Modify Compact R-tree Dynamic Index Structure for Myanmar GIS Database

Su Nandar Aung, Myint Myint Sein
University of Computer Studies, Yangon
sunandaraung @ucsy.edu.mm , myint@ucsy.edu.mm

Abstract

Spatial databases have been increasingly and widely used in recent years. In order to handle spatial data efficiently, as required in CAD and GIS applications, a spatial database system needs an index mechanism that will help it retrieve data item quickly according to their spatial locations. However, traditional indexing methods are not well suited to data objects of non-zero size located in multi-dimensional spaces. In this paper, we propose a category-based compact R-tree structure based on the category of services of Yangon region. The propose system can achieve almost 100% storage utilization and reduce time of search, insert and delete operation. The cost of searching time in this system is faster than normal compact R-tree which uses one index structure for all. The propose system is also aimed to reduce integrating errors of images contain the user location and current location and cover for all categories in desire range without searching the specific categories.

Keywords: Spatial Database, Spatial Indexing Structure, R-tree, Compact R-tree, Spatial Query.

1. Introduction

Spatial database systems manage large collections of geographic entities, which apart from spatial attributes contain non spatial information. Typical applications of spatial databases include VLSI circuit layout data in CAD and cartography in a Geographic Information System (GIS). Spatial objects in reality are associated with multiple quality attributes in addition to their spatial locations. Traditional spatial queries and joins focus on manipulating only spatial locations and distances. For example, when an area of interest is found while browsing a digital map or a circuit layout, we may draw a query window to inscribe that area and ask the system to retrieve all detailed information about the countries in the map or the components in the circuit layout which are inside or overlapped with the query window.[6] Thus an index structure based on objects' spatial locations is desirable but classical one-dimensional database indexing structures are not appropriate to multi-dimensional spatial searching. Structures based on exact matching based on exact matching of values, such as hash tables, are not useful because a range search is required. [2]

In recent years, a number of structures have been proposed for handling multi-dimensional data. R-tree and compact R-tree are the most useful dynamic index structure for efficiently retrieving objects from a spatial database according to their spatial locations. It has been widely used to index the spatial objects in a large pictorial database such as the GIS applications.

2. Spatial Indexing

A fundamental issue in spatial database is how to store, search and delete spatial data efficiently. In order to handle spatial data efficiently, a spatial database needs an index structure. A good index structure has ability to collect similar data into same portion. SDBMS supports spatial data models, spatial indexing. Conventional DBMS are unable to support spatial data processing efficiently for the following reasons:

1) First, spatial data are large in quantity, complex in structures and relations
2) Second, the retrieval process employs complex spatial operators like intersection, adjacency, and containment and
3) Third, it is difficult to define a spatial ordering, so conventional techniques cannot be employed for spatial operations.

For the above reasons, spatial indexes need for spatial data of SDB. [4]

2.1. R-tree and Its Variations

The R-tree is a height-balanced tree with index records in its leaf nodes containing pointers to data objects. For a large number data objects, the R-tree itself is also large and should be disk-resident so that a node in an R-tree corresponds to a disk block. The R-tree index structure is completely dynamic because insert and deletes can be intermixed with searches and no periodic reorganization is required.

A spatial database consists of a collection of spatial objects and each object has a unique identifier that can be used to retrieve it. Leaf nodes in an R-tree contain index record entries of the form *(I, obj-id)*, where *I* is the minimum bounding rectangle (MBR) and *obj-id* is the identifier of the spatial object being indexed. A non-leaf code contains entries of the form *(I, child-ptr)*, where *child-ptr* is the address of a lower node in the R-tree and *I* covers all rectangles in the lower node's entries. Let *M* be the maximum number of entries that will fit in one node and *m* be a parameter specifying the minimum number of entries in a node such that $2 <= m <= M/2$. Thus an R-tree satisfies the following properties:

1) The root has at least two children unless it is also a leaf node.
2) Every non-root node has between m and M entries.
3) All leaves appear on the same level.

