15-826: Multimedia Databases and Data Mining

Lecture #6: Spatial Access Methods
Part III: R-trees
C. Faloutsos

Must-read material

- MM-Textbook, Chapter 5.2
- Ramakrishnan+Gehrke, Chapter 28.6
R-trees – impact:

- Popular method; like multi-d B-trees
- guaranteed utilization; fast search (low dim’s)
- Used in practice:
  - Oracle spatial (R-tree default; z-order, too)
    docs.oracle.com/html/A88805_01/sdo_intr.htm
  - IBM-DB2 spatial extender
  - Postgres: create index ... using [ rtree | gist ]
  - Sqlite3: www.sqlite.org/rtree.html

Outline

Goal: ‘Find similar / interesting things’
- Intro to DB
- Indexing - similarity search
- Data Mining
Indexing - Detailed outline

• primary key indexing
• secondary key / multi-key indexing
• spatial access methods
  – problem dfn
  – z-ordering
  – R-trees
  – ...
• text
• ...

Indexing - more detailed outline

• R-trees
  – main idea; file structure
  – algorithms: insertion/split
  – deletion
  – search: range, nn, spatial joins
  – performance analysis
  – variations (packed; hilbert;...)
Spatial Access Methods - problem

- Given a collection of geometric objects (points, lines, polygons, ...)
- Find cities within 100mi from Pittsburgh

Solution#2: R-trees

- multi-dim trees
- Allow nodes to overlap
- Guaranteed 50% utilization
R-trees

- z-ordering: cuts regions to pieces -> dup. elim.
- how could we avoid that?
- Idea: try to extend/merge B-trees and k-d trees

R-trees

- [Guttman 84] Main idea: allow parents to overlap!

Antonin Guttman
[http://www.baymoon.com/~tg2/]
R-trees

- [Guttman 84] Main idea: allow parents to overlap!
  - => guaranteed 50% utilization
  - => easier insertion/split algorithms.
  - (only deal with Minimum Bounding Rectangles - MBRs)

R-trees

- eg., w/ fanout 4: group nearby rectangles to parent MBRs; each group -> disk page
R-trees

- eg., w/ fanout 4:
R-trees - format of nodes

- \{\text{MBR}; \text{obj-ptr}\} for leaf nodes

- \{\text{MBR}; \text{node-ptr}\} for non-leaf nodes
R-trees - range search?

R-trees - range search?
R-trees - range search

Observations:
• 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.)

R-trees - range search

Observations - cont’d
• a point query may follow multiple branches.
• everything works for any dimensionality
Indexing - more detailed outline

• R-trees
  – main idea; file structure
  – algorithms: insertion/split
  – deletion
  – search: range, nn, spatial joins
  – performance analysis
  – variations (packed; hilbert;…)

R-trees - insertion

• eg., rectangle ‘X’
R-trees - insertion

- eg., rectangle ‘X’

R-trees - insertion

- eg., rectangle ‘Y’
R-trees - insertion

• eg., rectangle ‘Y’: extend suitable parent.

• Q: how to measure ‘suitability’?
R-trees - insertion

- eg., rectangle ‘Y’: extend suitable parent.
- Q: how to measure ‘suitability’?
- A: by increase in area (volume) (more details: later, under ‘performance analysis’)
- Q: what if there is no room? how to split?

R-trees - insertion

- eg., rectangle ‘W’
R-trees - insertion

• eg., rectangle ‘W’ - focus on ‘P1’ - how to split?

P1

K

W

• (A1: plane sweep, until 50% of rectangles)
• A2: ‘linear’ split
• A3: quadratic split
• A4: exponential split
R-trees - insertion & split

• pick two rectangles as ‘seeds’;
• assign each rectangle ‘R’ to the ‘closest’ ‘seed’

Q: how to measure ‘closeness’?
R-trees - insertion & split

- pick two rectangles as ‘seeds’;
- assign each rectangle ‘R’ to the ‘closest’ ‘seed’
- Q: how to measure ‘closeness’?
- A: by increase of area (volume)
R-trees - insertion & split

• pick two rectangles as ‘seeds’;
• assign each rectangle ‘R’ to the ‘closest’ ‘seed’

• smart idea: pre-sort rectangles according to delta of closeness (ie., schedule easiest choices first!)
R-trees - insertion - pseudocode

• decide which parent to put new rectangle into (‘closest’ parent)
• if overflow, split to two, using (say,) the quadratic split algorithm
  – propagate the split upwards, if necessary
• update the MBRs of the affected parents.

