New Linear Node Splitting Algorithm for *R*-trees

C.H. Ang and T.C. Tan

Department of Information Systems & Computer Science
National University of Singapore
Republic of Singapore, 119260
E-mail: angch@iscs.nus.sg, tantc@iscs.nus.sg

Abstract. A new linear-time node splitting algorithm for R-trees is proposed. Compared with the node splitting algorithm that requires quadratic time and is used in most implementations of R-tree, it is more superior in terms of the time required to split a node, the distribution of data after splitting, as well as the area of overlapping. Most important of all, it has a better query performance. The claim is substantiated by an analysis of the algorithm and a set of empirical results.

1 Introduction

The volume of data to be handled by many information systems is ever increasing. In order to process the data efficiently, data collected are organized into huge disk files that can be accessed quickly with the use of appropriate index structures. The B-tree is commonly used when the data to be manipulated are points from one dimensional space while octree [5], multi-priority tree [8] and R-tree [3] can be used for higher dimensional space.

The R-tree, a data structure that is regarded as the multi-dimensional version of the B-tree, is known for its ease of implementation, its space efficiency, and its adaptability to higher dimensionality. It has a structure that is quite similar to that of the B-tree. It usually uses a separate file to organize all the indexes to the objects that are being stored. The objects can be retrieved via their identifiers (ids) in the index file. The indexes are being stored in disk blocks termed *nodes*. Information stored includes the object ids, the minimum bounding rectangles (*MBR*s) of the objects or groups of objects, and the pointers that put all nodes in an R-tree into place. There are many entries in an R-tree node. Each entry of a leaf node is of the form (R, P) where R is the MBR of the object with id P. For a non-leaf node, R is the MBR that encloses all the objects that can be reached by following the node pointer P.

To perform an insertion in an R-tree, a new object is inserted into the subtree whose MBR needs the least amount of enlargement. When a node N is already filled to its capacity M, an inclusion of a new object will cause an overflow and hence the node has to be split. A new node is then allocated and some of the data in the overflowed node are moved over to this new node. The MBR of N

is updated and a new index is inserted into the index node which contains N's index. Different ways of splitting a node may be used to achieve different goals, such as the time efficiency in the R-tree creation, the minimal overlapping of the MBRs, or the minimal total coverage of the MBRs.

Since the MBRs in an R-tree may overlap, a query rectangle that intersects with any of these MBRs requires the traversal of the related subtrees. As such, using R-tree to answer window queries may be slow. Various ways to improve R-tree's performance have been proposed; the R^+-tree [7, 6], the R^*-tree [1], and the Hilbert R-tree [4] are just some of them. These variants of the R-tree either avoid overlapping altogether by splitting the objects into many parts, or pack the objects in different ways. Where node splitting is concerned, the variants of R-tree are more or less using the same algorithm.

In this paper, we propose a new linear time node splitting algorithm that outperforms the quadratic algorithm. The paper is organized as follows. Section 2 reviews the available node splitting algorithms and Section 3 describes the new algorithm. We then apply our algorithm to some examples taken from other papers for a quick comparison. In addition, several experiments are conducted, the performance are assessed based on several criteria, and the results are reported and analyzed in Section 4. Section 5 is the conclusion.

2 Some Existing Algorithms

Each node in an R-tree has a capacity M and, except for the root, has to be filled with no less than m entries. The value of m is usually between $\frac{1}{3}$ to $\frac{1}{2}$ of M. In his seminal paper [3] on R-tree, Guttman proposed several ways to split a node: the exhaustive method, the quadratic algorithm, and the linear algorithm. All methods proposed attempt to minimize the total area covered by the MBRs of the 2 nodes resulted from splitting and they are briefly described below:

1. *Exhaustive method:*
   It tries all possible ways in splitting the node in order to find the optimal solution. Due to the exponential time required, this method is impractical and will be excluded from our discussion henceforth.
2. *Quadratic algorithm:*
   It consists of two phases. In phase 1, a pair of objects is selected to be the seeds with one in each of the two nodes (groups) resulted from the splitting. These are the two objects that would have produced the largest white space if they were to be placed in the same node.

