Review for Best Multidimensional Index Structure

Ochin Sharma, Krishan Kumar
Manav Rachna International University, Faridabad, India

***Abstract*:** Whenever we deal with the one dimensional data, it is straight forward but when it comes to dealing with multidimensional data; several challenges appears regarding storage, operational optimization and performance. A lot of work has been carried out for flawless storing, inserting and deleting of high dimensional data that is indeed huge and very large and to be getting accommodated in an efficient manner for optimized performance. 'R tree' and its variants are very useful to process multidimensional data. In this paper, a research work has been carried out o survey various 'R Tree variants' to process multidimensional data objects.

Index Terms—Tree, Index, Multidimensional, R-Tree, PR Tree

I. INTRODUCTION

One dimensional index structures assume a single search key, and retrieve records that match a given search-key value. (The search key can be a single field or a combination of fields). Many applications, e.g. CAD, OLAP, multimedia require us to view data as two or more dimensions [4,5].
The queries to be supported on such data are partial-match queries: specify values for a subset of the dimensions, range queries: give the range for each dimension, nearest-neighbor queries: ask for the closest point to the given point. The possible solution can be:

A. Brute force
All cordinates are arranges sequentially and searched.
B. Projection
All cordinates are sorted on the basis of any criteria or attribute and arranged sequentialy.
C. Multikey access
In this we reference the records R in file F by using any possible subset of these fields (key), as shown in the following examples:
a) Entire record specified (exact match query, point query)
b) 'Rohit' named person born in 1951 (a partially specified query)
c) All records with last name 'Kumar' (single-key query)
d) Multimedia all objects within specified coordinates.
e) Everyone born between 1920 and 1927 (range or interval query)

In space each of the regions can be thought of as a rectangle, and each of the points in that region has its record placed in a block belonging to that rectangle [6]. If needed, overflow blocks can be used to increase the size of a rectangle. Consider a 2D data as:

Age	Salary
26	60
45	60
50	75
50	100
50	120
70	110
85	140
30	260
25	400
45	350
50	275
60	260

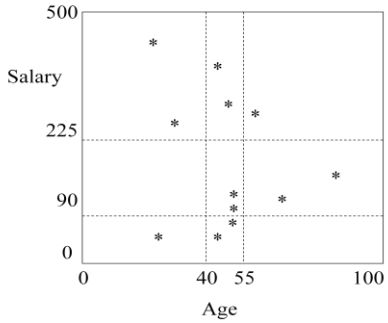

Fig 1

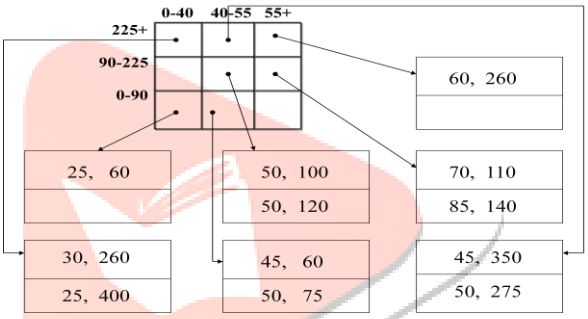

Fig 2

II. VARIOUS INDEXED STRUCTURES

A) R Tree

R-trees are data structures used for indexing multi-dimensional information such as geological coordinates, rectangles and polygons. The R-tree was projected by Antonin Guttman in 1984 and has originated significant use in both theoretical and practical contexts. The real world application of an R-tree might be to store high dimensional data or spatial objects such as hospital locations or the specific maps of roads, buildings, markets or nearby restaurants etc. and then find answers quickly to queries such as to find all restaurants within 2 km of the current location and to display the navigation path[1].

Actually R-tree has two main disadvantages as:

1. The execution of a point location query in an R-tree may lead to the investigation of several paths from the root to the leaf level. This characteristic may lead to performance deterioration, specifically when the overlap of the MBRs is significant.
2. A few large rectangles may increase the degree of overlap significantly, leading to performance

degradation during range query execution, due to empty space.