There are several variations to R-trees such as R+-tree, R*-tree and packed R-tree, etc. They all try to improve the search performance by minimizing "coverage", "overlap", or circumference of directory rectangles. [2]

2.2. Compact R-tree

The simplicity and low implementation cost of GUttman's R-tree structure make it more popular than R+-tree and R*-tree structures. However, all the above three dynamic structures suffer from low storage utilization problem. Thus the main goal of compact R-tree structure is to maximize the storage utilization while retaining the same search performance as in Guttman's R-tree. The method of building a compact R-tree is different from that of building a Guttman's R-tree. In a dynamic environment, the mechanisms of maintaining a dynamic index structure are invoked by the insert and delete operations. Inserting index records for new data objects in an R-tree structure is similar to insertion in a B-tree in that new index records are added to the leaves, nodes that overflow are split, and splits may propagate up the tree.

The compact R-tree removes an entry from an overflowed node to an under-full sibling node and installs the new entry in the evacuated spot. There are three different situations when an insert operation is performed.

1) Trying to insert a new entry into an under-full node. No split occurs.
2) Trying to insert a new entry into a full node and there is at least one under-full node at the same level: An entry is selected from the set which contains all the entries in the target full node and the entry to be inserted. Add the selected entry to the most appropriate under-full sibling node.
3) Trying to insert a new entry into a full node and all sibling nodes are also full: The target full node splits to accommodate the new entry and the split may propagate upward. [6]

3. Proposed System

The overview of the proposed system is shown in Figure1. According to the figure, the system starts with user's desired query with (lat/long) is input to the database. By querying the desired lat/long from the relevance category-based index, the database output images and the corresponding category of services within given range.

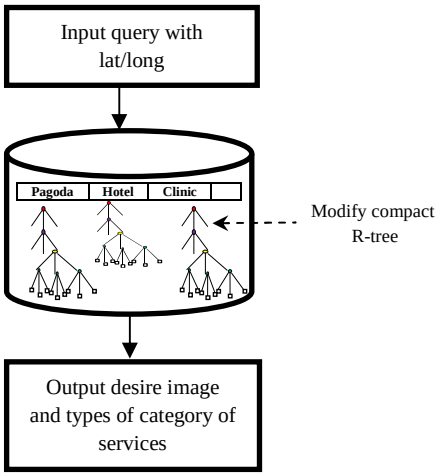

Figure1. System overview

In the propose system, the main contribution is to create category-based compact R-tree index structure for efficiently insert, delete and search for spatial data from spatial database. The own Data-Set is created for Yangon region which containing the Latitude, Longitude, Position, Category and Images grabbed from Google Earth. The propose index structure reduce the time cost more than normal compact R-tree which use one index structure for all categories.

Moreover, the propose system can easily specify what category of services contain in given range. As compact R-tree uses one index for all objects, it can find one object per one time. It can increase computational overhead for searching in one large index. The proposed system uses many indices for each category of services, it can use parallel at the same time and mitigate on disk lookup bottleneck.

Some example category of services in Yangon downtown region is shown in table1.

Table 1. Example categories of Yangon downtown region

Air Port	Embassy	Photo Studio
Ticket Shop	Police Station	ATM Service
Fire Station	Hospital	Post Office
Bank	Gas Station	Press Offset
Book Shop	Garden	Residence
Bus Station	Clinic	Restaurant
Pet Shop	Hospital	University
Pagoda	Cinema	Hotel
Gold & Jewellery	Mobile Sale & Service	Computer (Sale,Service)

Example query:

Selection query: "*Find all services within {[30,104], [0, 80]}*"

Nearest Neighbor-queries: "*Find the closets services to a query point q within {[30,104], [0, 80]}*"

Range queries: "*find all hotels in {[30,104], [0, 80]}*"

Aggregation queries: "*find the total number of hotels within {[30,104], [0, 80]}*"

Skyline queries:" *Which building can we see within {[30,104], [0, 80]}?*"

The propose index structure can efficient operate selection query, range query, NN query, aggregation query and skyline query with minimum time consuming. Compact R-tree which use one index for all query is used for window query consume time because it search the whole nodes in the index until it meet the given query's requirement. It can search many nodes that don't concern for the given query. As a reason, it cost many time and computational overhead for searching.

