ISSN No. 0976-5697

Volume 2, No. 5, Sept-Oct 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Performance Evaluation Of Spatial Indexing Techniques

S Mahaboob Hussain*
Dept. of CSE,
UCEV, JNTUK
Vizianagaram, A.P., India
mahaboobhussain.smh@gmail.com

Pullela S V V S R Kumar
Associate Professor of CSE
VS Lakshmi College of Engg for Women
Kakinada, A.P., India
pullelark@yahoo.com

Dr MHM Krishna Prasad
Associate Professor & Head
Dept. of IT,
UCEV, JNTUK
Vizianagaram, A.P., India
krishnaprasad.mhm@gmail.com

Abstract: Spatial databases store information related to objects positional locations e.g., various kinds of multidimensional objects represented by points, line segments, polygons and other kinds of geometric entities. These databases should support for efficient storage, indexing and querying of spatial data. Special purpose indexing structures are important for accessing the spatial data, and for processing spatial queries. To handle the spatial data efficiently, a database system needs an indexing mechanism that will help it to retrieve data items quickly according to their spatial locations instead of using GIS (Geospatial Information System). Hence, in this paper, authors evaluated R, R+ and R* dynamic indexing algorithms, on a variety of real time and synthetic datasets, to access the spatial data. From the experimental study, one can observe that the R* indexing technique performs well (than R and R+) for spatial databases.

Keywords: Spatial Databases, Indexing, Querying, Splitting Algorithms, RTree, R*Tree, R+Tree

I. INTRODUCTION

In order to deal with multidimensional data, some spatial access methods (*hereinafter*, SAMs) are consider, these are desirable to design spatial data management systems to support and perform spatial operations fastly. Spatial databases contain multidimensional data with explicit knowledge about objects, their extent, and their position in space. Multidimensional data include which points, line segments, rectangles, polygons, regions, volumes, and polyhedral in 2D, 3D or higher. Several Multidimensional Access Methods, some general purpose and some application specific, that support search operations in spatial databases have been proposed and evolved for the last 30 years [1]. In this paper, some data tree structures (R, R*, R+) and are discussed and presented a possible high performance spatial access method for the two dimensional datasets.

The discussion can be carried on at least two levels. At the implementation level anxiety may be over word extent, record types and file structures. R tree and its variants (R*Tree, R+Tree) are one of the prominent spatial access methods, where the objects stored in R tree are in the context of Minimum Bounding Rectangles (MBR). A rectangle, oriented to the x and y axis, which bounds a geographic feature or a geographic data set. It is specified by two coordinates: x_{min},y_{min} and x_{max},y_{max} [2]. R-tree in every node stores a set of rectangles. At the leaves there are stored pointers to representation of polygon's boundaries and polygon's MBRs. At the internal nodes each rectangle is associated with a pointer to a child and represents minimum bounding rectangle of all rectangles stored in the child.

II. SPATIAL INDEXING/ACCESS METHODS

A. *The R-Tree:*

The R-tree [3] is the origin of all R-tree variants. These R-trees are hierarchical tree data structures, meant for efficient indexing of multidimensional objects with spatial extent. Similar to B-trees [4, 8], the R-trees are balanced and they ensure efficient storage utilization. R-trees are used to store, instead of the original space objects, their minimum boundary boxes (MBBs). The MBB of an n-dimensional object is defined to be the minimum n-dimensional rectangle that contains the original object. The R-trees manage MBBs and not real objects, thus they cannot fully answer a query, unless the objects in the database are equal to their MBBs. In general, they are used to efficiently solve the filter step of a query that is finding the database objects whose MBB intersects with the MBB of the query object.

a. Adopted Structure of the R-Tree.: An R-tree is a height-balanced tree similar to a B-tree with index records in its leaf nodes containing pointers to data objects nodes correspond to disk pages. If the index is disk-resident, and the structure is designed so that a spatial search requires visiting only a small number of nodes. The index is completely dynamic inserts and deletes can be intermixed with searches and no periodic reorganization is required. The data structure splits space with hierarchically nested,

and possibly overlapping, minimum bounding rectangles (MBRs, otherwise known as bounding boxes, i.e. "rectangle", what the "R" in R-tree stands for). The Fig. 1, shows the structure of an R-tree and illustrate the containment and overlapping relationships that exist between its rectangles. The height of an R-tree containing N index records is at most *|logmN|-1*, because the branching factor of each node is at least m. The maximum number of nodes is *|N/M|+|N/M2|+1*. Worst-case space utilization for all nodes except the root is *m/M [3]*.

