OMT: Overlap Minimizing Top-down Bulk Loading Algorithm for R-tree

Taewon Lee and Sukho Lee

School of Electrical Engineering and Computer Science,
Seoul National University, Republic of Korea

warrior@db.snu.ac.kr and shlee@cse.snu.ac.kr

Abstract. Efficient index construction in multidimensional data spaces is important for many database applications because both construction time and performance gains in query processing are important. To efficiently manage multidimensional data in scientific and data warehousing environments, R-tree based index structures have been widely used. In this paper, we propose a top-down bulk-loading algorithm for the R-tree called OMT that minimizes overlapped area between directory nodes to extremely improve the query performance. The main idea of our work is that the overlap between nodes in the top level of the tree is more critical than that in the leaf level. We first determine the topology of the resulting R-tree with a few calculation and then group the data items to create the entries of the root node. Since we create the tree from the top to bottom, we controls the overlapped area between the directory nodes. After that, we recursively partition each entry to create lower level nodes with following 100% space utilization constraint. This approach can improve the query performance of the resulting R-tree by decreasing unnecessary node visits. We present a detailed algorithm for OMT.

1 Introduction

R-trees are a common indexing technique for multi-dimensional data and are widely used in spatial and multi-dimensional databases. Usually, we build an R-tree by inserting one object at a time. However, this is a slow operation and it produces an R-tree with low space utilization and large overlap.

Bulk-loading methods are used to load huge amount of data set in relatively short time and to produce a well structured R-tree. It produces an R-tree with nearly 100% space utilization and improved query times (due to the fact that fewer nodes need to be accessed while performing a query).

Several bulk-loading methods were proposed and most of them attempted to minimize the overlapped area between nodes while achieving 100% space utilization [2, 3]. They first preprocess the data file of N data items so that the data items are ordered in $\lceil N/M \rceil$ consecutive groups of M items, where M is the maximum number of entries that can be placed in the same node. Then they recursively pack these nodes treating each node as a data item at the next level, until the root node is created.

Since they use a bottom-up approach, upper level nodes are created later than the leaf nodes. This usually leads to the large overlap between directory nodes and as a result, an R-tree suffers from poor performance in the query evaluation. This is explained in detail in Section 2.

In this paper, we propose a top-down bulk-loading algorithm that minimizes overlap between the directory nodes. Since we can determine the topology of the resulting R-tree by 100% space utilization constraint, we can create an R-tree efficiently from top to bottom. As a result, we can achieve the same space utilization and better query performance compared to the previous methods.

We present the motivation of our work and the detailed algorithm of *OMB* in the following section.

2 OMT: Overlap Minimizing Top-Down Bulk Loading Algorithm

In this section, we describe the motivation of our work and then our algorithm in detail. We discuss about the cost of R-tree construction in the last part of this section.

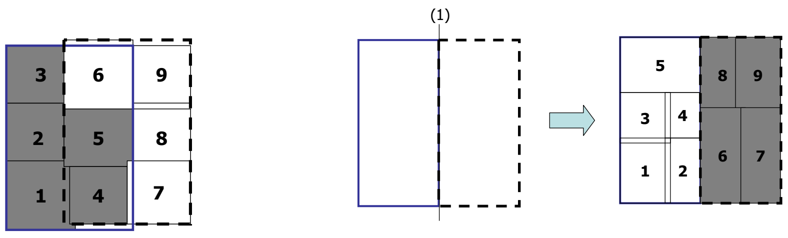
2.1 Motivation of our work

In STR method, which is the state of the art in bulk loading area, they "tile" the data space using $\sqrt{N/M}$ vertical slices so that each slice contains enough items to pack roughly $\sqrt{N/M}$ nodes [3]. To do so, they sort the data items by x-coordinate and partition them into $\sqrt{\lceil N/M \rceil}$ vertical slices. And then they sort the items of each slice by y-coordinate and pack the items into nodes by grouping them into runs of length M. As a result, all the leaf level nodes are generated. Directory nodes are created by treating each MBR of leaf nodes as a data item and recursively packing these MBRs into nodes at the next level, until the root node is created.

