A New Node Splitting Algorithm for R-Tree

Chandan Garai\textsuperscript{1} and Ranjan Dasgupta\textsuperscript{2}

\textsuperscript{1} Student, Dept. of Computer Science & Engineering, National Institute of Technical Teachers’ Training And Research, Kolkata, West Bengal, India
\textsuperscript{2} Dept. of Computer Science & Engineering, National Institute of Technical Teachers’ Training And Research, Kolkata, West Bengal, India

Abstract – R-Tree being a multi-dimensional version of B-Tree is being used in various applications including geographic information systems, computer aided designing systems, spatial databases etc. Efficient searching of this tree is always a challenge to the research community and several methods have been proposed for this purpose. In this paper, we propose a new splitting algorithm to help efficient retrieval by reducing chances of overlap.

Keywords: Spatial Database, Multi-Dimensional Indexing, R-Tree, Node Splitting Algorithm.

1 Introduction

An R-Tree [2] [4] [7-8] is a height-balanced tree. The index records in its leaf nodes contain pointers to data objects. It is the multidimensional extension of B-tree. Nodes correspond to disk pages. The structure is designed in such a manner that a spatial search requires visiting only a small number of nodes. The index is dynamic in nature. Insertions and deletions can be intermixed with searches and no periodic reorganization is required.

2 Spatial Database and R-Tree

A spatial database [1] [5-6] [9] [10] consists of a collection of tuples representing spatial objects, and each tuple has a unique identifier which can be used to retrieve it. Leaf nodes in an R-Tree contain index record entries of the form

\[(I, \text{tuple identifier})\]

Where, tuple_identifier refers to a tuple in the database and I is an n-dimensional rectangle which is the minimum bounding rectangle (MBR) of the spatial object indexed. Non-leaf nodes contain entries of the form

\[(I, \text{child pointer})\]

Where, child_pointer is the address of a lower node in the R-Tree and I covers all internal minimum bounding rectangles (iMBR) in the lower node’s entries. Fig. 1 shows the index structure of R-Tree.

![Fig. 1 Index Structure of R-Tree](image)

Let, M is the maximum number of entries that will fit in one node and let \( m \leq \frac{M}{2} \) be a parameter specifying the minimum number of entries in a node. An R-Tree [2] satisfies the following properties

1. Every leaf node contains between m and M index records unless it is the root.
2. For each index record (I, tuple_identifier) in a leaf node, I is the smallest rectangle that spatially contains the n-dimensional data object represented by the indicated tuple.
3. Every non-leaf node has between m and M children unless it is the root.
4. For each entry (I, child_pointer) in a non-leaf node, I is the smallest rectangle that spatially contains the rectangles in the child node.
5. The root node has at least two children unless it is a leaf.
6. All leaves appear on the same level.

3 Node Splitting Algorithms

In R-Tree, when we need to insert a data to any node and the node is full, then, we need to split that node to make room for the newly inserted records. There have been several R-Tree node splitting algorithms developed since R-Tree was introduced. Among them, Guttman’s Quadratic-Cost Algorithm [2], the Optimal node splitting algorithm [11], the Linear node splitting algorithm [12] and Basic node splitting with bipartition [3] are most popular. One important area of concern is the overlapping area. As in region data types zero overlapping is not possible, so a good node splitting algorithm should confirm the minimum overlapping area among the minimum bounding rectangles.

3.1 Guttman’s node splitting algorithms

These algorithms were proposed when R-Tree was introduced for the first time. In that work there were three node splitting algorithms, namely the Exhaustive algorithm, A Quadratic-Cost Algorithm and A Linear-Cost Algorithm.

3.1.1 Exhaustive algorithm

In this algorithm the author finds the most straightforward way to find the minimum area node split is to generate all possible groupings and choose the best grouping. However the number of possibilities is
approximately $2^{M-1}$ for $M$ number of MBRs. So, the number of possible splits is very large.

### 3.1.2 Quadratic-Cost algorithm

The mostly used one among the three algorithms is the quadratic method. This node splitting algorithm first picks two records that may cause the worst split if put into the same node. These two records are used as seeds and the algorithm repetitively finds a record that may affect the splitting quality the most and assigns it to the appropriate node until all records are assigned. If there are just enough records left unassigned that can make one of the two newly generated nodes to satisfy the lower bound of the number of records, then, the rest of records will be assigned to that node directly.

### 3.1.3 Linear-Cost algorithm

This algorithm is identical to Quadratic Split but uses a different way to choose the seeds. It finds the iMBRs which have highest low side and the lowest high side along any dimension. According to that it finds the seeds for the splitting. The rest of the iMBRs are simply chosen at random.