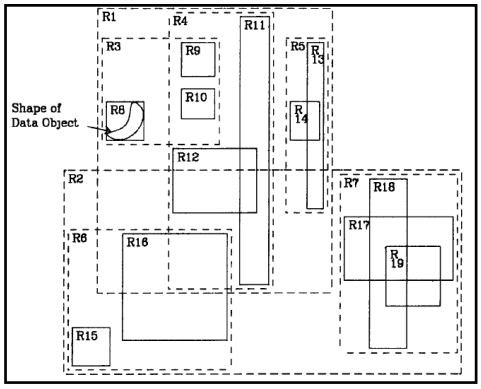

Fig 3: MBR representation in Rtree scheme

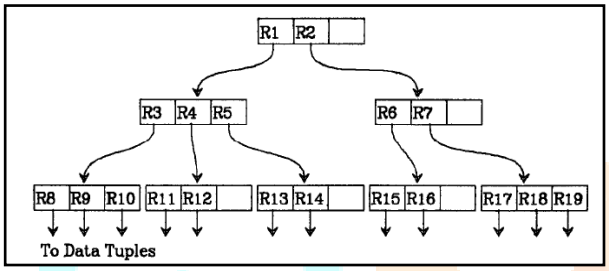

Fig 4: Indexed structure in RTree

B) The R⁺ tree

R⁺ trees [2] are a structure that avoids visiting multiple paths during point location queries, and thus the retrieval performance is improved. In addition, MBR overlapping of internal modes is avoided.
This is achieved using the clipping technique. R⁺ tree does not allow overlapping of MBRs at the same level of tree.

Algorithm Insert (*R*, *IR*)
Input:An R⁺ tree rooted at node *R* and an input rectangle *IR*
Output:
The new R⁺ tree that results after the insertion of *IR*
Procedure:
Find where *IR* should go and add it to the corresponding leaf nodes.
1. If *R* is not a leaf, then for each entry (*p*, *RECT*) of *R* check if *RECT* overlaps *IR*. If so,
Insert (*CHILD*, *IR*), where *CHILD* is the node pointed to by *p*.
2. If *R* is a leaf, add *IR* in *R*. If after the new rectangle is inserted *R* has more than *M* entries, **SplitNode**(*R*) to re-organize the tree

Algorithm Search (*R*, *W*)
Input:
An R+ - tree rooted at node *R* and a search window (Rectangle) *W*
Output:
All data objects overlapping *W*
Procedure:
Decompose search space and recursively search tree
1. [Search Intermediate Nodes]
If *R* is not a leaf, then for each entry (*p*, *RECT*) of *R* check if *RECT* overlaps *W*. If so,
Search(*CHILD*,$W \cap_{RECT}$), where *CHILD* is the node pointed to by *p*.
2. [Search Leaf Nodes]
If *R* is a leaf, check all objects *RECT* in *R* and return those that overlap with *W*.

At some stage in the implementation of the insertion algorithm a node may become full, therefore no further entries can be store in it. So for this, a node splitting method is necessary the same as in the R-tree case. The major difference between the R⁺ tree split algorithm and that of the R-tree is that downward chaining may be necessary, as well as the upward chaining.Recollect that in the R-tree, upward chaining is enough to guarantee the structure's integrity.

C) The R-tree*

R*trees [3] were projected in 1990 but are very well accepted in today's literature as a existing performance-wise structure that is Many times used for performance comparisons. As discussed, the R-tree is based completely on the region minimization of all MBRs. Alternatively, the R*tree goes ahead of this criterion and examines quite a few others. The criteria taken by the R*tree is the following:

- *Minimum region covered by each MBR:* This measure aims at minimizing the region enclosed by MBRs but not by the rectangles, to decrease the number of paths pursued during query processing.

- *Minimum overlap among MBRs:* More the overlapping among MBR, More is the likely number of paths followed for a query.

- *Minimum of MBR margins:* This measure aims at shaping additional quadratic rectangles to get better the performance of queries those have a large quadratic shape. Additionally, minimum region can be achieved as quadratic objects can be grouped more easily and the subsequent MBRs on higher levels are likely to be smaller.

- *Maximum storage utilization:* Whenever node storage utilization is small the height of the tree increases due to additional nodes to accommodate data and hence time taken to process the query increases. This is true

particularly for multi dimensional query where sufficient segments of the entry persuade the query.

D) PR-trees

The Priority R-tree [4] is a worst-case optimal substitute to the spatial tree R-tree, proposed by Arge, De Berg, Haverkort and Yi, K. in an article in 2004. The prioritized R-tree is fundamentally a mix of k-dimensional tree and a R-tree. The term priority refer from four priority-leaves that represents the most extreme values of all dimensions, incorporated in each branch of the tree. While answering a query by investigating the lower-branches, the prioritized R-tree initialy checks for overlap in its priority nodes. The lower-branches is checked if the smallest value of the first dimension of the query is more than the value of the lower-branches. This gives access to a fast indexation with the value of the first dimension of the bounding rectangle.