   In phase 2, for each of the remaining objects, calculate the difference, upon the inclusion of that object into each of the two groups, between the increases in the areas of the MBRs of the two groups. The object that causes the largest difference will be chosen and assigned to the group with the smaller MBR area expansion. We say that the object is 'nearer' to this selected group.

Thus in each iteration of phase 2, only one object is being assigned to the group of objects that is 'nearer'. At the end of each iteration, if all the remaining unassigned objects can be included into the smaller node such that the node will then contain exactly m elements, then this is done so. This is to make sure that each node will contain the minimum number of entries. With such bulk assignment of objects, it is obvious that the goal of minimizing the total coverage may be compromised.

3. *Linear algorithm:*
   It consists also of two phases. In phase 1, the two objects that are separated farthest apart along the direction of certain dimension are chosen as the seeds. The remaining objects will subsequently be assigned one by one to the group that is nearer, in the same sense as described above. This algorithm also maintains the minimum occupancy of each node.

According to the empirical results obtained by Guttman, it seems that the performance of the linear algorithm is not too far off from that of the quadratic algorithm in terms of the space requirement of the resulting R-trees as well as the query performance using them, whereas in [1] the quadratic algorithm is shown to perform much better than the linear one.

Let us look at some examples to see how the above algorithms work. In Figure 1, the overflowed node containing rectangles R1 to R4 is to be split into 2 nodes. Using the linear algorithm, R1 and R2 are chosen as seeds since their separation in the x direction is the largest. Since R3 is nearer to R2 and R4 is nearer to R1, the resulting two nodes contain {R1, R4} and {R2, R3}. The MBRs of these two nodes overlap. If the overflowed node is split into {R1, R3} and {R2, R4}, then there will be no overlap.

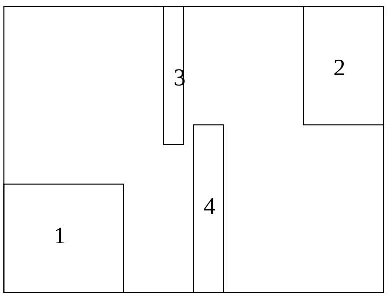


Fig. 1. Overflowed node with 4 rectangles.

In Figure 2, the node containing 5 rectangles is to be split. Using the quadratic algorithm, R1 and R2 are chosen as the seeds since they would have created the largest white space if they were to be placed into the same node. Since R5 is nearer to R1, R3 is nearer to {R1, R5}, and R4 is nearer to R2, the node is split into two nodes {R1, R3, R5} and {R2, R4} which again have their MBRs overlapping each other. If the overflowed node is split into {R1, R5} and {R2, R3, R4}, or {R3, R5} and {R1, R2, R4}, then there will be no overlap.

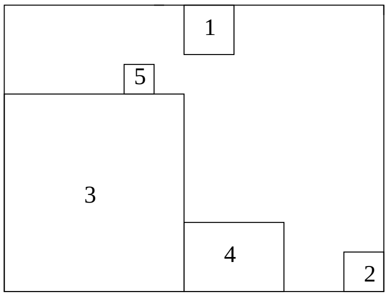


Fig. 2. Overflowed node with 5 rectangles.

It has been pointed out in [1] that inserting a new object into a node which needs the least expansion of its MBR is biased towards the node that contains more rectangles. It is generally true that the more rectangles a node contains, the larger its MBR is, and hence the least enlargement is needed to include a new rectangle. As a result, a large node tends to get larger. Therefore, during node splitting, the first few assignments made in redistributing the rectangles have great influence on the distribution of the remaining rectangles. The uneven distribution of the rectangles resulted from the application of the node splitting algorithm proposed by Guttman is apparent from the empirical results shown in section 4.

3 New Linear Algorithm

In order to optimize the search performance of the R-tree, we should aim to minimize the overlapping areas of the bounding rectangles involved. This will reduce the chance of the query rectangle intersecting with an overlapped area which requires several search paths to be followed.

In order to minimize the overlap between the two new nodes produced when a node splits, all rectangles are pushed as far apart as possible towards the boundary of the MBR of the overflowed node. For ease of discussion, let us first consider only rectangles in two dimensional space. Each rectangle S can be represented by a 4-tuple (xl, yl, xh, yh) where (xl, yl) and (xh, yh) are the coordinates of the bottom-left corner and the top-right corner of S respectively.

