Spatial Database Systems

R-trees

Gilberto Ribeiro de Queiroz
(gribeiro@dpi.inpe.br)

Roadmap

- Geometric Approximations
- R-tree
- R^*-tree
- Packed R-trees
- R^+-tree

Geometric Approximations

Well Known Approximations

MBR

RMBR

Convex Hull

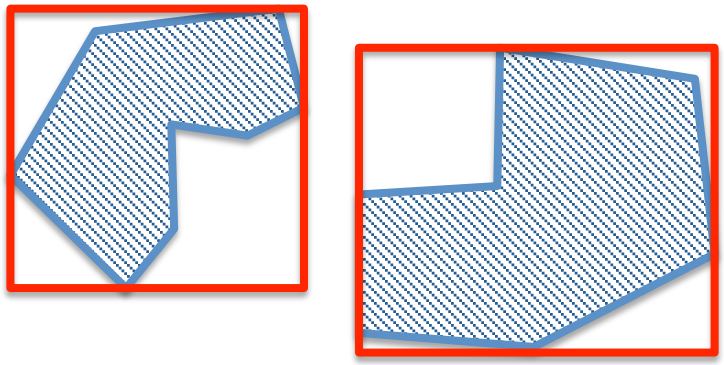
MBC

MBE

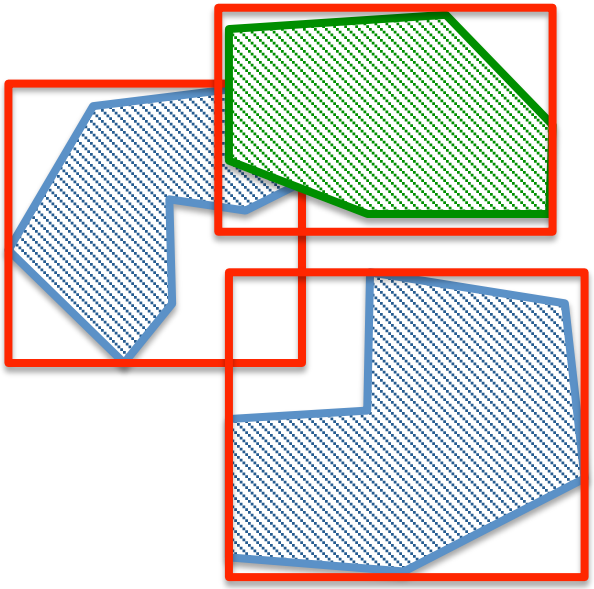
Minimum Bounding Rectangle (MBR)

- MBR is the most used approximation:
  - Also known as: Minimum Bounding Box (MBB) or Bounding Box (BBOX);
  - A common approximation for objects with extension (lines and polygons);
  - In 2D space only a pair of coordinates are needed to represent it;
  - Fast filter for geometry relationship test.

MBR in Practice



Case 1) If the rectangles don't intersect: then the geometries will not intersect too.



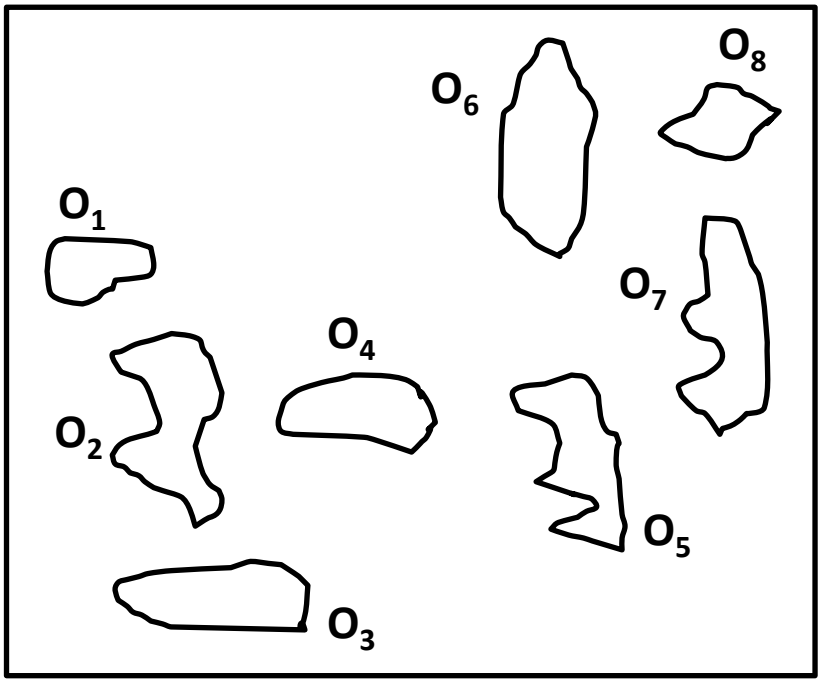
Case 2) If the rectangles intersect: then the geometries ***may*** intersect.