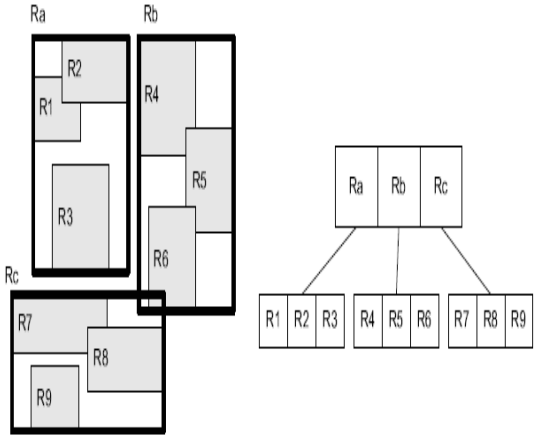

Figure 1: Adopted structure of the RTree

Nodes will tend to have more than *m* entries, and this will decrease tree height and improve space utilization. If nodes have more than 3 or 4 entries the tree is very wide, and almost all the space is used for leaf nodes containing index records. The parameter *m* can be varied as part of performance turning, and different values are tested experimentally. In this paper, we adopted the algorithms proposed by [3] for constructing the tree, inserting and deleting the elements.

B. The R+-Tree:

A variation to Guttman's R-trees (R+ trees) introduced by Sellis et al.,[5] as a way to overcome the problem of inefficient searching that arises when sibling nodes (overlapping rectangles) overlap in the R-tree. As a direct solution to these problems they use *clipping*, i.e. there is no overlap between intermediate nodes of the tree at the same level, and objects that intersect more than one MBB at a specific level are clipped and stored on several different pages. As a result, point queries on the R+ tree require traversing only one path of the tree, which is to be achieved by increase of storage requirement of the tree.

a. Adopted Structure of the R+Tree: As mentioned above, R-trees are a direct extension of B-trees in k-dimensions. The data structure is a height-balanced tree which consists of intermediate and leaf nodes. Data objects are stored in leaf nodes and intermediate nodes are built by grouping rectangles at the lower level. Each intermediate node is associated with some rectangle which completely encloses all rectangles that correspond to lower level nodes. In this paper, we adopted the algorithms proposed by [5] for constructing the tree, inserting and deleting the elements. Following Fig. 2 shows an example set of data rectangles.

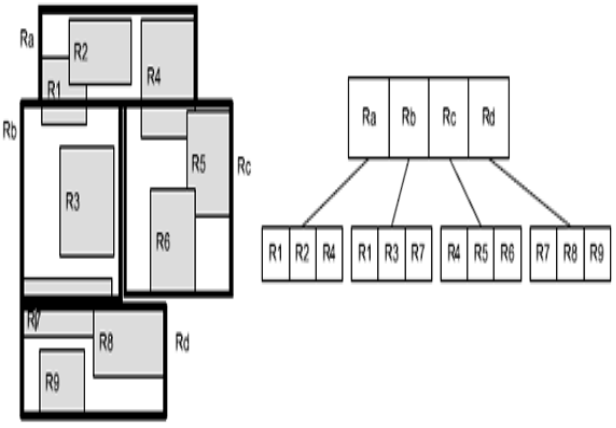

Figure 2: Adopted structure of the R+Tree

C. The R-Tree:*

To work on the improved version of the R-Tree we implemented its variant R* tree [6] and the insertion algorithms are adopted to overcome the weakness of the original R-tree. This version introduces a new insertion policy that crucially improves the performance of the tree. The main objective of this policy is to minimize the overlap region between sibling nodes in the tree. A straightforward advantage of this is the minimization of the tree paths that are traversed at an object search. In this paper, we adopted the algorithms proposed by [6] for constructing the tree, inserting and deleting the elements.

a. Description: While traversing the insertion path, the insertion algorithm follows the nodes, whose MBB has the minimum increase of overlap. Thus, the search performance is improved [7]. Whenever a new entry has to be stored into a full node, the node is not necessarily split, but some entries are deleted, and re-inserted to sibling nodes. The entries for re-insertion are chosen to be those with maximum distance from the centre of the node's MBB. This feature increases storage requirement, and improves the quality of the partition by making it almost independent of the sequence of insertions. That results in a minimum overlap between the MBBs.