Let the MBR of the overflowed node N be R_N=(L, B, R, T). We use four lists $LIST_L$, $LIST_B$, $LIST_R$, and $LIST_T$ to remember those rectangles that are nearer to a border than to its opposite. These lists are the main structures used in the New Linear Algorithm which is described as follows:

Algorithm NewLinear

$LIST_L \leftarrow LIST_R \leftarrow LIST_B \leftarrow LIST_T \leftarrow \emptyset$
For each rectangle $S = (xl, yl, xh, yh)$ in the overflowed node N with R_N=(L, B, R, T),
  If $xl - L < R - xh$
    then $LIST_L \leftarrow LIST_L \cup S$
    else $LIST_R \leftarrow LIST_R \cup S$
  If $yl - B < T - yh$
    then $LIST_B \leftarrow LIST_B \cup S$
    else $LIST_T \leftarrow LIST_T \cup S$
If $\max(|LIST_L|, |LIST_R|) < \max(|LIST_B|, |LIST_T|)$
  then split the node along the x direction
else If $\max(|LIST_L|, |LIST_R|) > \max(|LIST_B|, |LIST_T|)$
  then split the node along the y direction
else /* tie breaker */
  If overlap$(LIST_L, LIST_R) <$ overlap$(LIST_B, LIST_T)$
    then split the node along the x direction
  else If overlap$(LIST_L, LIST_R) >$ overlap$(LIST_B, LIST_T)$
    then split the node along the y direction
  else split the node along the direction with smallest total coverage

Note that each rectangle S appears once in either $LIST_L$ or $LIST_R$ depending on whether it is horizontally closer to the left border or the right border of R_N, and once in either $LIST_B$ or $LIST_T$ depending on whether it is vertically closer to the bottom border or the top border of R_N. The split will be carried out horizontally or vertically depending on the evenness of the distribution. The direction that ensures a more even distribution will be used to split the node. In the event of a tie, the least overlap is favoured, and if again there is a tie, the one with the minimum coverage is chosen.

If there are n rectangles in the overflowed node, then it requires $O(n)$ to scan the list of rectangles once in order to create the four lists. Note that $|LIST_L| + |LIST_R| + |LIST_B| + |LIST_T| = 2n$. Once the direction of splitting the node is determined, the two lists related are the contents of the two nodes resulted from the split.

It is quite clear that this algorithm does not require any seed to grow the two new nodes. Moreover, it can be easily extended to handle the node splitting for data taken from k-dimensional space. What is needed is to have two lists to keep track of those objects that are closer to one of the two borders in each dimension. The given list of rectangles to be redistributed is still scanned once. The direction chosen to split the overflowed node is the one that ensures a more even data distribution. Since there are kn entries in the $2k$ lists, the time complexity of the algorithm is $O(kn)$, or simply $O(n)$ as k is usually a small value.

In the list of rectangles to be redistributed, if there are a few rectangles that are outliers while others form a cluster, then the redistribution will be lopsided. In this case, the outliers can be reinserted so that they may be absorbed by some of the surrounding nodes. If the outliers keep falling back to the same overflowed node, then they are to reside in a separate node which, of course, will have a very low utilization. This is really a rare case and we would like to report that such situation has not been encountered so far in our experiments.

4 Comparisons and Empirical Results

To have a quick comparison between the performances of various node splitting algorithms, let us visit the previous examples again. Referring to Figure 1, we have $LIST_L$={R1, R3} and $LIST_R$={R2, R4} in the x direction, and $LIST_B$={R1, R4} and $LIST_T$={R2, R3} in the y direction. Since both lists contain the same number of rectangles and only the split in the x direction does not produce two nodes that overlap, so $LIST_L$ and $LIST_R$ are used and the overflowed node is split in the x direction.

As for Figure 2, we have {R1, R5} and {R2, R3, R4} in the y direction, and {R3, R5} and {R1, R2, R4} in the x direction. There is no overlap in either way of splitting, so either one can be used. Since the split in the y direction will produce a smaller total coverage, it is preferred and chosen.

In addition, two examples have been used in [1] to show the deficiency of the quadratic algorithm and the variant proposed by Greene in [2]. They are reproduced in Figures 3 and 4 for comparison. In both examples, the new linear algorithm produces better or at least equally good results.