R-tree

(Guttman, 1984)

R-tree: Overview

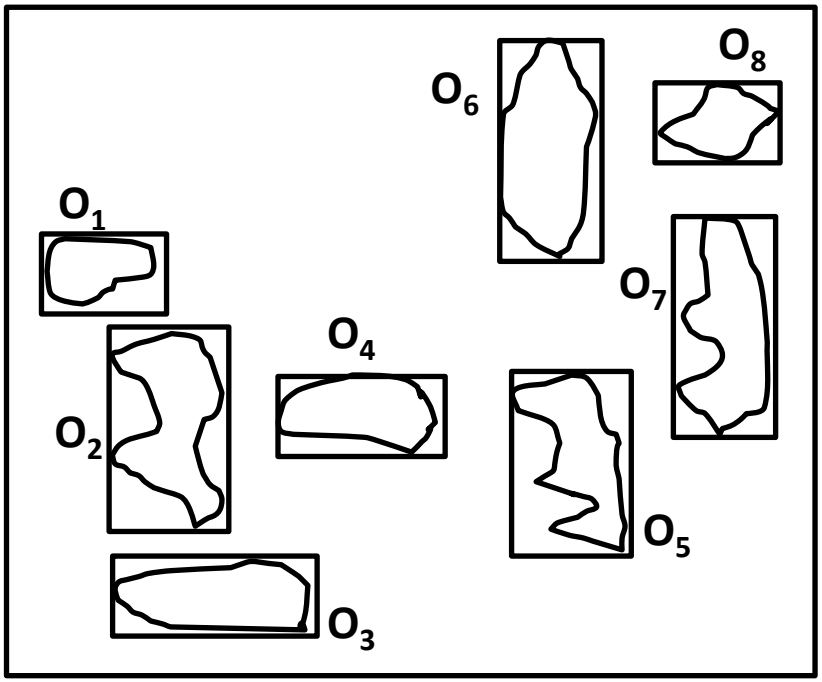
- Given a set o spatial objects in a space R^k ($k > 1$), the R-tree index organize the underlying space into a hierarchy of possible overlapping k-dimensional intervals:



8 Objects in R^2

R-tree: Overview

- Given a set o spatial objects in a space R^k ($k > 1$), the R-tree index organize the underlying space into a hierarchy of possible overlapping k-dimensional intervals:

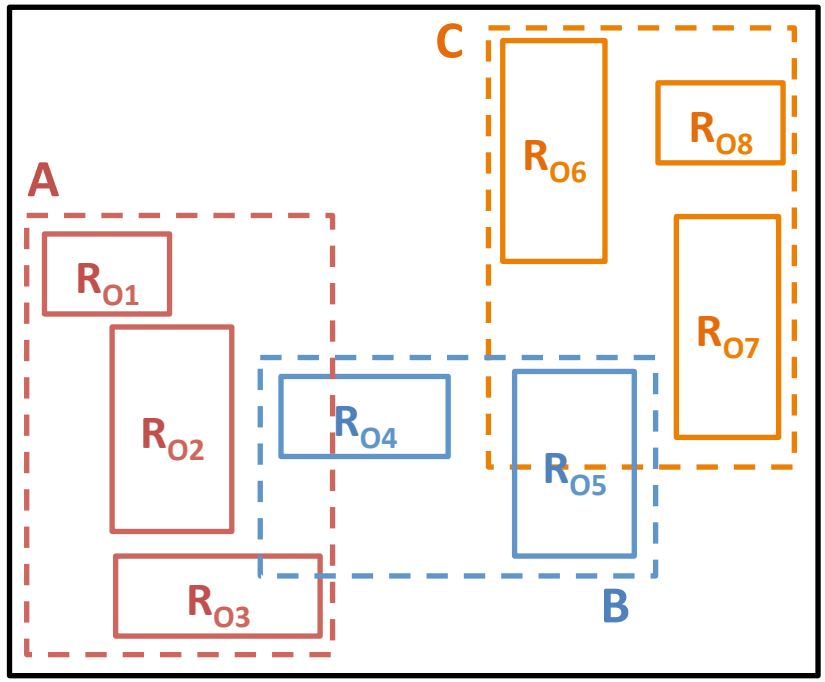


8 Objects in R^2

Use MBR for object approximation

R-tree: Overview

- Given a set o spatial objects in a space R^k ($k > 1$), the R-tree index organize the underlying space into a hierarchy of possible overlapping k-dimensional intervals:



8 Objects in R^2

Use MBR for object approximation

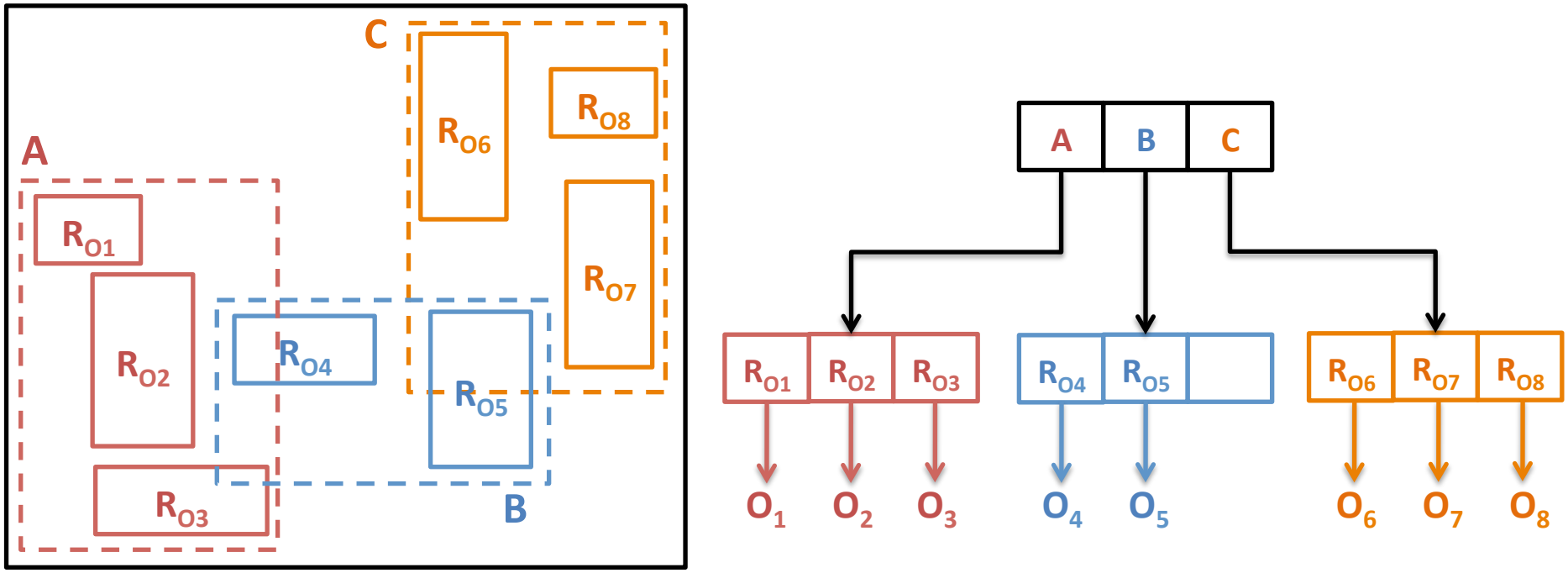
Group rectangles in a hierarchy

R-tree: Overview

- Intervals are organized in a tree data structure:
  - Leaf nodes: $\left[(I, tuple\text{-}id)\right]$
  - Child nodes: $\left[(I, child\text{-}ptr)\right]$

