
R-trees

Christos Faloutsos
www.cs.cmu.edu/~christos

Roadmap

1) Roots: System R and Ingres
2) Implementation: buffering, indexing, q-opt
3) Transactions: locking, recovery
4) Distributed DBMSs
5) Parallel DBMSs: Gamma, Alphasort
6) O/R DBMS
7) Data Analysis - data mining
8) Benchmarks
9) vision statements
 extras (streams/sensors, graphs, multimedia, web, fractals)

Detailed roadmap

1) Roots: System R and Ingres
2) Implementation: buffering, indexing, q-opt
   - OS support for DBMS
     - R-trees and GiST
       - Z-ordering
       - Buffering
     - ...
3) Transactions: locking, recovery

Outline

• R-trees
  – Problem definition - Spatial Access Methods
  – 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, ...)
• organize them on disk, to answer spatial queries (like??)
Spatial Access Methods - problem

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer
  - point queries
  - range queries
  - k-nn queries
  - spatial joins (‘all pairs’ queries)

SAMs - motivation

- Q: applications?
SAMs - motivation

- traditional DB
- GIS

age

salary

CAD/CAM

find elements too close to each other

Q: how would you organize, e.g., n-dim points, on disk? (C points per disk page)

SAMs: solutions

- z-ordering
- R-trees
- (grid files)

Outline

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

15-721 C. Faloutsos
**R-trees**

- How to group nearby points/shapes together?
- Idea: try to extend/merge B-trees and k-d trees

**R-trees**

- But: insertions/deletions are tricky (splits may propagate downwards and upwards)
- no guarantee on space utilization

**R-trees**

- e.g., w/ fanout 4: group nearby rectangles to parent MBRs; each group -> disk page

**R-trees**

- e.g., w/ fanout 4:

  - [Robinson, 81]: if \( f \) is the fanout, split point-set in \( f \) parts; and so on, recursively
  - [Guttman 84] Main idea: allow parents to overlap!
    - => guaranteed 50% utilization
    - => easier insertion/split algorithms.
    - (only deal with Minimum Bounding Rectangles - MBRs)

- no guarantee on space utilization
R-trees

- e.g., w/ fanout 4:

R-trees - format of nodes

- \{(MBR; node-ptr)\} for non-leaf nodes

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. (i.e., no need for dup. elim.)
R-trees - range search
Observations - cont’d
- A point query may follow multiple branches.
- Everything works for any dimensionality

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.
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?

R-trees - insertion

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

R-trees - insertion & split

- pick two rectangles as ‘seeds’;
- assign each rectangle ‘R’ to the ‘closest’ ‘seed’
- (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 - 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 & 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 - observations

- many more split algorithms exist (next!)

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

R-trees - deletion

- variations: later (eg. Hilbert R-trees w/ 2-to-1 merge)

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

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

  A1: depth-first search; then, range query

R-trees - nn search

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

R-trees - nn search

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

best of P4

worst of P2

P4 is useless

for 1-nn

R-trees - nn search

what is really the worst of, say, P2?

R-trees - nn search

variations: [Hjaltason & Samet] incremental nn:
- build a priority queue
- scan enough of the tree, to make sure you have the k nn
- to find the (k+1)-th, check the queue, and scan some more of the tree
- 'optimal' (but, may need too much memory)
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

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?

Why does it matter?

R-trees - performance analysis

- A: because we can design split etc algorithms accordingly; also, do query-optimization
- motivating question: on, e.g., split, should we try to minimize the area (volume)? the perimeter? the overlap? or a weighted combination? why?

R-trees - performance analysis

- How many disk accesses for range queries?
  - query distribution wrt location?
  - “ “ wrt size?