In Figure 3b, the quadratic node splitting algorithm is used with the restriction that at least 30% of the space in a node must be utilized, ie, $m = 0.3M$. Although there is no overlapping of the resulting nodes, the distribution of rectangles is not even. The split as carried out by R^*-tree produced the same result as shown in Figure 3e. When m is increased to $0.4M$, although the distribution is now more even as shown in Figure 3c, the nodes overlap significantly. On the other hand, Greene's method is able to split the node more evenly with smaller overlap. Yet none of these methods can produce a better result than that of the new linear algorithm which produces two nodes containing the same number of rectangles with no overlapping.

In Figure 4b, Greene's split produces two nodes which overlap and are unbalanced. The new linear algorithm is able to produce the same result as that of R^*-tree which uses a more sophisticated and time consuming node splitting method.

Besides having a lower time complexity, the new algorithm produces a more even distribution of data when a node is split and the overlap between the MBRs bounding the two nodes is also smaller. To demonstrate its strength, a series of experiments has been conducted to compare the performance of the

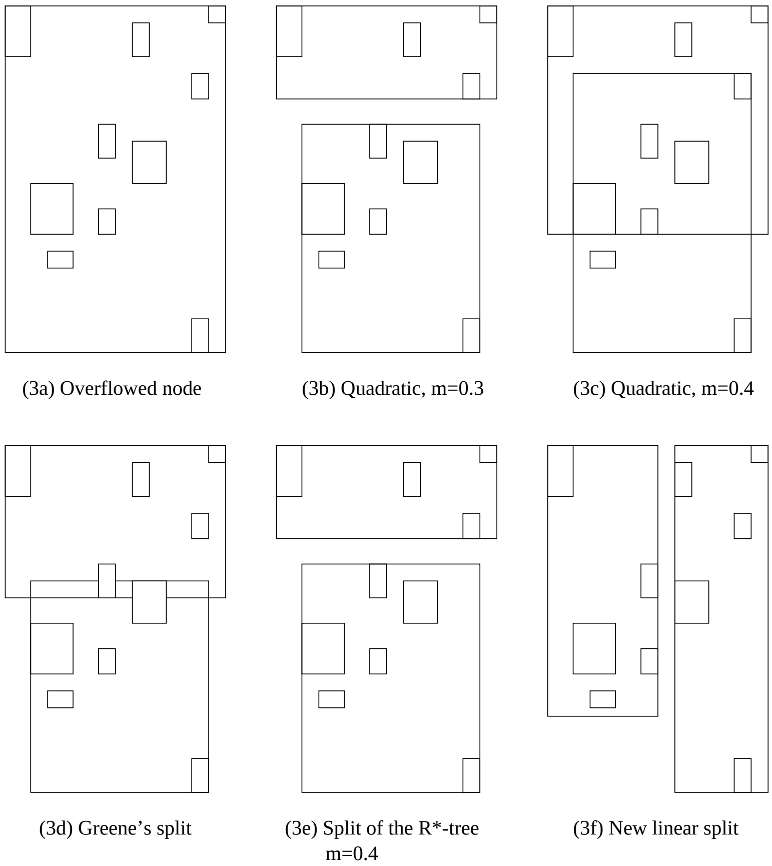
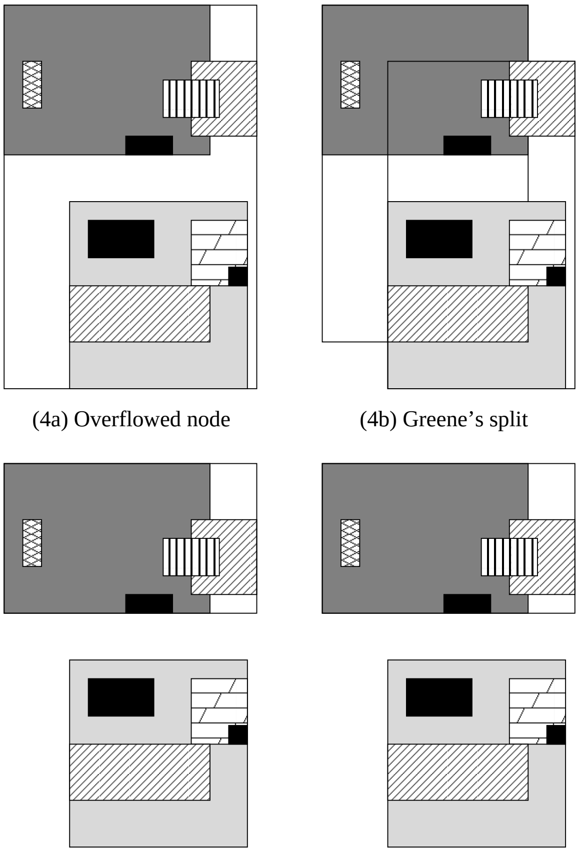