### 3.3 The optimal node splitting algorithm

Here two algorithms were proposed. The first one is a basic node splitting algorithm which partitions a full node into two. It is like the exhaustive method but has a time complexity of $O(Nd)$ instead of $O(2N)$, where $d$ is the dimension of data. The second algorithm is called SHIFT method. It first executes a pre-processing to get complementary MBRs with respect to all records. Then it tests each possible bipartition which is $O(Nd)$, and find the one with the best metric. Here area and perimeter are two metrics used to gain a high occupancy of disk pages.

### 3.4 A linear node splitting algorithm

The algorithm partitions the records into two groups along each axis according to the distance between a record’s MBR and the minimum and maximum coordinates on the axis. It then chooses a split axis by examining the number of records in each group. Then the partition along the selected axis will be the final splitting result. The algorithm’s time complexity is $O(N)$, where $N$ is the number of records in a node. The algorithm does not consider the overlapping between the nodes.

### 3.5 Basic node splitting with bipartition

This algorithm first decides the splitting axis using some parameter. Taking the ratio of the two axis it decides the splitting dimension and if not possible then counting the maximum numbers of iMBR stretched along a particular axis. As this algorithm is not concerned about the internal organization of the iMBRs therefore, sometimes splitting from other dimension can give better result and less overlapping.

### 4 Scope of Work

There are three parameters for R-Tree node splitting. Area is one of them. Smaller node’s area gives lower probability of the node being accessed. Some of the algorithms [2-3] [11-13] which are already available are only concerned about area increasing while choosing a leaf for new data and splitting a full node. Another parameter is the perimeter. Perimeter is responsible for the shape of the node’s MBR. For a particular area, smaller perimeter conforms to more square shape of the MBR.

Overlap area is the third area of concern. Overlap between nodes causes multiple nodes to be accessed when a query object falls into that particular region. So it is one important parameter and needs to be dealt properly while designing R-Tree constructing algorithms. The algorithms which are already available [2] [11-13] are less concerned about the overlapping area during the time to decide the splitting axis. As attaining zero overlapping is practically impossible for region data types, so, a new node splitting algorithm, which can check the overlapping before deciding the splitting axis is proposed here. Two case study shows that, it can give a satisfactory performance by ensuring to minimize the total overlapping area.

### 5 Our Proposed Node Splitting Algorithms

Here in the algorithm, length-of-X is the length of the MBR along the X axis and length-of-Y is the length of the MBR along the Y axis. X-length-count is the summation of the lengths of all the iMBRs along the X axis and Y-length-count is the summation of the lengths of all the iMBRs along the Y axis. TOR-of-X is the total overlapping region if splitted along the X axis and TOR-of-Y is the total overlapping region if splitted along Y axis. If a node in a R-Tree can contain M number of data then the lower bound(m) is ⌈$m/2$⌉. Overflow MBR is the MBR which contains more number of iMBRs than M.

**Algorithm SplitNode:**

**Input:** iMBR of a data  
**Output:** two new nodes

**Step 1:** The algorithm ‘SplitNode’ first invokes ‘FindSplitDim’ to decide the splitting dimension for that particular node.

**Step 2:** When the splitting axis is determined, then, the algorithm divides the MBR in the middle of the axis.

**Step 3:** Now for each iMBR it checks which iMBR is
fully contained by which MBR. Then it assigns that iMBR to that new node.

Step 4: If any iMBR is not fully contained by any of the MBR alone then fitting it into which MBR gives less area increase gets the priority.

Step 5: Breaks ties by assigning it to the one which will produce less overlapping area.

Step 6: Now the two new nodes are checked for the lowerbound condition. If passes then these nodes are the final nodes.

Step 7: If any MBR fails then the splitting dimension is changed and the same procedure is applied again.

Step 8: If it is also not able to produce satisfactory result then for the underflowing MBR, the nearest iMBR, which produce less area increase, is inserted.

Step 9: Step 8 continues until the underflowing MBR gets sufficient iMBR to overcome the lowerbound criteria.

Algorithm FindSplitDim:

Input: Over flown MBR
Output: Splitting axis

Step 1: ‘FindSplitDim’ first checks the length of the MBR in X & Y dimension and takes the longest axis as the initial splitting axis.

Step 2: If it is an square MBR then it checks the summation of the iMBRs in X & Y dimension. If the X-length-count is greater than Y-length-count then it takes Y as the initial splitting axis and vice-versa.

Step 3: If these two are equal then it arbitrarily chooses the initial splitting axis.