where $I = (I_0, I_1, \ldots, I_{k-1})$

$I_i = [a, b]$



An R-tree must satisfy the following properties

- The maximum number of entries in a node: M
- The minimum number of entries in a node: $m \geq \frac{M}{2}$
- Every node contains between *m* and *M* valid entries, unless it is the root node.
- If the tree is more than one level high, the root has at least two child nodes.
- For each entry of the form $(I, \textit{child-ptr})$, I is the smallest interval containing the intervals in the child node.
- For each entry of the form $(I, \textit{tuple-id})$, I is the smallest interval containing the spatial object.
- All leaves appear on the same level.

R-tree: Properties

- The height of an R-tree for indexing *N* spatial objects is: $\left(\log_m N\right) - 1$

- The maximum number of nodes:

$$\left\lceil \frac{N}{m} \right\rceil + \left\lceil \frac{N}{m^2} \right\rceil + \ldots + 1$$

- Worst case space utilization of nodes (except the root): $\frac{m}{M}$

R-tree: Searching

- Query algorithms traverse the tree in a top-down fashion:

```
Description:
  n -> search node
  I -> search interval

 1.    search(n, I)
 2.      if(is_not_leaf(n))
 3.        for_all entry in nodo do
 4.          if(intersects(entry, I))
 5.            search(entry.child-ptr, I)
 6.      else /* it is a leaf node */
 7.        for_all entry in nodo do
 8.          if(intersects(entry, I))
 9.            emit_found_candidate(entry.tuple-id)
```

R-tree: Insertions and Removal

Key idea: apply heuristics to minimize the number of paths traversed for improving query performance

R-tree: Insertion Operations

- Remarks:
  - When the number of valid entries in a node *n* gets larger than *M*, a new sibling node *n'* must be created and filled with part of the entries from node *n*.
  - If *n*'s parent node doesn't have room for *n'*, it has to split too.

R-tree: Insertion Algorithm

```
Description:
  I -> interval to be inserted
  n -> node

 1.    insert(I, n)
 2.      leaf_node <- choose_leaf(I, n)
 3.      if(has_room(leaf_node))
 4.        insert(I, leaf_node)
 5.        adjust_tree(leaf_node)
 6.      else /* leaf node doesn't have one more room */
 7.        leaf_node' <- split(leaf_node)
 8.        adjust_tree(leaf_node, leaf_node')
```

