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;...)

Reminder: problem

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

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
(first attempt: k-d-B-trees)
- [Robinson, 81]: if $f$ is the fanout, split point-set in $f$ parts; and so on, recursively

- 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)

- 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

• \{(MBR; obj-ptr)\} for leaf nodes

\[
\begin{array}{c}
\text{P1} \\
A & B & C \\
\text{P2} \\
D & E \\
\text{P3} \\
F \\
\text{P4} \\
G \\
\end{array}
\]

\[
\begin{array}{c}
x\text{-low;} x\text{-high} \\
y\text{-low;} y\text{-high} \\
\text{ptr} \\
\end{array}
\]

15-826 Copyright: C. Faloutsos (2006) #13

R-trees - format of nodes

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

\[
\begin{array}{c}
x\text{-low;} x\text{-high} \\
y\text{-low;} y\text{-high} \\
\text{ptr} \\
\end{array}
\]

15-826 Copyright: C. Faloutsos (2006) #14

R-trees - range search?

\[
\begin{array}{c}
P1 \\
P3 \\
I \\
\end{array}
\]

15-826 Copyright: C. Faloutsos (2006) #15

R-trees - range search?

\[
\begin{array}{c}
P1 \\
P3 \\
I \\
\end{array}
\]

15-826 Copyright: C. Faloutsos (2006) #16

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.)

15-826 Copyright: C. Faloutsos (2006) #17

R-trees - range search

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

15-826 Copyright: C. Faloutsos (2006) #18
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 ‘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 & 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’?

• assign each rectangle ‘R’ to the ‘closest’ ‘seed’

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

**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 (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 - 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

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

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

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

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

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

A: the smallest of the two red segments!

P2 is useless for 1-nn

• A: the smallest of the two red segments!
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 (i.e., check their children)

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+ Svevik], 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?

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

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

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

• why does it matter?

• A: because we can design split etc algorithms accordingly; also, do query-optimization
R-trees - performance analysis

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

R-trees - performance analysis

• How many times will \( P_1 \) be retrieved (unif. queries)?

R-trees - performance analysis

• How many times will \( P_1 \) be retrieved (unif. POINT queries)? A: \( x_1 \times x_2 \)

R-trees - performance analysis

• How many times will \( P_1 \) be retrieved (unif. queries of size \( q_1 \times q_2 \))?

R-trees - performance analysis

• How many times will \( P_1 \) be retrieved (unif. queries of size \( q_1 \times q_2 \))? A: \((x_1+q_1) \times (x_2+q_2)\)
**R-trees - performance analysis**

Thus, given a tree with $N$ nodes ($i = 1, \ldots, N$) we expect

$$\text{#DiskAccesses}(q_1,q_2) = \sum (x_{i,1} + q_1) * (x_{i,2} + q_2)$$

$$= \sum (x_{i,1} \cdot x_{i,2}) + q_2 * \sum (x_{i,1}) + q_1 * \sum (x_{i,2}) + q_1 * q_2 * N$$

**R-trees - performance analysis**

Thus, given a tree with $N$ nodes ($i = 1, \ldots, N$) we expect

$$\text{#DiskAccesses}(q_1,q_2) = \sum (x_{i,1} + q_1) * (x_{i,2} + q_2)$$

$$= \sum (x_{i,1} \cdot x_{i,2}) + q_2 * \sum (x_{i,1}) + q_1 * \sum (x_{i,2}) + q_1 * q_2 * N$$

**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

**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 $q_1=q_2 = 0.1$ (or $=0.5$ or so).
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

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;...)

Outline
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
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
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? A: 30%

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’;
terrible for ‘y’
- **Q: how to improve?**

- **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’

- **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

• In fact, it can be made dynamic (how?), as well as to handle regions (how?)
R-trees - variations

- What if we have regions, instead of points?
- I.e., how to impose a linear ordering ('h-value') on rectangles?
  - A1: h-value of center
  - A2: h-value of 4-d point (center, x-radius, y-radius)
  - A3: ...

R-trees - variations

- Can we do better splits?
- What about static datasets (no ins/del/upd)?
  - Hilbert R-trees - main idea
  - Handling regions
  - Performance/discussion
- What about other bounding shapes?

R-trees - variations

- with h-values, we can have deferred splits, 2-to-3 splits (3-to-4, etc)
- experimentally: faster than R*-trees (reference: [Kamel Faloutsos vldb 94])

R-trees - variations

- Gutman’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+, SIGMOD97] (used in Informedia)

Indexing - Detailed outline

- spatial access methods
  - problem dfn
  - z-ordering
  - R-trees
    - misc topics
      - grid files
      - dimensionality curse
      - metric trees
      - other nn methods
  - text, ...

R-trees - conclusions

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

References

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

References, cont’d