R-trees - insertion - observations

• many more split algorithms exist (see refs)
Indexing - more detailed outline

- R-trees
  - main idea; file structure
  - algorithms: insertion/split
  - deletion
    - search: range, nn, spatial joins
    - performance analysis
    - variations (packed; hilbert;...)

R-trees - deletion

- delete rectangle
- if underflow
  - ??
R-trees - deletion

- delete rectangle
- if underflow
  - temporarily delete all siblings (!);  
  - delete the parent node and
  - re-insert them

Indexing - more detailed outline

- R-trees
  - main idea; file structure
  - algorithms: insertion/split
  - deletion
  - search: range, nn, spatial joins
  - performance analysis
  - variations (packed; hilbert;...
R-trees - range search

pseudocode:
check the root
for each branch,
    if its MBR intersects the query rectangle
        apply range-search (or print out, if this is a leaf)

R-trees - nn search

[Diagram showing R-tree structure with points and bounding boxes]
R-trees - nn search

• Q: How? (find near neighbor; refine...)

A1: depth-first search; then, range query
R-trees - nn search

- A1: depth-first search; then, range query

![Diagram of R-trees](image)
R-trees - nn search

- A2: [Roussopoulos+, sigmod95]:
  - priority queue, with promising MBRs, and their best and worst-case distance
- main idea:

Consider only P2 and P4, for illustration.
R-trees - nn search

best of P4

worst of P2

=> P4 is useless for 1-nn
R-trees - nn search

- what is really the worst of, say, P2?

• A: the smallest of the two red segments!
Indexing - more detailed outline

- R-trees
  - main idea; file structure
  - algorithms: insertion/split
  - deletion
  - search: range, nn, **spatial joins**
  - performance analysis
  - variations (packed; hilbert;...)

R-trees - spatial joins

**Spatial joins**: find (quickly) all counties intersecting lakes

![Spatial joins example image]
R-trees - spatial joins

Spatial joins: find (quickly) all counties intersecting lakes
R-trees - spatial joins

Assume that they are both organized in R-trees:
R-trees - spatial joins

for each parent P1 of tree T1
for each parent P2 of tree T2
if their MBRs intersect,
  process them recursively (ie., check their children)

R-trees - spatial joins

Improvements - variations:
- [Seeger+, sigmod 92]: do some pre-filtering; do plane-sweeping to avoid $N1 \times N2$ tests for intersection
- [Lo & Ravishankar, sigmod 94]: ‘seeded’ R-trees
  (FYI, many more papers on spatial joins, without R-trees: [Koudas+ Sevcik], e.t.c.)
Indexing - more detailed outline

- R-trees
  - main idea; file structure
  - algorithms: insertion/split
  - deletion
  - search: range, nn, spatial joins
  - performance analysis
  - variations (packed; hilbert;...)

R-trees - performance analysis

- How many disk (=node) accesses we’ll need for
  - range
  - nn
  - spatial joins
- why does it matter?
R-trees - performance analysis

- How many disk (=node) accesses we’ll need for
  - range
  - nn
  - spatial joins
- why does it matter?
- A: because we can design split etc algorithms accordingly; also, do query-optimization
R-trees - performance analysis

• motivating question: on, e.g., split, should we try to minimize the area (volume)? the perimeter? the overlap? or a weighted combination? why?

• Thus, given a tree with N nodes (i=1, ... N) we expect

\[
\text{#DiskAccesses}(q1,q2) = \sum (x_{i,1} + q1) \times (x_{i,2} + q2) = \sum (x_{i,1} \times x_{i,2}) + q2 \times \sum (x_{i,1}) + q1 \times \sum (x_{i,2}) + q1 \times q2 \times N
\]

volume

surface area

count
R-trees - performance analysis

• How many disk accesses for range queries?
  – query distribution wrt location? uniform; (biased)
  – “ “ wrt size? uniform

Proof
R-trees - performance analysis

• easier case: we know the positions of parent MBRs, eg:

Proof

R-trees - performance analysis

• How many times will P1 be retrieved (unif. queries)?