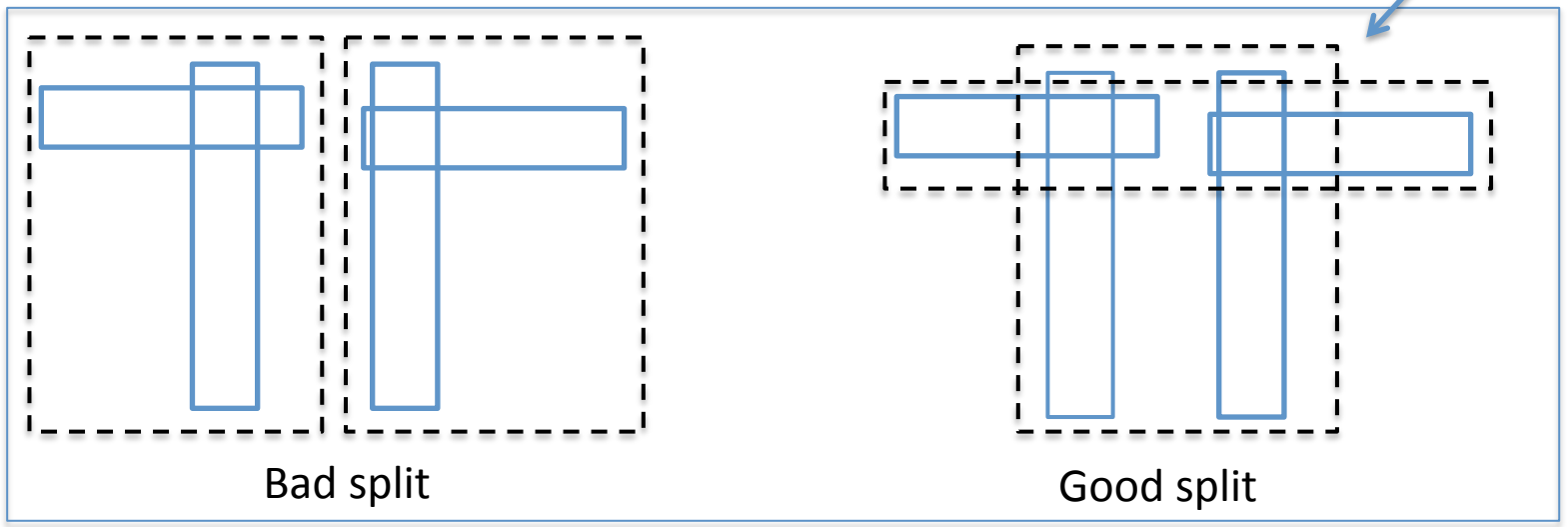
R-tree: Insertion Algorithm

- choose_leaf:
  - Choose the leaf that requires the least interval (or MBR) enlargement.

- adjust_tree:
  - Propagate changes upward.
  - splits may be propagate upward
  - There are several strategies for splitting nodes

R-tree: Node Splitting

- Guttman (1984) proposed 3 strategies for the partitioning of *M + 1* entries into two groups:
  - Exhaustive algorithm:
    - Evaluate all the 2^{M-1} possible grouping choices => Computational Complexity: $O(2^M)$
  - Quadratic algorithm: Computational Complexity: $O(M^2)$
  - Linear algorithm: Computational Complexity: O(M)



Source: Guttman (1984)

R-tree: Removal Operations

- When the number of entries drops below *m*, *m - 1* entries should be moved to other sibling nodes.

R-tree: Deletion Algorithm

```
Description:
  I -> interval of the object to be removed from the index
  n -> node

 1.    delete(I, n)
 2.      leaf_node <- find_leaf(I, n)
 3.      remove_entry(I, leaf_node)
 4.      condense_tree(leaf_node)

 /* has the root node n only one child? */
 5.      if(num_children(root) == 1)
 6.        n <- first_child(n).child-ptr
```

R-tree: Deletion Algorithm

- condense_tree:
  - It will eliminate the node if it has few entries and will perform the relocation of these entries (calling insert for them).
  - Propagate node elimination upward (updating intervals on the path to the root).

R*-trees

(Beckmann et al., 1990)

R*-tree

- Optimization is based on a combination of area, margin and overlap of each enclosing rectangle:
  - Not just area.

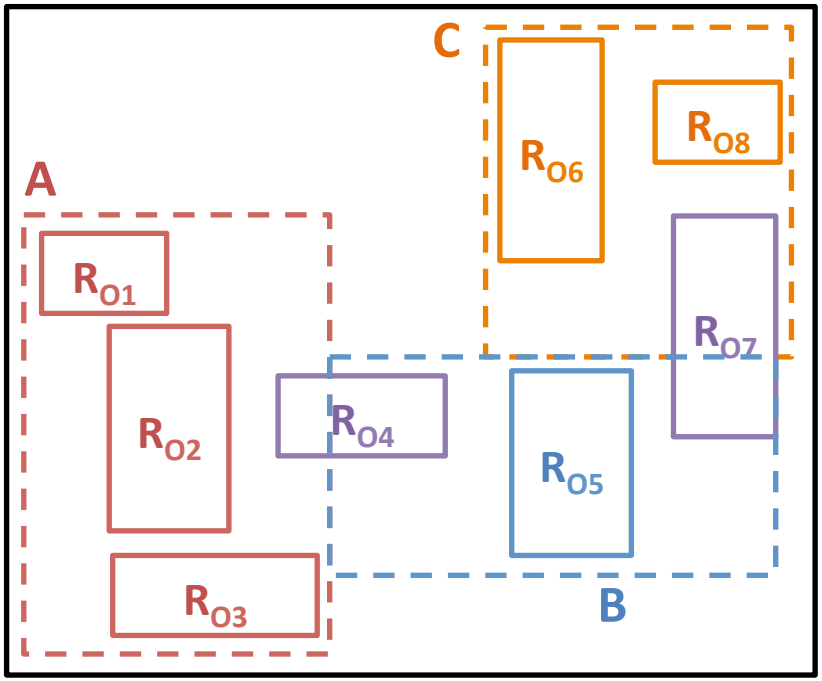
- Outperforms Guttman's original linear and quadratic variants.