The insertion algorithm stores the lower and the upper bounds of the MBR for each dimension to two arrays. It then sorts these two arrays d times, according to one dimension at a time, and finds the best axis according to which the split will occur, by finding the best possible split distribution. The criteria for the best split include the minimization of the margin, area, and overlap between the sibling nodes to be created. The function stores in distribution array the sorted indices of MBR's and returns the index within dimension according to which the split should take place. The function returns in distribution (0) an

ordering of the MBR's and in dist the index within this order, with respect to which the split will take place.

III. EXPERIMENTAL EVALUATION

This section presents the experimental observations obtained while evaluating the behavior of the spatial access methods: the R-tree, R+ Tree and R* Tree.

A. Implementation Details and Experimental Platform:

The experimental platform is a Intel Pentium IV (R) Core(TM)2 Duo CPU E7500@ 2.93GHz machine with 2GB main memory and a 256GB HDD, running on Microsoft Windows XP.

In this phase of research, authors adopted the quadrant split algorithm [3] for comparing most popular R-Tree and its variants R+-Tree and R*- Tree.

JDK based interface is developed to make simpler for the evaluating performance of these three SAM in a single shot. Below Fig. 3, shows the designed interface with the options to select the trees and datasets to insert into the trees and also shows the performance of the each tree.

In this paper, authors experimented the above indexing techniques on the multi dimensional spatial datasets which is generated synthetically of different sizes varying from *10K* to *300K*, where *K=1000*, instances.

B. Performance Tests and Results:

In general, there are four principal ways of comparing algorithms such as indexing techniques [9]: i.e., by direct argument, by mathematical modelling, by simulation, and by experiment. In this section, authors consider the experimental approach to evaluate the techniques.