Fig. 3. Comparisons with other algorithms.

linear, the new linear, and the quadratic algorithms. We do not include those splitting algorithms used in the variants of R-tree in our comparisons because they are basically the quadratic algorithm employing different criteria in phase 2 to distribute the rectangles.

We note that the time needed to create an R-tree is mainly determined by the I/O activity involved and, to a certain extent, by the efficiency of the node splitting algorithm used to grow the tree. A node splitting algorithm that fails to ensure an even data distribution will usually produce a tree with more nodes that are either quite full or less than half-filled. This may cause more node splittings subsequently and thus more I/O's and time will be incurred.

The programs used in the experiments are all written in C and are run on Digital AlphaServer 8400 5/350 with 4 processors. The first experiment is to measure the time taken in creating R-trees when different splitting algorithms are used. Five R-trees are built by inserting n rectangles that are randomly



(4a) Overflowed node (4b) Greene's split

(4c) Split of the R*-tree (4d) New linear split

Fig. 4. Comparisons with other algorithms.

generated, with n ranging from 10,000 to 100,000 in steps of 10,000, and the average time is recorded. Table 1 shows the results of this experiment.

Note that the splitting algorithm is called only when a node overflows, an ocassion that arises only once after many insertions. Yet as can be seen from Table 1, the new linear algorithm manages to reduce the R-tree creation time by more than 40%. This is really a very significant improvement. For small R-trees, the creation times are reduced by more than 50%. As the size of the R-tree increases, its depth also increases and more I/O's are required to insert a rectangle. As a result, the saving in the time used in splitting the overflowed nodes becomes less dominant.

Since node splitting will invariably produce two nodes of different sizes most of the time, we would like to define some metrics to measure the quality of the split. Let us call the node with fewer rectangles $L1$ and the one with more rectangles $L2$. We are interested in the percentage of the space that is covered by $L1$ and $L2$ within the MBR of the overflowed node, the percentage of the overlap between them, and the number of rectangles in each of them. The statistics are shown in Table 2.

Number of Rectangles	Quadratic	New Linear
10,000	8.88	4.98
20,000	16.25	7.60
30,000	23.78	11.38
40,000	31.82	16.17
50,000	42.20	22.02
60,000	51.45	31.13
70,000	61.95	33.97
80,000	70.16	39.53
90,000	77.61	40.18
100,000	90.11	49.43

Table 1. *R*-tree creation time (sec.)

	Quadratic,m=0.33M	Quadratic,m=0.5M	New Linear	Linear
L1's Coverage (%)	49.20	50.05	48.07	61.53
L2's Coverage (%)	55.74	58.84	49.76	64.03
Overlap (%)	7.37	10.91	0.36	27.73
Number of rectangles in L1	47.18	50.00	47.18	43.56
Number of rectangles in L2	52.82	50.00	52.82	56.44

Table 2. Rectangle distribution and overlap

From Table 2, it is noted that the original linear algorithm, even though it is simple to implement and fast to execute, does not produce satisfactory results. The nodes cover too big an area, overlap too much, and the rectangles are not evenly distributed.

Compared with quadratic algorithm with $m = 0.33M$ and $m = 0.5M$, the new linear algorithm always produce nodes that cover area of almost equal size, with minimum total coverage, minimum overlap and the rectangles are evenly distributed. Note that even though the quadratic algorithm is able to produce nodes of equal size when $m = 0.5M$, this is achieved at the expense of the extent of overlap and the total coverage.

The last experiment is designed to assess the quality of the resulting *R*-trees in terms of query performance. An *R*-tree could be created in shorter time with the use of the new linear node splitting algorithm, but what is more important is whether the *R*-tree created is efficient for query processing.