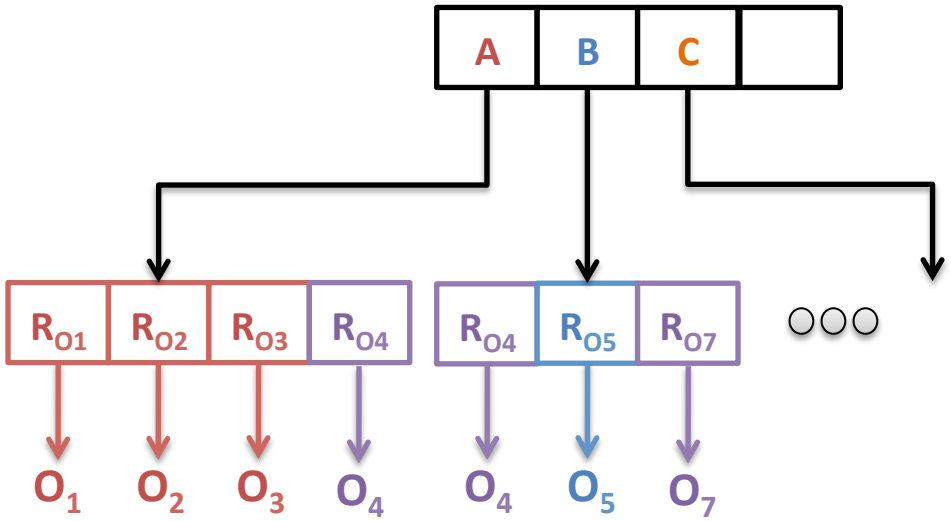
R^+-trees

(Sellis et al., 1987)

R^+-trees

- Variant of Guttman's R-tree that adopts non-overlapping intervals:
  - Attempt to improve search operations;
  - Drawback: need some extra space.





Packed R-trees

(Roussopoulos and Leifker, 1985)

(Kamel and Faloutsos, 1993)

Packed R-trees

- Methods for bottom-up construction of an R-tree.
- Pack:
  - attempt to maximize the fill factor for nodes
- Well known variations:
  - Lower-x packed R-tree
  - Distance Sorted R-tree
  - Hilbert R-tree

Last Question…

How to process efficiently the nearest neighbor query?

References

References

- Antonin Guttman. 1984. ***R-trees: a dynamic index structure for spatial searching***. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data* (SIGMOD '84). ACM, New York, NY, USA, pp. 47-57.

- Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. 1987. ***The R+-Tree: A Dynamic Index for Multi-Dimensional Objects***. In *Proceedings of the 13th International Conference on Very Large Data Bases* (VLDB '87), Peter M. Stocker, William Kent, and Peter Hammersley (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 507-518.

- Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. ***The R*-tree: an efficient and robust access method for points and rectangles***. *SIGMOD Rec.* 19, 2 (May 1990), pp. 322-331.

Packed R-trees

- Nick Roussopoulos and Daniel Leifker. 1985. ***Direct spatial search on pictorial databases using packed R-trees***. In *Proceedings of the 1985 ACM SIGMOD international conference on Management of data* (SIGMOD '85). ACM, New York, NY, USA, 17-31.

- Ibrahim Kamel and Christos Faloutsos. 1993. ***On packing R-trees***. In *Proceedings of the second international conference on Information and knowledge management* (CIKM '93), Bharat Bhargava, Tim Finin, and Yelena Yesha (Eds.). ACM, New York, NY, USA, 490-499.

Cost Models

- Yannis Theodoridis and Timos Sellis. 1996. ***A model for the prediction of R-tree performance***. In *Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems* (PODS '96). ACM, New York, NY, USA, 161-171.