A Survey of R-Trees

Aaron Gorenstein
University of Wisconsin - Madison
agorenst@cs.wisc.edu

Rebecca Lam
University of Wisconsin - Madison
rjlam@cs.wisc.edu

March 11, 2013

1 Introduction

What was the motivation for this survey? What was the point of the original paper?

An overview of the rest of the paper here.

2 Overview of R-Trees

To solve the problem of performing efficient searchs on spatial data, Guttman proposed the R-tree, which inspired a variety of different variations analagous to the family of B-trees. In Section 2.1 we outline the original R-tree paper, and in Section 2.2 we examine the variants and draw appropriate comparisons.

2.1 R-Trees

In 1984, Guttman first proposed the idea of modifying the B-tree structure to use minimum bounding rectangles (MBR) as a way to restrict the search space during a lookup for spatial data. This data structure is called the R-tree. R-trees are structured similarly to B-trees except, instead of having separation values in each internal node that divide its subtrees, R-tree internal node entries correspond to MBRs that bound its descendents. For instance, the MBR of a particular node completely overlaps the MBRs of the nodes of its child and its child's children. ike in the B-tree case, nodes correspond to disk pages and leaves point to database objects.

R-trees are bound by two parameters m and M, the minimum and maximum number of entries for each node except the root, respectively. An internal node entry is of the form (mbr, p), where mbr is the MBR containing the MBRs of its descendents and p is the pointer to its child subtree. The mbr entry is of the form (I_0, I_1, ..., I_{n-1}), where n is the number of dimensions and I_i is of form $[a,b]$, a closed bounded interval along the i-th dimension. Similarly, a leaf node entry is of the form (mbr, oid), where mbr is the MBR containing the object, and oid is the identifier for the object in the database. Finally, the root node must have at least three entries except if it is a leaf.

There are multiple operations that are associated with an R-tree, which we discuss in the following sections.

2.1.1 Search

In order to find all entries overlapped by a bounding rectangle S in the R-tree, the pseudocode of Figure 1 is used. In a similar fashion to a B-tree traversal, each node in the tree starting from the root is checked for overlap using the mbr field in the entry. If there is overlap, the search descends into the subtree pointed to by p until it reaches a leaf. If the leaf entry's mbr overlaps with S, then the object ID oid is returned. Note that there is no worst-case performance guarantee for this algorithm.

2.1.2 Insert

Insertion again is similar to B-tree insertion methods, as illustrated by the algorithm in Figure 2. The algorithm traverses the tree to find the appropriate node to insert into and performs splits when inserting into full nodes, but there is one important distinction: node splitting heuristics. B-tree node splitting is simple since it is only necessary to partition the two resulting nodes into two equally

```
function SEARCH(T, S)                    ▷ Return all entries overlapped by S given an R-tree rooted at T
    if T is not a leaf then
        for all E in T do
            if E.mbr overlaps S then
                SEARCH(E.p, S)
            end if
        end for
    else
        for all E in T do
            if E.mbr overlaps S then return E.oid
            end if
        end for
    end if
end function
```

Figure 1: Pseudocode for searching a R-tree rooted at T given a search rectangle S

sized nodes. In R-trees, the goal typically is to have MBRs that cover the least amount of dead space, which is found by inserting entries into the node that would result in the least amount of enlargement. In the original R-tree paper Guttman discusses three different types of node splitting algorithms: linear, quadratic, and exponential.

2.1.3 Delete

Delete

2.2 R-Tree Variants

Much like its cousin, the B-tree, the R-tree has a few main variants such as the R^+-tree and the R^*-tree, which we discuss in the following sections.

```
function INSERT(T, E)                    ▷ Insert entry E into the R-tree rooted at T
    CHOOSELEAF(T, E)                     ▷ Traverse tree from T to appropriate leaf. At each level choose node L whose MBR will need the least enlargement to cover E.mbr or if there is a tie, choose node with minimum area. Return L
    if L is not full then
        Insert E into L
    else
        SPLITNODE(L)                     ▷ Returns L and LL containing E and the old entries of L
    end if
    ADJUSTTREE(L) ▷ Ascend from leaf node L up to the root T and propagate splits.
end function
```

Figure 2: Pseudocode for inserting into an R-tree rooted at T given an entry E

```
function DELETE(T, E)    ▷ Deletes entry E from the R-Tree rooted at T
end function
```

Figure 3: Pseudocode for deleting an entry E from an R-tree rooted at T

2.2.1 R+-Trees

2.2.2 R*-Trees

3 Implementation Challenges

4 Database Related Challenges

5 Modern Applications

cool topics like parallelism, concurrency, emerging applications

6 Conclusion