Spatial Index Structures
(R-tree Family)

Instructor: Cyrus Shahabi
Problem

• Given a collection of geometric objects (points, lines, polygons, ...)
• organize them on disk, to answer spatial queries (range, nn, etc)
Problem

• Spatial objects
  – Points, lines, rectangles, regions, …

• Hierarchical data structures
  – Based on recursive decomposition, similar to divide and conquer method, like B-tree.

• Why not B-Tree?
  – More than one dimension
  – Concept of closeness relies on all the dimensions of the spatial data
R-trees

- [Guttman 84] Main idea: extend B+-tree to multi-dimensional spaces!
  - (only deal with Minimum Bounding Rectangles - MBRs)
• Height-balanced tree similar to B-tree for $k$-dimensions
• Every leaf node contains between $m$ ($m \leq M/2$) and $M$ index records, unless it is the root
• For each index record $(I, \text{tuple-identifier})$ in a leaf node, $I$ is the MBR that contains the $n$-dimensional data object represented by the indicated tuple
• Every non-leaf node has between $m$ and $M$ children unless it is the root
• For each entry $(I, \text{child-pointer})$ in a non-leaf node, $I$ is the MBR that spatially contains the rectangles in the child node.
• All leaves appear on the same level
• The root node has at least two children unless it is a leaf
Example

- $m=2, M=4$: group nearby rectangles to parent MBRs; each group $\rightarrow$ disk page
Example

- $m=2$, $M=4$
Example

• $m=2$, $M=4$
R-trees - format of nodes

- \{(MBR; \text{obj\_ptr})\} for leaf nodes
R-trees - format of nodes

• \{(MBR; node\_ptr)\} for non-leaf nodes
Insertion Processes

A new index entry

ChooseLeaf

Has room

Yes

Install X

No

SplitNode

Adjust all related entries

AdjustTree

Different variant:

- Exhaustive
- Quadratic
- Linear
- Packed
- Hilbert Packed
- …etc.
Processes of Quadratic Spilt
(page 52 in Guttman’s paper [1])

Pick first entry for each group
Run PickSeeds
Processes of Quadratic Spilt
(page 52 in Guttman’s paper)

PickSeeds

PS1 [Calculate inefficiency of grouping entries together]
For each pair of E1 and E2, compose a rectangle R including E1 and E2
Calculate \( d = \text{area}(R) - \text{area}(E1) - \text{area}(E2) \)

PS2 [Choose the most wasteful pair ]
Choose the pair with the largest \( d \)
Processes of Quadratic Spilt
(page 52 in Guttman’s paper)

Pick first entry for each group (PickSeeds)

<table>
<thead>
<tr>
<th>G1</th>
<th>G2</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>m</td>
</tr>
</tbody>
</table>

Check if done

No

Select entry to assign (PickNext)
Processes of Quadratic Split
(page 52 in Guttman’s paper)

Pick first entry for each group (PickSeeds)

<table>
<thead>
<tr>
<th>G1</th>
<th>G2</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>m</td>
</tr>
</tbody>
</table>

Check if done

PN1 [Determine cost of putting each entry in each group] For each entry E
calculate $d1 =$ the increased MBR area required for G1
calculate $d2 =$ the increased MBR area required for G2

PN2 [Find entry with greatest preference for one group] Choose the entry with the maximum difference between $d1$ and $d2$
Processes of Quadratic Split
(page 52 in Guttman’s paper)

Pick first entry for each group (PickSeeds)

Check if done

Select entry to assign (PickNext)
Processes of Quadratic Split

(page 52 in Guttman's paper)

Pick first entry for each group (PickSeeds)

Check if done

No

Select entry to assign (PickNext)
Processes of Quadratic Spilt
(page 52 in Guttman’s paper)

Pick first entry for each group (PickSeeds)

Check if done

No

Select entry to assign (PickNext)
Processes of Quadratic Split
(page 52 in Guttman’s paper)

Pick first entry for each group (PickSeeds)

<table>
<thead>
<tr>
<th>G1</th>
<th>G2</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>m</td>
</tr>
</tbody>
</table>

Check if done

No

Select entry to assign (PickNext)