There are point query, intersection query, containment query, etc. A typical query may ask for all rectangles that intersect with a given query rectangle to be retrieved. Since a point query is easy to satisfy and containment query is not used frequently, we therefore focus on the performance of intersection query.

In the last experiment, 100 rectangles are generated randomly and used to perform intersection queries on the two R-trees which contain 100,000 rectangles and are created using the quadratic algorithm and the new linear algorithm respectively. The number of node accesses is then recorded. The same experiment is repeated 10 times and the data collected can be found in Table 3. It is noted that on the average, the number of node accesses has been reduced by 16%. This is again a significant improvement.

Test	Quadratic	New Linear
1	833	743
2	949	731
3	960	768
4	1028	820
5	922	868
6	912	752
7	996	821
8	892	834
9	964	796
10	965	804
11	951	804

Table 3. Number of node accesses for 100 intersection queries

5 Conclusion

In this paper, we have described a simple linear time algorithm to split an overflowed node in an R-tree. It is shown in many aspects to be more efficient than the quadratic algorithm that is currently in use. With the use of the new node splitting algorithm, the overlap between the nodes resulted from the split is reduced significantly with no compromise in the evenness of data distribution.

Two experiments are carried out to find the running time required to create an R-tree using various node splitting algorithms as well as to gauge the quality of the splits. Since variants of R-tree are using basically the same quadratic node splitting algorithm proposed by Guttman with some modifications, the metric that is used to measure the quality of the splitting will not be much different from that of the R-tree, and their time complexity will not change asymptotically. As such, it is not necessary to involve them in the comparisons as far as the efficiency of the node splitting algorithm and the quality of the split are concerned. The third experiment shows that the R-tree created using the new linear node splitting algorithm is more superior in query processing as well.

Although the algorithm proposed is optimal, there are rooms for some minor improvements which are to be worked on in future. Here are some examples:

1. To break the tie between two lists of equal number of rectangles, in addition to opting for a node with smaller overlap, we may use other criteria such as having smaller total coverage or smaller extent in certain dimension.
2. We may choose the direction that minimizes the overlap as the splitting direction instead of one that minimizes the difference in the list sizes. We have tried this out and observed a further reduction in the overlap with no noticeable effect on the evenness of data distribution.
3. We may tinker with the assignments of the rectangles that cause the overlap so as to see whether the overlap can be reduced. These rectangles, instead of assigning them to the nearest border, may be assigned to the opposite if the overlap can be reduced.

Acknowledgement

We would like to thank Lee Wee Fong, Lee Sue Chin, and Lim Joo Pheng for their help in the implementation and Ooi Beng Chin for his valuable comments.

References

1. Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger: The R^*-tree: An efficient and robust access method for points and rectangles. Proc. of the SIGMOD Conference, Atlantic City, NJ. (May 1990) 322-331
2. D. Greene: An implementation and performance analysis of spatial data access methods. Proc. of the 5th International Conference on Data Engineering. (1989) 606-615
3. Antonin Guttman: R-tree: A Dynamic Index Structure for Spatial Searching. Proc. of the SIGMOD Conference, Boston. (June 1984) 47-57
4. Ibrahim Kamel and Christos Faloutsos: Hilbert R-tree: An improved R-tree Using Fractals. Proc. of the 20th VLDB Conference, Santiago, Chile. (May 1994) 500-508
5. Hanan Samet: The Design and Analysis of Spatial Data Structures. Addison-Wesley (1989)
6. T. Sellis, N. Roussopoulos, and C. Faloutsos: The R^+-tree: A dynamic index for multi-dimensional objects. Proc. of 13th VLDB Conference, England, (1987) 507-518. Also available as Computer Science TR 1795, University of Maryland, College Park, MD (1987)
7. M. Stonebraker, T. Sellis, and E. Hanson: An Analysis of rule indexing implementations in data base systems. Proc. of the First International Conference on Expert Database Systems, Charleston, SC. (April 1986) 353-364
8. Ching-Der Tung, Wen-Chi Hou, and Jiang-Hsing Chu: Multi-Priority Tree: An Index Structure for Spatial Data. Proc. of 1994 International Computer Symposium. (1994) 1285-1290