** The storage utilization ($U$) of a tree $T$ is:

$$U = \frac{1}{n} \sum_{j=1}^{l} \frac{E_j}{P_j}$$
This title is available in InfoSci-Journals. Recommend this product to your librarian:
www.igi-global.com/e-resources/library-recommendation/?id=2

Related Content

A Framework for Managing Complexity in Information Systems
www.igi-global.com/article/framework-managing-complexity-information-systems/181668?camid=4v1a

Web Service Integration and Management Strategies for Large-Scale Datasets
www.igi-global.com/chapter/web-service-integration-management-strategies/7893?camid=4v1a

Flexible Querying Techniques Based on CBR
www.igi-global.com/chapter/flexible-querying-techniques-based-cbr/20353?camid=4v1a

Semantic Integration and Knowledge Discovery for Environmental Research
www.igi-global.com/chapter/semantic-integration-knowledge-discovery-environmental/4291?camid=4v1a