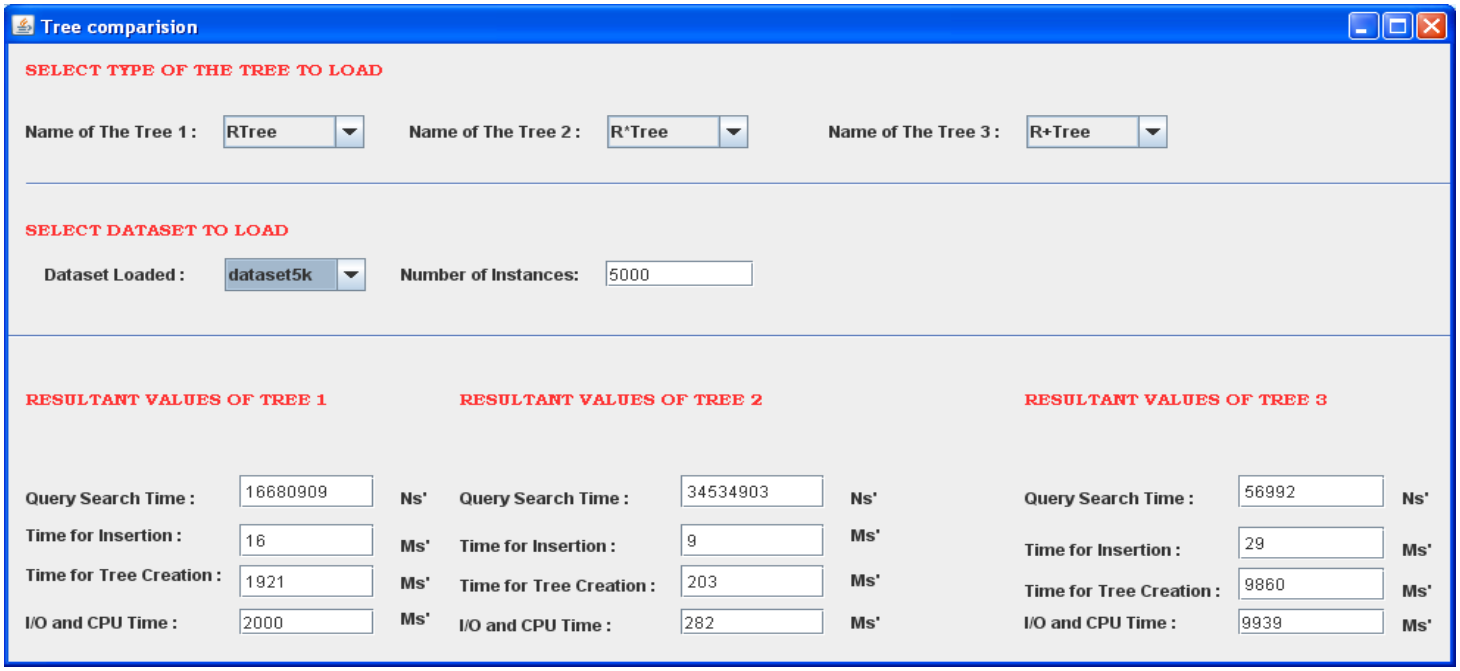

Figure 3. Design of the interface

The earlier discussed R Tree, its variants R+ Tree and R* Tree structures for accessing the spatial data structures, which are implemented with the efficient node splitting algorithms for inserting the geometrical data in the each of the tree structure with 2-dimensional spatial data. The performance of the insert algorithm in these tree structures are done by using the hash index file, which maintains each entries of the co-ordinate. It is a cache of the all objects and is updated with the new objects. For simplify, the following method will use the R-Tree, R+-Tree and R*-Tree to index the data file's page life, and the SAM leaf page's *PID* and *entry id* in the current data file tuple.

To analyze the performance for building up the different R-tree variants and measured the performance for insert and store. Here, insert denotes the average number of disk accesses per insertion and store denotes the storage utilization after completely building up the files. During the first part of each test run, the program read geometry data from files and constructed an index tree, beginning with an empty tree and calling insert with each new index record, then the insert performance was measured.

While traversing the insertion path, the insertion algorithm follows the nodes which are having Minimum Bounding Box (MBB) and minimum increase of overlap. Thus, the search performance is improved. As the R*-Tree algorithm for splitting a node is totally different from its R-tree equivalent. First, the algorithm decides the axis with respect to which the split will take place. Then, the projections of the MBBs over the split-axis are sorted according to the value of their left end point. This sequence can be divided to two sub-sequences, in *M-2m+1* ways. Among these splits, the algorithm chooses the best one.

The results are given in tabular forms and graphs that show's an overview of experiments depending on the different distributions (i.e., data files) for the trees. By taking the

different size of the data and inserting into the tree structure authors got different output values in milliseconds (*ms*) and calculated the searching for the range query in nanoseconds (*ns*). And authors compare the results for time taken for tree creation and total building time for all SAMs.

C. Insertion/Creation Performance:

In this paper, authors computed the insertion time for the given data files (Spatial data sets) and obtained the tree creation time is similar as the insertion time. The following table 1, shows the tree creation time for different SAMs.

Table 1. Time(ms) variation for Tree creation.

Instances/SAM	R-Tree	R^+-Tree	R+-Tree
1K	297	62	765
5K	1859	203	11266
10K	3984	375	43812
20k	7063	672	176813
30k	11734	1531	238781
40k	15531	1906	333515
50K	19656	1656	410547
100k	40547	3765	659250
200k	69906	6562	1144235
300k	118375	11500	1460765

Due to the variation of the node splitting algorithm and the insertion algorithm procedures in the three trees, one can observe the differences in the time variation for creating the tree structures for different size of spatial datasets. The following Fig. 4 shows the time taken for three SAMs for inserting the datasets of the size varying from 1K to 300K instances respectively.

D. Query/Search Performance:

Authors compare the performance of the R-tree variants for range and nearest-neighbour queries. In query performance R+Tree overcome the problem of inefficient searching that arises when sibling nodes overlap in the R-tree, whereas R Tree Searching is done in a similar way as in a B-tree, for both point and region queries, the paths where rectangle intersects with the query object are followed and it does not traverse one path of the tree is enough when searching for an objects in the B Tree, as the MBBs of entries in the same nodes may overlap one another. In the worst case, the search algorithm may have to visit all index pages, in order to answer a query. The following table 2 shows the time taken in *nanoseconds* for the query operation in the proposed spatial indexing structures.

