An Efficient Index Structure for Spatial Databases

Kap S. Bang
Oklahoma State University

Huizhu Lu
Oklahoma State University

In this paper, the authors propose an efficient spatial data structure called the Multi-R tree. The Multi-R tree is an improvement of the R-tree, R+-tree and R*-tree, and can be used as an index structure for spatial databases. The Multi-R tree improves performance by distributing spatial objects into several data spaces instead of one data space in the R-tree, the R+-tree or the R*-tree. Each data space is associated with a tree in the Multi-R tree. The structure of the Multi-R tree eliminates the node redundancy which appears in the R+-tree at leaf level and keeps disjoint intermediate rectangles. A set of new algorithms for the Multi-R tree is also proposed and implemented.

Three popular spatial data structures, the R-tree, R+-tree and R*-tree, are implemented based on the algorithms given in original literature to be compared with the Multi-R tree. An experimental performance analysis for four implemented structures is given with various types of testing data sets: random data, uniformly distributed data, VLSI layout data and TIGER/Line file. Namely, the number of disk accesses and actual response time for each of those four data structures to process a query are compared. Construction times, space utilization and actual memory sizes of the four data structures are also given. Results show that the Multi-R tree requires fewer disk accesses and less processing time than the R-tree, R+-tree and the R*-tree do for a deletion operation and answering a range query in most cases except for a point query or a range query with very small size. In the cases of a point query or small size query processing, the performance of the Multi-R tree is still better than the performances of the R-tree and R*-tree but slightly worse than the R*-tree. Thus, the Multi-R tree may be used as an efficient index structure for spatial databases, e.g., geographical information system, CAD, and VLSI etc.

Since conventional database management systems (DBMS) were developed for business to handle one-dimensional data objects, such as integers, real numbers, or strings of characters, they are not efficient for handling multi-dimensional (spatial) data objects, e.g., boxes or polygons (Neivergelt & Hinterberger, 1984). The spatial data exist in many applications, such as geographic information system (GIS) (Günther & Lambert, 1994; Kasturi, Fernandez, Amlani & Feng, 1989), CAD, and VLSI (Banerjee & Kim, 1986). They are also used in computer vision (Jagadish & O’Gorman, 1989), robotics, and image databases (Grosky, 1989). In large spatial databases, large sets of spatial objects have to be stored in the secondary storage. Therefore, an efficient spatial indexing method for fast retrieval of data should be able to handle this matter. Besides, there exist other factors that affect the performance of the spatial indexing methods, e.g., intra-node search and buffering (Kamel & Faloutsos, 1992). An efficient spatial data structure for handling large spatial data is an important ingredient of special database management systems (DBMS).

In the past decade, many spatial data structures have been proposed. Due to the limitation of the length of the article, we could not present all excellent spatial data structures herein, but only briefly introduce the data structures of the R-tree family: R-tree, R+-tree, R*-tree and parallel R-tree (MXR-tree) which are the background of our proposed spatial data structure. The above spatial data structures are of the catalog of native space indexing method. All native space indexing method preserve proximity (Lomet, 1992) by decomposing original data space on which data objects are drawn.

The R-tree developed by Güttman (1984) is an extension of the B-tree to n-dimensions \((n \cdot 2)\). It is a height-balanced tree with index records stored in leaf nodes contain-
ing pointers to data objects. The R-tree consists of two node types: intermediate and leaf node types. Leaf node entries consist of two parts, (1) a tuple identifier used to refer to a tuple in a database, and (2) coordinates that define an n-dimensional rectangle for enclosing the spatial object. Each entry in an intermediate node consists of (1) a child pointer to the node at a lower level, and (2) coordinates representing a rectangle that completely encloses all rectangles at the lower levels (Güttman, 1984). The R-tree uses non-disjoint space decomposition. That is, the R-tree structure allows overlapping among the intermediate rectangles. In Figure 1(a), intermediate rectangles 7 and 8 overlap and so do intermediate rectangles 8 and 9. If \( k \) overlaps exist in an area and a search range includes that area, it is necessary to search all \( k \) possible paths to find objects overlapping with the search area. To alleviate this problem, a packing technique was proposed in Roussopoulos & Leifker (1985). However, this packing algorithm can not be applied to every insertion (Sellis, Roussopoulos & Faloutsos, 1987).

The R*-tree (Beckmann & Kriegel, 1990) is a more refined variant of the R-tree and also uses non-disjoint space decomposition. In insertion, the R*-tree algorithm selects a node that gives minimal overlap with its neighbors after it is enlarged to include newly inserted object. It uses modified split algorithm to reduce area, margin and overlap. In node splitting, entries in the overflowing node are sorted by using lower and upper coordinates of their rectangles. For each sort, the possibilities of dividing \( C + 1 \) entries into two groups are \( C - 2m + 2 \) where \( C \) denotes node capacity and \( m \) denotes the minimum number of entries in a node (Beckmann & Kriegel, 1990). Among all possible horizontal and vertical distributions, the axis that gives the minimal margin value is selected, and the distribution that gives the minimum overlap value is chosen. The minimum area value is used to break a tie. However, this structure still can not avoid overlapping sub-region problem. The splitting algorithm of the R*-tree requires much more construction speed compared to that of the R+-tree, almost 9 times longer (Hoel & Samet, 1992). The R*-tree takes less storage space than the R+-tree but its performance is not as good as the R*-tree because of the non-disjointed nature of the space decomposition as in the R-tree (Hoel & Senet, 1992). Figure 2(a) shows a distribution of object rectangles in the R*-tree and Figure 2(b) illustrates the R*-tree structure for Figure 2(a).

---

Figure 1: (a) Organization of bounding rectangles (the solid lines construct object rectangles; the dash lines construct intermediate rectangles) and (b) the R-tree structure

Figure 2: (a) Organization of bounding rectangles (the solid lines construct object rectangles; the dash lines construct intermediate rectangles) and (b) the R*-tree structure
Related Content

An Enterprise Model Repository: Architecture and System
www.igi-global.com/article/enterprise-model-repository/3246?camid=4v1a

Multi-Objective Optimization-Based Networked Multi-Label Active Learning
Lei Li, Yuqi Chu, Guanfeng Liu and Xindong Wu (2019). Journal of Database Management (pp. 1-26).
www.igi-global.com/article/multi-objective-optimization-based-networked-multi-label-active-learning/232719?camid=4v1a

INDUSTRY AND PRACTICE: A Metadata Management System to Support Data Interoperability, Reuse and Sharing
Stephanie Cammarata, Iris Kameny, Judy Lender and Corrinne Replogle (1994). Journal of Database Management (pp. 30-42).
www.igi-global.com/article/industry-practice-metadata-management-system/51134?camid=4v1a

Intelligent Search for Experts Using Fuzzy Abstraction Hierarchy in Knowledge Management Systems
www.igi-global.com/article/intelligent-search-experts-using-fuzzy/3374?camid=4v1a