Proof
R-trees - performance analysis

• How many times will P1 be retrieved (unif. POINT queries)?

Proof

\[ A = x_1 \times x_2 \]
R-trees - performance analysis

- How many times will P1 be retrieved (unif. queries of size q1 x q2)?

Proof
R-trees - performance analysis

• How many times will P1 be retrieved (unif. queries of size $q_1 \times q_2$)?
R-trees - performance analysis

• How many times will P1 be retrieved (unif. queries of size q1xq2)? A: \((x1+q1)*(x2+q2)\)

\[
\begin{align*}
#\text{DiskAccesses}(q1,q2) &= \sum (x_{i,1} + q1) \cdot (x_{i,2} + q2) \\
&= \sum (x_{i,1} \cdot x_{i,2}) + q2 \cdot \sum (x_{i,1}) + q1 \cdot \sum (x_{i,2}) + q1 \cdot q2 \cdot N
\end{align*}
\]
**R-trees - performance analysis**

- Thus, given a tree with N nodes (i=1, ... N) we expect
  
  \[ \text{DiskAccesses}(q1,q2) = \sum (x_{i,1} + q1) \times (x_{i,2} + q2) \]
  
  \[ = \sum (x_{i,1} \times x_{i,2}) + q2 \times \sum (x_{i,1}) + q1 \times \sum (x_{i,2}) + q1 \times q2 \times N \]

**Observations:**

- for point queries: only volume matters
- for horizontal-line queries: (q2=0): vertical length matters
- for large queries (q1, q2 >> 0): the count N matters
R-trees - performance analysis

Observations (cont’ ed)
• overlap: does not seem to matter
• formula: easily extendible to $n$ dimensions
• (for even more details: [Pagel +, PODS93], [Kamel+, CIKM93])

Berndt-Uwe Pagel

R-trees - performance analysis

Conclusions:
• splits should try to minimize area and perimeter
• i.e., we want **few, small, square-like** parent MBRs
• rule of thumb: shoot for queries with $q1=q2 = 0.1$ (or $=0.5$ or so).
Indexing - more detailed outline

- R-trees
  - main idea; file structure
  - algorithms: insertion/split
  - deletion
  - search: range, nn, spatial joins
  - performance analysis
- variations (packed; hilbert;...)

R-trees - variations

- what about static datasets (no ins/del/upd)?
- Q: Best way to pack points?
R-trees - variations

• what about static datasets (no ins/del/upd)?
• Q: Best way to pack points?
• A1: plane-sweep
  great for queries on ‘x’;
  terrible for ‘y’
R-trees - variations

• what about static datasets (no ins/del/upd)?
• Q: Best way to pack points?
  • A1: plane-sweep
    great for queries on ‘x’;
    terrible for ‘y’
  • Q: how to improve?

R-trees - variations

• A: plane-sweep on HILBERT curve!
R-trees - variations

- A: plane-sweep on HILBERT curve!
- (see [Kamel+, VLDB’94]

Solution#2: R-trees

- multi-dim trees
- Allow nodes to overlap
- Guaranteed 50% utilization – fast search (in low dim’s)
R-trees - conclusions

• ...

• Used in practice:
  – Oracle spatial (R-tree default; z-order, too)
    docs.oracle.com/html/A88805_01/sdo_intr.htm
  – IBM-DB2 spatial extender
  – Postgres: create index ... using [ rtree | gist ]
  – Sqlite3: www.sqlite.org/rtree.html

• R* variation is popular

References

• Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger: The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. ACM SIGMOD 1990: 322-331

References

• Jagadish, H. V. (May 23-25, 1990). Linear Clustering of Objects with Multiple Attributes. ACM SIGMOD Conf., Atlantic City, NJ.

• Ibrahim Kamel, Christos Faloutsos: On Packing R-trees, CIKM, 1993

References, cont’’d


Other resources

• Java applets and more info:
  donar.umiacs.umd.edu/quadtree/points/rtrees.html