As we can see in Fig. 1 (a), MBR of directory nodes tend to have large overlap because of the 100% space utilization constraint. For example, if we assume the maximum number of entries($=M$) to be 5 in Fig. 1 (a), first directory node should cover 5 leaf nodes to satisfy the space utilization constraint. As a result, it is inevitable to have large overlap with another directory node.

To perform a query, we retrieve and examine each rectangle that intersects query region Q at every level [1]. It is possible to follow several paths from the root. In this reason, it is important to minimize the overlapped area from the top level of an R-tree. Reducing the possibility of several path lookup during the query processing radically in the top level of the tree is the most important thing if possible.

As in Fig. 1 (b), top-down approach can reduce overlap between directory nodes since it first creates the nodes from top to bottom.



(a) bottom-up bulk-loading(STR)　　(b) top-down bulk-loading(OMT)

Fig. 1. Motivation of our bulk-loading algorithm OMT. (M=5)

2.2 A detailed description of OMT

The first step of our algorithm is to determine the topology of the tree resulting from our bulk-loading. This is possible because of the 100% space utilization constraint. For this, we first decide the height of an R-tree by Eq. (1). From this, we can decide the number of data items in one subtree rooted by an entry of the root node. Since we will pack the data items with 100% space utilization, most of the nodes except a few will have M entries. From this, we can calculate the size of a subtree. This is given in Eq. (2).

$$h = \lceil \log_M N \rceil \tag{1}$$

$$N_{subtree} = M^{h-1} \tag{2}$$

Now, we can determine the number of entries of the root node and the number of vertical slices at the top level. This is given in Eq. (3). This also represents the number of nodes to be generated from one vertical slice except the last one. Note that the last vertical slice may result in more than S entries.

$$S = \left\lfloor \sqrt{\left\lceil \frac{N}{N_{subtree}} \right\rceil} \right\rfloor \tag{3}$$

Using Eq. (3), we can create entries of the root node by partitioning the data items into S groups. Each vertical slice is partitioned into S groups and each group makes up an entry of the root node.

Next, we partition each entry of the root into M groups. Each group again will be the entries in the lower level. We recursively partition each group in the lower level into M groups until each group contains only M data items.

2.3 Construction cost of our bulk-loading method

We assume the leaf level as level 1 and the root level as h. In our algorithm, nodes in each level except the root level are partitioned into M groups. To do this, we need to sort the data items by turns x and y-coordinate in each other level. As we started to sort them by x-coordinate, in level h-2, we sort the data items by y-coordinate and in level h-4, we sort them by x-coordinate and so on.

This may seem to cost large. However, it does not cost much compared to the STR method which sorts the entire data once by x-coordinate and sorts data items in each vertical slice by y-coordinate. Since we have not yet performed any experiments, we cannot exactly compare the construction cost with that of STR. By our analytical evaluation, it shows that our algorithm costs about 2 times more than that of STR to build up the R-tree. However, it is still extremely faster than the traditional one-by-one insertion method.

Although it is important for a bulk loading algorithm to build up an R-tree as fast as possible, the more important role of bulk loading is to build an R-tree that can perform queries efficiently. Our algorithm achieves the same space utilization with that of STR and also improves the query performance by greately reducing the overlap between the directory nodes.

3 Conclusion and Future Work

In this paper, we have proposed a top-down bulk loading algorithm to improve the query performance of an R-tree. Previously proposed bulk loading methods build up an R-tree from bottom-up. This inevitably causes high overlapped area between directory nodes. As a result, it makes unnecessary traversals during the search query operation which degrade the performance of an R-tree.

The key idea of our work is that the overlap between nodes in the upper level is more critical than that in the leaf level. Therefore, we devised an algorithm which creates the top level nodes first.

Since we have not yet performed any experiments, we need to prove our idea in the near future. We think that the query performance will be great enough to cancel out the defect of higher construction cost.

For the future work, we are going to develop improved huristics to effectively construct the R-tree.

References

1. Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM-SIGMOD Conference*, pages 47–57, June 1984.
2. Ibrahim Kamel and Christos Faloutsos. On packing R-trees. In *Proceedings of the second international conference on Information and knowledge management*, pages 490–499, 1993.
3. S. T. Leutenegger, J. M. Edgington, and M. A. Lopez. STR: A Simple and Efficient Algorithm for R-Tree Packing. In *Proceedings of the IEEE Data Engineering*, pages 497–506, 1997.