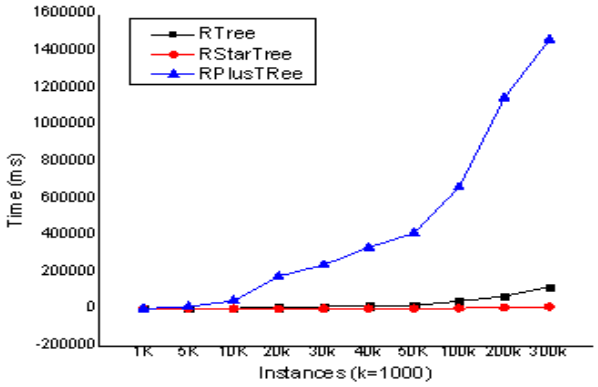

Figure 4. Time(ms) variation for Tree creation

Table 2. Time (ns) Query/Search Performance

Instances/SAM	R-Tree	R^+-Tree	R+-Tree
1k	1668.0909	3453.4903	5.6992
5k	2247.3824	2963.1391	5.4913
10k	2320.1571	3106.4919	5.8195
20k	3015.3512	3178.0802	5.5875
30k	2207.1128	3290.4794	55375
40k	2173.3573	3946.8331	5.4041
50k	2343.0449	4612.488	5.3628
100k	2430.0082	3671.0606	9.8788
200k	2584.709	2843.5181	10.1266
300k	2846.8876	3640.0484	9.9973

The figure 5, shows the experimental observations for various range querys. From figure 5, one can clearly observe that the R+Tree performance is better than the other two trees for Query performance.

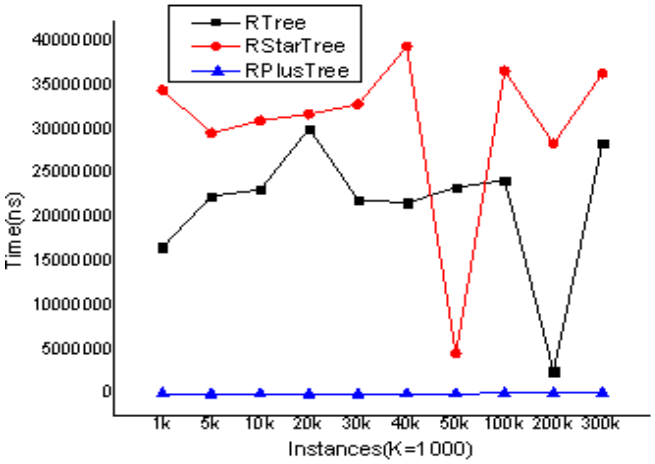

Figure 4. Time (ns) Query/search performance

The main advantage of R+Tree compared to R-trees is the improved search performance, especially in the case of point queries and the main reason is, there is no overlap between intermediate nodes of the tree at the same level, and objects that intersect more than one MBB at a specific level are clipped and stored on several different pages. RTrees suffering the case of few, large data objects, which force a lot of "forking" during

the search [3]. R+-trees handle these cases easily, because they split these large data objects into smaller ones. As a result, point queries on the R+Tree require traversing only one path of the tree. The price to pay is the increase of storage requirement of the tree.

E. CPU/IO Cost Performance:

In this paper, authors consider the total time taken for the tree to build as CPU/IO cost. In this evaluation the performance of the R*Tree is comparatively better than the other indexing techniques. In table 3 authors shows the total building time for the each Tree data structure for different size of 2-dimensional spatial datasets.

Table 3. CPU/IO Cost Performance

Instances/SAM	R-Tree	R^+-Tree	R+-Tree
1K	375	297	734
5K	2047	469	4132
10K	3782	734	8906
50K	20000	3078	38984
100K	38859	5672	83719
200k	66531	9891	169094
300k	104531	15469	256453
100K	38859	5672	83719
200k	66531	9891	169094
300k	104531	15469	256453

Figure 6 shows the time taken for building the trees for 2-dimensional dataset of size varying from 1K to 300k of instances.