<table>
<thead>
<tr>
<th>G1</th>
<th>G2</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>m</td>
</tr>
<tr>
<td>k</td>
<td>X</td>
</tr>
<tr>
<td>j</td>
<td></td>
</tr>
</tbody>
</table>
Excercise

- \((m,M) = (2,4)\)
- Build a R-Tree for these spatial data
R-trees: Search
R-trees: Search

• Main points:
  – every parent node completely covers its ‘children’
  – a child MBR may be covered by more than one parent - it is stored under ONLY ONE of them. (ie., no need for dup. elim.)
  – a point query may follow multiple branches.
Search Object in R-Tree

Input \((T, j)\)

- Leaf? [Yes, No]
  - Yes: Check each entry \(E\) of \(T\)
  - No: For each subtree \(E\) of \(T\)
    - \(E = j\) [Yes, No]
      - Yes: Qualifying record
      - No: \(E\) Contains \(j\) [Yes, No]
        - Yes: \((E, j)\)
          - No: Continue searching
Overlap query in R-Tree (find objects that overlap with \( Z \))

Input \((T, Z)\)

Leaf?  

Yes

Check each entry \( E \) of \( T \)

Overlap  

Yes

Qualifying record

No

Search each subtree \( E \) of \( T \)

Overlap  

Yes

Overlap \((E, j)\)

A

B

G

E

C

x

m

j

k

l

i

h

g

de

d

C. Shahabi
Main Drawbacks of R-Tree

• R-tree is not unique, rectangles depend on how objects are inserted and deleted from the tree.

• In order to search some object you might have to go through several rectangles or the whole database
  – Why?
  – Solution?
R⁺-Tree

- Overcome problems with R-Tree
- If node overlaps with several rectangles insert the node in all
- Decompose the space into disjoint cells
R+-Tree Properties

• R+-tree used approach of discomposing space into cells
  – R+-trees deals with collection of objects bounded by rectangles
• R+-trees is extension of k-d-B-tree
• Retrieval times are smaller
• When summing the objects, needs eliminate duplicates
• Again, it is data-dependent
R+-Tree

- R+ trees differ from R trees in that:
  - Nodes are not guaranteed to be at least half filled
  - The entries of any internal node do not overlap
  - An object may be stored in more than one leaf node
R+-Tree

Advantages

• Because nodes are not overlapped with each other, point query performance benefits since all spatial regions are covered by at most one node.
• A single path is followed and fewer nodes are visited than with the R-tree

Disadvantages

• Since rectangles are duplicated, an R+ tree can be larger than an R tree built on same data set.
• Construction and maintenance of R+ trees is more complex than the construction and maintenance of R trees and other variants of the R tree.
R*-tree

- The original R-tree tries to minimize the area of each enclosing rectangle in the index nodes.
- Is there any other property that can be optimized?

R*-tree → Yes!
R*-tree

- The R* tree uses the same algorithm as the R-tree for query and delete operations. The primary difference is the insert algorithm
  - specifically how it chooses which branch to insert the new node into and the methodology for splitting a node that is full.
R*-tree

- Optimization Criteria:
  - (O1) Area covered by an index MBR
  - (O2) Overlap between index MBRs
  - (O3) Margin of an index rectangle
  - (O4) Storage utilization

- Sometimes it is impossible to optimize all the above criteria at the same time!
R*-tree

• ChooseSubtree:
  – If next node is a leaf node, choose the node using the following criteria:
    • Least overlap enlargement
    • Least area enlargement
    • Smaller area
  – Else
    • Least area enlargement
    • Smaller area
References

• Antonin Guttman, R-trees: a dynamic index structure for spatial searching, Proceedings of the 1984 ACM SIGMOD international conference on Management of data, June 18-21, 1984, Boston, Massachusetts

• Norbert Beckmann, et al. , The R*-tree: an efficient and robust access method for points and rectangles, SIGMOD 1990

• Roussopoulos et al. , The R+-Tree: A Dynamic Index for Multi-Dimensional Objects, VLDB 1987

• National Technical University of Athens , Theoretical Computer Science II: Advanced Data Structures