Advanced - skip

Advanced - skip
R-trees - performance analysis

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

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

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

• How many times will P1 be retrieved (unif. queries of size q1xq2)?
R-trees - performance analysis

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

\[ x1 \]
\[ x2 \]
\[ q1 \]
\[ q2 \]

R-trees - performance analysis

- Thus, given a tree with N nodes (i=1, ... N) we expect
  \[ \#DiskAccesses(q1,q2) = \sum (x_{i1} + q1) * (x_{i2} + q2) \]
  \[ = \sum (x_{i1} * x_{i2}) + \]
  \[ q2 \] * \[ \sum (x_{i1}) \] +
  \[ q1 \] * \[ \sum (x_{i2}) \] +
  \[ q1 * q2 * N \]

R-trees - performance analysis

Observations:
- for point queries: only volume matters
- for horizontal-line queries: \( q2=0 \): vertical length matters
- for large queries (\( q1, q2 \gg 0 \)): 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])

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).
R-trees - performance analysis

- How many disk (=node) accesses we’ll need for:
  - range
  - nn
  - spatial joins

R-trees - performance analysis

Range queries - how many disk accesses, if we just now that we have
- N points in n-d space?
A: ?

R-trees - performance analysis

Range queries - how many disk accesses, if we just now that we have
- N points in n-d space?
A: can not tell! need to know distribution

R-trees - performance analysis

What are obvious and/or realistic distributions?
A: uniform
A: Gaussian / mixture of Gaussians
A: self-similar / fractal. Fractal dimension ~ intrinsic dimension

R-trees - performance analysis

What are obvious and/or realistic distributions?
Formulas for range queries and k-nn queries: use fractal dimension [Kamel+, PODS94], [Korn+ ICDE2000] [Kriegel+, PODS97]
Formulas for spatial joins of regions: open research question
**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**

Guttman’s R-trees sparked much follow-up work

- can we do better splits?
  - i.e, defer splits?

---

**R-trees - variations**

Guttman’s R-trees sparked much follow-up work

- can we do better splits?
  - i.e, defer splits?

**R-trees - variations**

A: R*-trees [Kriegel+, SIGMOD90]

- defer splits, by forced-reinsert, i.e.: instead of splitting, temporarily delete some entries, shrink overflowing MBR, and re-insert those entries
  - Which ones to re-insert?
  - How many?

**R-trees - variations**

Q: Other ways to defer splits?
**R-trees - variations**

Q: Other ways to defer splits?
A: Push a few keys to the closest sibling node (closest = ??)

---

**R-trees - variations**

R*-trees: Also try to minimize area AND perimeter, in their split.
Performance: higher space utilization; faster than plain R-trees. One of the most successful R-tree variants.

---

Guttman’s R-trees sparked much follow-up work
- can we do better splits?
  - what about static datasets (no ins/del/upd)?
  - Hilbert R-trees
- what about other bounding shapes?

---

**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’;
bad 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

- Dynamic (‘Hilbert R-tree):
  – each point has an ‘h’-value (hilbert value)
  – insertions: like a B-tree on the h-value
  – but also store MBR, for searches

R-trees - variations

Guttman’s R-trees sparked much follow-up work
- can we do better splits?
- what about static datasets (no ins/del/upd)?
- what about other bounding shapes?
- what about other bounding shapes? (and why?)

R-trees - variations

- A1: arbitrary-orientation lines (cell-tree, [Guenther])
- A2: P-trees (polygon trees) (MB polygon: 0, 90, 45, 135 degree lines)
Outline

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

- GiST

GiST: unifying the variants

- "Generalized Search Tree"
- common API for all these variants? (why?)

GiST

- source code at http://gist.cs.berkeley.edu, with
  - R-trees
  - R*-trees
  - etc

R-trees - variations

- A3: L-shapes; holes (hB-tree)
- A5: SR-trees [Katayama+, SIGMOD97] (used in Informedia)
R-trees - conclusions

- Popular method: like multi-d B-trees
- guaranteed utilization
- good search times (for low-dim. at least)
- Informix (→ IBM) ships DataBlade with R-trees

References


References cont’d


References, cont’d