Step 4: Now it checks which iMBRs are left aligned. There must be at least one.

Step 5: After getting the iMBR then sets the value of the variable Z as the value of x2 of that iMBR.

Step 6: Now the algorithm finds the iMBR whose smaller value of x axis is lesser than variable Z and the bigger value x of that iMBR is greater than that of Z.

Step 7: If there is such an iMBR then the value of Z gets updated by the value of x2.

Step 8: When there is no such iMBR which has the same criteria and Z is equal to the bigger value of the minimum bounding rectangle then non-overlapping is not possible for that MBR along the X axis.

Step 9: So the algorithm checks the Y axis in the same manner. If a non-overlapping region is found then the final value of Z must be lesser than the bigger value of that axis.

Step 10: When non-overlapping split is not possible along any of the two dimensions then the algorithm invokes the ‘FindTotalOverlappingRegion’.

Step 11: After getting the values of the two variables (TOR-of-X & TOR-of-Y), now the ‘FindSplitDim’ algorithm decides the final splitting dimension.

Step 12: Dimension with less overlapping area is the key criteria for selecting the splitting dimension. If a tie occurs then it arbitrarily decides the splitting axis and returns the value.

Algorithm FindTotalOverlappingRegion:

Input: Over flown MBR whose non overlapping splitting is not possible
Output: TOR-of-X & TOR-of-Y

Step 1: A new variable P is initialised with the length of that particular axis.

Step 2: The algorithm starts in the same manner as FindSplitDim and when there is an iMBR which passes that particular criteria then the algorithm finds the minimum length of small value of that iMBR along X axis to Z and Z to the larger value of that iMBR along X axis.

Step 3: If this minimum value is less than the present value of P then P gets updated with it.

Step 4: When Z = X2 then the value of P is taken into account and multiplied with the length of the MBR along the other axis and value is stored in TOR-of-X variable.

Step 5: In the same way the TOR-of-Y is also calculated and the values are returned.

6 Step-by-step Operations

We have taken two examples to check the step-by-step operations of our algorithm and compare the output of our node splitting algorithm with basic node splitting with bipartition. Here, 4 numbers of data can be put into one node at maximum i.e. an MBR can contain maximum 4 numbers of iMBRs. The MBRs which have more than 4 numbers of iMBRs are taken as over flown MBRs.

Case study I:

In Fig. 2.a we have taken an over flown MBR. For the MBR X1 = 0, X2 = 36, Y1 = 0, Y2 = 21. There are five iMBRs, r1, r2, r3, r4, r5. Table 1 shows the specifications of these iMBRs
Table 1: Specifications of the iMBRs

<table>
<thead>
<tr>
<th>iMBR</th>
<th>x1</th>
<th>x2</th>
<th>y1</th>
<th>y2</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>0</td>
<td>12</td>
<td>11</td>
<td>21</td>
</tr>
<tr>
<td>r2</td>
<td>24</td>
<td>36</td>
<td>11</td>
<td>21</td>
</tr>
<tr>
<td>r3</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>r4</td>
<td>30</td>
<td>36</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>r5</td>
<td>9</td>
<td>27</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

Step 1: The algorithm first checks for the length of the MBR along X & Y dimension.

Step 2: The larger one decides the initial splitting axis. Here X is the initial splitting axis.

Step 3: Now the algorithm checks whether in X dimension, there is any overlapping or not.

Step 4: For that it first checks which iMBR has x1 equals to X1. There must be at least one iMBR satisfying this criterion. Here is r1 & r3. Our algorithm takes r1 arbitrarily.

Step 5: Then for that iMBR (r1) it updates the value of Z variable with the value of x2 (here it becomes Z = 12).

Step 6: Now the algorithm continues with for loop and checks which iMBR has (x1 < Z) & (x2 > Z). Here it finds that r5 satisfies that criterion. So the Z now becomes 27.

Step 7: Again in next step it gets r2 and the Z becomes 36. Now (Z = X2). So, in X dimension non-overlapping splitting is not possible.

Step 8: Now the algorithm changes the dimension and does the same. Here it finds that in Y dimension non-overlapping splitting is possible.

Step 9: It takes Y as the final splitting dimension and starts to split.

Step 10: The splitting is done in the middle of the splitting axis and the iMBRs which are fully contained by any of the two MBRs are assigned to them. Here r1 & r2 are fully contained by R1 and r3, r5, r4 are fully contained by R2. So, these are assigned accordingly.

Fig. 2.c shows the result of splitting using our proposed algorithm. The corresponding R-Tree is shown in Fig. 2.e.