In the Priority R-tree structure the bulk-loading algorithm utilizes priority rectangles. Window queries can be answered in on a PR-tree $O((N/B)^{1-1/d}+T/B)I/Os$ and the index is thus the first R-tree variant that answers queries with an asymptotically optimal number of I/Os in the worst case. For simplicity, it first describes a two-dimensional pseudo-PR-tree. The pseudo-PR-tree answers window queries efficiently but is not a real R-tree, since it does not have all leaves on the same level. Next it obtains a real two-dimensional PR-tree from the pseudo-PR-tree. A PR-tree on a set of N hyper-rectangles in d dimensions can be bulk loaded in $O(N/B\log_{M/B}N/B)I/Os$, such that a window query can be answered in $O((N/B)^{1-1/d}+T/B)I/Os$ where T is the number of reported rectangles, I/O is Input / Output, where each node (except for the root) has degree Θ(B). Each leaf contains Θ(B) data rectangles and all leaves are on the same level of the tree. If B is the number of rectangles that fits in a disk block, an R-tree on N rectangles occupies Θ(N/B) disk blocks and has height Θ(logB N).

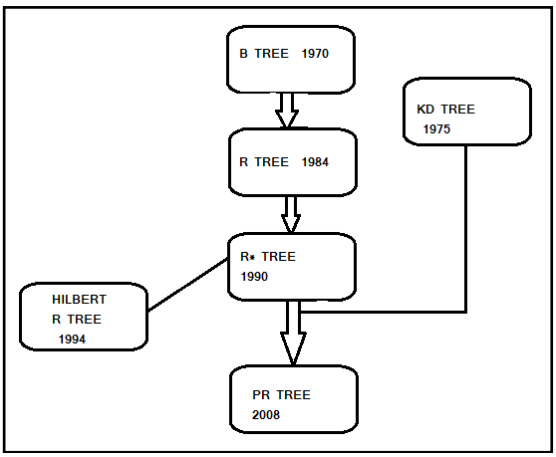

Fig 5: Historical Progress upto PR Tree

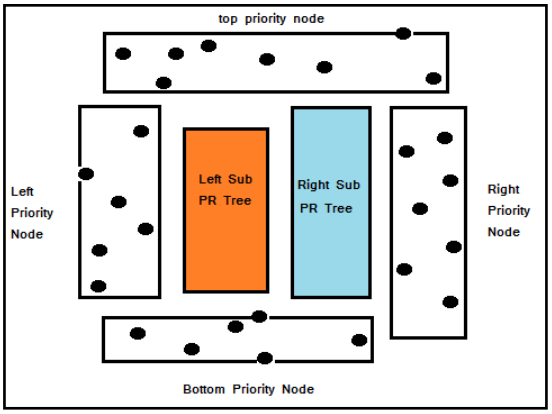

Fig 6: Bounding Rectangles in PR Tree for xy dimension query

CONCLUSION

The R-tree, projected by Guttman, has foundation for all the forthcoming variations of dynamic R-tree structures. The R*tree follow an engineering approach and evaluated various factors focussing on the performance of the R-tree. For this grounds, it is well thought-out the most strong variant and has found several applications, in together research and commercial applications. However, it is significant to mention that the PR-tree is the first approach that offers assured worst-case performance and overcomes the degenerated cases when approximately the entire tree has to be traversed. Hence, although it is more complex algorithm, it has to be well thought-out the best variant reported till now.

REFERENCES

[1] Guttman A.: 'R-trees: A Dynamic Index Structure for Spatial Searching', Proc. ACM SIGMOD Int. Conf. on Management of Data, Boston, MA, 1984, pp. 47-57.

[2] Sellis T., Roussopoulos N., Faloutsos C.: 'The R+ Tree: A Dynamic Index for Multi-Dimensional Objects', Proc. 13th Int. Conf. on Very Large Databases, Brighton, England, 1987, pp 507-518.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In Proc. ACM SIGMOD Int. C'onf. on Management of Data, Atlantic City, 1990.

[4] L. Arge, M. de Berg, H. J. Haverkort, and K. Yi. The priority R-tree: A practically efficient and worst-case optimal R-tree In Proceedings of ACM Management of Data (SIGMOD), pages 347–358, 2004.

[5] J. T. Robinson. The K-D-B-tree: A search structure for large multidimensional dynamic indexes. In Proceedings of ACM Management of Data (SIGMOD), pages 10–18, 1981.

[6] I. Kamel and C. Faloutsos. Hilbert R-tree: An improved R-tree using fractals. In Proceedings of Very Large Data Bases (VLDB), pages 500–509, 1994.

[7] R. Bayer and E. M. McCreight, "Organization and maintenance of large ordered indexes," in *SIGFIDET Workshop*. ACM, 1970, pp. 107-141.