4. Experimental Results

In table 2, we can see a comparison on the response time (in second) for the queries search on compact R-tree and category-based compact R-tree.

Table2. Average time for compact R-tree and proposed tree

Number of Data objects	Compact R-tree	Category-based Compact R-tree
1000	0.50	0.20
2000	1.89	0.36
3000	2.2	0.80
4000	2.6	1.40

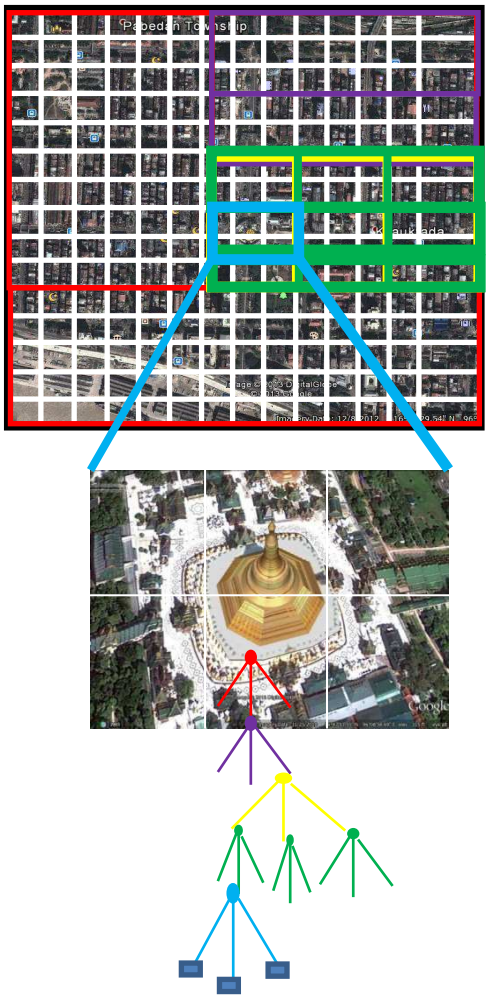

Figure2. Proposed index structure

5. Conclusion and Further Work

This paper presented index structure for optimizing time which is dynamic index structure for spatial database. This structure can achieve more optimal storage utilization (100%) than R-tree and can reduce searching and computational overhead than R-tree and compact R-tree. Many further extensions can be considered for efficient index structure for spatial database. As a further extension, we'll add an efficient method to answer top-k spatial keyword queries in this proposed index structure.

References

[1] P.Patel and D.Garg. Comparison of Advance Tree Data Structures. International Journal of Computer Applications (0975-8887) Volume 41-No.2, March 2012.

[2] A.Guttman. R-tree: A Dynamic Index Structure for Spatial Searching, Proc. ACM SIGMOD, pp. 47-57, 1984.

[3] Y.Gui-jun and Z.Ji-xian. A Dynamic Index Structure for Spatial Database Querying Based on R-trees.

[4] B.Chin Ooi, R.Sack-Davis and J.Han. Indexing in Spatial Databases.

[5] S. Krishnaveni and S.P.Tamizhselvi. Indexing and Ranking in Spatial Database. Proc. Journal of Computer Applications ISSN: 0974-1925, Volume-5, Issue EICA2012-5, February 10, 2012.

[6] P.W.Huang, P.L.Lin and H.Y.Lin. Optimizing storage utilization in R-tree dynamic index structure for spatial databases. Proc. The Journal of Systems and Software 55 (2001) 291-299.