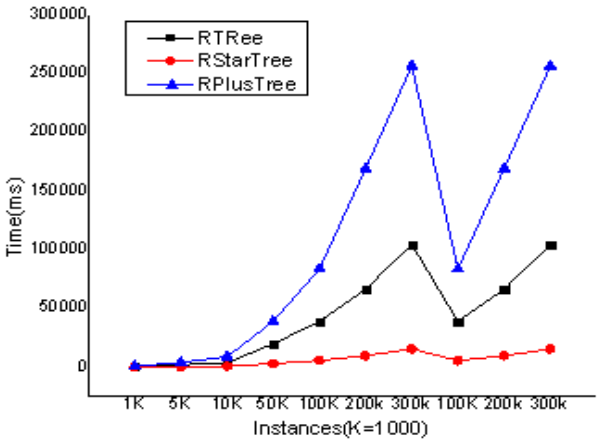

Figure 5. CPU/IO Cost Performance

The R-tree clearly out performs the R-tree variants in all experiments*. But specifically for querying, R+ Tree performs well than R Tree and R*Tree, as per authors observation, it is due to the non-overlapping rectangles in the R+ Tree, makes R+ Tree efficient for searching.

Finally, the following table 4 shows the overview of observation on 2-dimensional spatial data structures.

The performance results are shown in the following table 4.

Legend: ">" means "the performance is better than".

Table 4. Performance and experimental results

SL.NO	CONDITIONS	RESULTS/OBSERVATIONS
1	Insertion time	R*tree > RTree > R+Tree
2	Creation time	R*tree > RTree > R+Tree
3	Deletion time	R*tree > RTree > R+Tree
4	CPU/IO time	R*tree > RTree > R+Tree
5	Query (Range) Time	R+Tree > RTree > R*Tree

IV. CONCLUSIONS & FUTURE WORK

Spatial databases store information related to objects positional locations e.g., various kinds of multidimensional objects represented by points, line segments, polygons and other kinds of geometric entities. The performance of spatial queries depends mainly on the underlying index structure used to handle them. The main disadvantage of the R-tree is it suffers largely from high overlap and high coverage, resulting mainly from splitting the overflowed nodes. A database system organizes both for multidimensional points and spatial data as demonstrated. From the experimental observations, which is performed on rectangle data, the R* tree clearly outperforms and it is the most robust method, which is underlined by the fact that for every query file and every data file less number of disk accesses are required than R-Tree and R+Tree. Moreover, for spatial data the gain in performance of the R*Tree over the other variants is increased additionally.

In this paper, authors worked on the rectangles for the given 2-dimensional spatial data objects. Further work in this area should deal with more complex spatial objects such as polygons as, where only very few access methods are available.

V. REFERENCES

[1] H. Ahn and N. Mamoulis (2001), Ho Min Wong, A Survey on Multidimensional Access Methods, UU_CS_2001_14, May, 2001.

[2] MiMi.hu,"Minimum Bounding Rectangle," MiMi.hu, http://en.mimi.hu/gis/minimum_bounding_rectangle.html.

[3] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," Proc. ACM SIGMOD International Conference on Management of Data, 1984, pp. 47–57. ISBN 0-89791-128-8

[4] Bayer and Rudolf (1971), "Binary B-Trees for Virtual Memory," Proc. of ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, California. November 11–12, 1971

[5] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R+-Tree: A dynamic index for multi-dimensional objects," Proc. of 13th International Conference on Very Large Data Bases, 1987, pp. 507-518.

[6] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," Proc. ACM SIGMOD International Conference on Management of Data, 1990, pp. 322-331.

[7] N. Roussopoulos and D. Leifker, "Direct Spatial Search on Pictorial Databases Using Packed RTrees," Proc. ACM SIGMOD International Conference on Management of Data, 1985, pp. 17-31.

[8] D. Comer, "The Ubiquitous B-Tree," Computing Surveys vol. 11, pp. 121-138, June 1979.

[9] J. Zobel, A. Moffat and K. Ramamohanarao, "Guidelines for Presentation and Comparison of Indexing Techniques," Proc. ACM SIGMOD, 1996.