Fig. 2.b shows the result of splitting using the algorithm basic node splitting using bipartition. This algorithm first check whether the given rectangle is j-long or not; i.e. the ratio between the length of X & Y axis is greater than Θ or not (Where Θ > 0). If it is a square MBR then it checks whether the number of iMBRs whose length along X axis is greater than the same of Y axis. Thus it decides the splitting axis and assigns the iMBR accordingly. There is no way to check whether the splitting generates a huge overlapping area. Here the splitting produces an overlapping area whether non-overlapping splitting is possible. Fig. 2.d shows the R-Tree constructed form the result of that splitting.

Case study II: Here we have taken another example. In Fig. 3.a we have taken an over flown MBR. For the MBR X1 = 0, X2 = 36, Y1 = 0, Y2 = 21. There are five iMBRs, r1, r2, r3, r4, r5. Table 1 shows the specifications of these iMBRs.
Table 2: Specifications of the iMBRs

<table>
<thead>
<tr>
<th>iMBR</th>
<th>x1</th>
<th>x2</th>
<th>y1</th>
<th>y2</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>0</td>
<td>18</td>
<td>11</td>
<td>21</td>
</tr>
<tr>
<td>r2</td>
<td>19</td>
<td>36</td>
<td>15</td>
<td>21</td>
</tr>
<tr>
<td>r3</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>r4</td>
<td>30</td>
<td>36</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>r5</td>
<td>10</td>
<td>28</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

Step 1: The algorithm starts by finding the iMBRs which have (x1 = X1). Here, r1 & r3.

Step 2: It continues by updating the value of Z after satisfying the criterion of (x1 < Z) & (x2 > Z).

Step 3: Finally for X dimension Z becomes X2 i.e. non-overlapping splitting is not possible in the X dimension.

Step 4: In the same way it checks in Y dimension and finds that in this case also non-overlapping splitting is not possible.

Step 5: Now, ‘FindTotalOverlappingRegion’ is invoked.

Step 6: As the same way it first checks for the iMBR having (x1 = X1). When it finds one, it updates the value of Z.

Step 7: Now here a new variable P is introduced and initialized with the length of the MBR along that axis.

Step 8: For rest of the iMBRs it checks if, (x1 < Z) & (x2 > Z). Now it finds the minimum of (Z-x1) & (x2-Z).

Step 9: If P > MIN{(Z-x1), (x2-Z)} then it updates the value of P with MIN{(Z-x1), (x2-Z)}. Finally P becomes 8 for X dimension.

Step 10: Finally when (Z = X2) then it multiplies P with the length along the other axis and stores the area in a variable as the total overlapping region along X axis. TOR-of-X is now (8 * 21) i.e. 168 units.

Step 11: In the same way it finds the total overlapping area for Y axis and returns the values. TOR-of-Y = 4 * 36 = 144 units.

Step 12: Then algorithm ‘FindSplitDim’ compares these values and decides the final splitting axis and returns the splitting dimension.

Step 13: Now the algorithm ‘SplitNode’ splits the node in Y dimension.

Step 14: r2 is assigned to R1 and r3 & r5 are assigned to R2 as they are fully contained by these MBRs.

Step 15: Now, r1 is assigned to R1 and r4 is assigned to R2 as they give less increase in the area.

Here in Fig. 3.b we can see the total overlapping region for X axis. It is also the result of the node splitting by the basic node splitting with bipartition. Fig. 3.d shows the R-Tree constructed by basic node splitting with bipartition.

The algorithm decides Y as the final splitting dimension for this MBR and Fig. 3.c shows total overlapping region along Y axis. It is the result of the node splitting using our proposed node splitting algorithm which significantly have less overlapping area. Fig. 3.e shows the R-Tree constructed by our proposed node splitting algorithm.

After splitting the MBR using our proposed algorithm, if any underflow condition occurs then the...
algorithm changes its splitting dimension and splits again. If it can’t satisfy the result then the under flown node includes the iMBR from the other which calls for a less area increase. This process continues until both the nodes passes the underflow condition.

7 Conclusions

In this paper, we have proposed a new algorithm to identify the best possible splitting axis to minimize overlapping. Minimization of overlapping in turn yields better result of searching. It goes without say, that in some cases, overlapping cannot be avoided, however our algorithm ensures minimum overlapping in such cases. We also compared our output with basic node splitting with bipartition and it has been shown that our splitting algorithm produces better result.

8 References


[10] Gang Liu, Qing Zhu, Zhenwen He, Yeting Zhang, Chonglong Wu, Xiaoming Li, Zhengping Weng, “3D GIS Database Model for Efficient Management of Large Scale Underground Spatial Data”.

