
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) OO/OR 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)
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' within ε)

SAMs - motivation

• Q: applications?
SAMs - motivation

SAMs: solutions

- z-ordering
- R-trees
- (grid files)
Q: how would you organize, e.g., n-dim points, on disk? (C points per disk page)

Outline

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

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

(first attempt: k-d-B-trees)

- [Robinson, 81]: if $f$ is the fanout, split point-set in $f$ parts; and so on, recursively

(first attempt: k-d-B-trees)

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

• eg., w/ fanout 4:

R-trees - format of nodes

• {MBR; obj-ptr} for leaf nodes

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. (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

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’

```
H  J  J
D E
P2
P3
```

R-trees - insertion

- eg., rectangle ‘Y’

```
H  J  J
D E
P2
P3
```

R-trees - insertion

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

```
H  J  J
D E
P2
P3
```
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?

- Assign each rectangle ‘R’ to the ‘closest’ seed

R-trees - insertion & split

- pick two rectangles as ‘seeds’;
- assign each rectangle ‘R’ to the ‘closest’ ‘seed’
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 (i.e., 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 (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;...)

CMU SCS
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)

15-721  C. Faloutsos  55

R-trees - nn search

15-721  C. Faloutsos  56

R-trees - nn search

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

15-721  C. Faloutsos  57
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:

R-trees - nn search

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?

![Diagram showing worst of P2](image)

R-trees - nn search

• what is really the worst of, say, P2?
• A: the smallest of the two red segments!

![Diagram showing worst of P2](image)

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

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

Assume that they are both organized in R-trees:

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 $N_1 \times N_2$ 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

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

• How many disk accesses for range queries?
  - query distribution wrt location?
  - “ ” wrt size?
R-trees - performance analysis

- How many disk accesses for range queries?
  - query distribution wrt location? uniform; (biased)
  - " " wrt size? uniform

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

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

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

1 1

0 0

P1

x1

x2

0 1

R-trees - performance analysis

• How many times will P1 be retrieved (unif. POINT queries)? A: x1 * x2

1 1

0 0

P1

x1

x2

0 1

R-trees - performance analysis

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

1 1

0 0

P1

x1

x2

0 1

q1
R-trees - performance analysis

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

\[ (x_1 + q_1) \times (x_2 + q_2) \]

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

\[
\text{DiskAccesses}(q_1, q_2) = \\
\text{sum} (x_{i1} + q_1) \times (x_{i2} + q_2) \\
= \text{sum} (x_{i1} \times x_{i2}) + \\
q_2 \times \text{sum} (x_{i1}) + \\
q_1 \times \text{sum} (x_{i2}) + \\
q_1 \times q_2 \times N
\]

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

\[
\text{DiskAccesses}(q_1, q_2) = \\
\text{sum} (x_{i1} + q_1) \times (x_{i2} + q_2) \\
= \text{sum} (x_{i1} \times x_{i2}) + \text{volume} \\
q_2 \times \text{sum} (x_{i1}) + \text{surface area} \\
q_1 \times \text{sum} (x_{i2}) + \text{count} \\
q_1 \times q_2 \times N
\]
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

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

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 \( q1=0.1 \) (or \( =0.5 \) or so).
R-trees - performance analysis

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

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

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

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?
  - what about static datasets (no ins/del/upd)?
  - what about other bounding shapes?

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?

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.

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)?
    – 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?

15-721  C. Faloutsos  106

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’

15-721  C. Faloutsos  107

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’

15-721  C. Faloutsos  108
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!
- In fact, it can be made dynamic (how?), as well as to handle regions (how?)
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?

R-trees - variations
• what about other bounding shapes? (and why?)
• A1: arbitrary-orientation lines (cell-tree, [Guenther]
• A2: P-trees (polygon trees) (MB polygon: 0, 90, 45, 135 degree lines)
R-trees - variations

- A3: L-shapes; holes (hB-tree)
- A5: SR-trees [Katayama+, SIGMOD'97] (used in Informedia)

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: unifying the variants

• ```Generalized Search Tree```

• API:
  – consistent(n, q)  //returns NO or MAYBE
  – union(r1, ... rn) // finds, e.g., MBR
  – penalty(p, n)    //cost to put p in n
  – pickSplit(r1, ... rn) //split set of objects

GiST

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

Outline

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

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

References, cont’d


References, cont’d