CMU SCS

15-826: Multimedia Databases and Data Mining

Lecture #6: Spatial Access Methods

Part III: R-trees

C. Faloutsos

CMU SCS

Must-read material

- MM-Textbook, Chapter 5.2
- Ramakrinshan+Gehrke, Chapter 28.6
- Guttman, A. (June 1984). *R-Trees: A Dynamic Index Structure for Spatial Searching*. Proc. ACM SIGMOD, Boston, Mass.

CMU SCS

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`

CMU SCS

Outline

Goal: 'Find similar / interesting things'

- Intro to DB
- ➡ Indexing - similarity search
- Data Mining

CMU SCS

Indexing - Detailed outline

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

CMU SCS

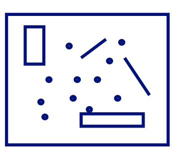
Indexing - more detailed outline

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

CMU SCS

Reminder: problem

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



CMU SCS

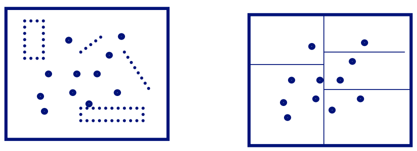
R-trees

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

CMU SCS

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

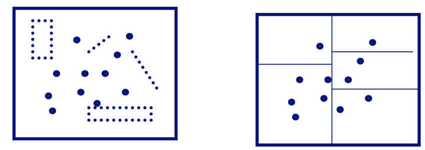
- [Robinson, 81]: if *f* is the fanout, split point-set in *f* parts; and so on, recursively



CMU SCS

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

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



CMU SCS

R-trees

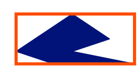
- [Guttman 84] Main idea: allow parents to overlap!

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

CMU SCS

R-trees

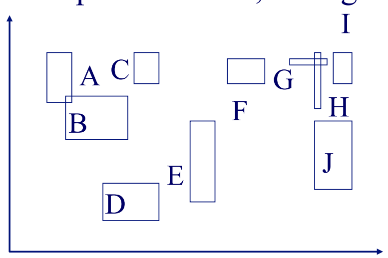
- [Guttman 84] Main idea: allow parents to overlap!
  - => guaranteed 50% utilization
  - => easier insertion/split algorithms.
  - (only deal with Minimum Bounding Rectangles - **MBR**s)



CMU SCS

R-trees

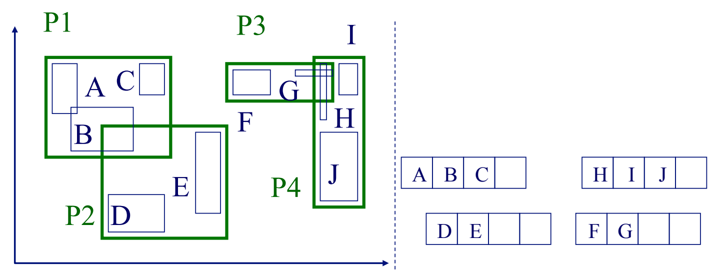
- eg., w/ fanout 4: group nearby rectangles to parent MBRs; each group -> disk page



CMU SCS

R-trees

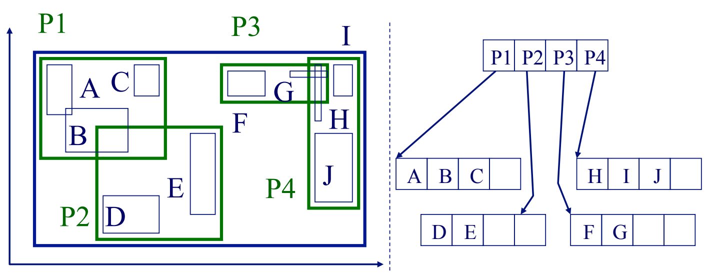
- eg., w/ fanout 4:



CMU SCS

R-trees

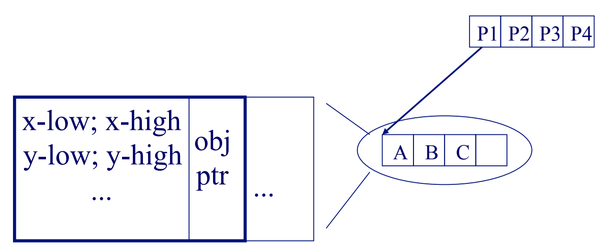
- eg., w/ fanout 4:



CMU SCS

R-trees - format of nodes

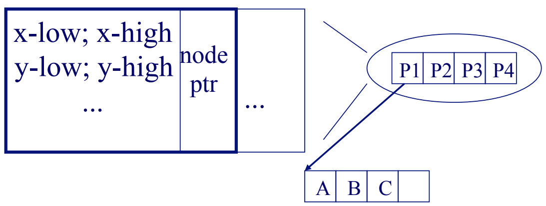
- {(MBR; obj-ptr)} for leaf nodes



CMU SCS

R-trees - format of nodes

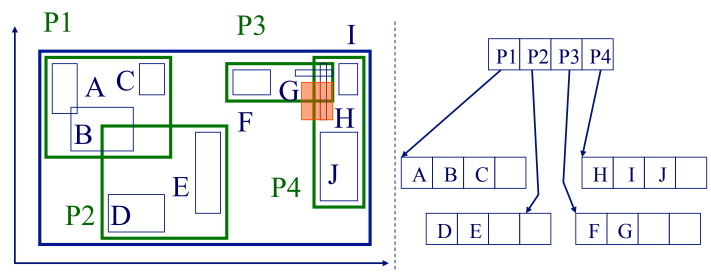
- {(MBR; node-ptr)} for non-leaf nodes



15-826 Copyright: C. Faloutsos (2017) #17

CMU SCS

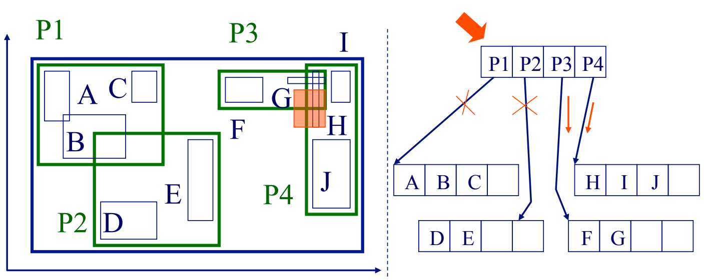
R-trees - range search?



15-826 Copyright: C. Faloutsos (2017) #18

CMU SCS

R-trees - range search?



15-826 Copyright: C. Faloutsos (2017) #19

CMU SCS

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 (2017) #20

CMU SCS

R-trees - range search

Observations - cont’ d

- a point query may follow multiple branches.
- everything works for **any** dimensionality

CMU SCS

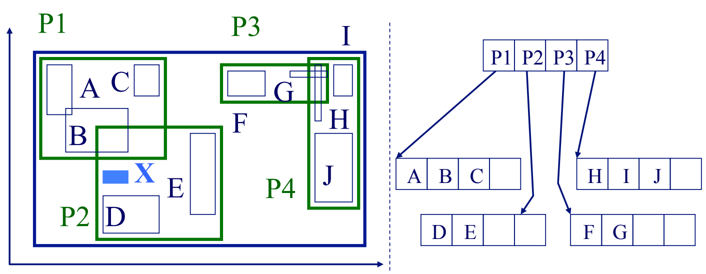
Indexing - more detailed outline

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

CMU SCS

R-trees - insertion

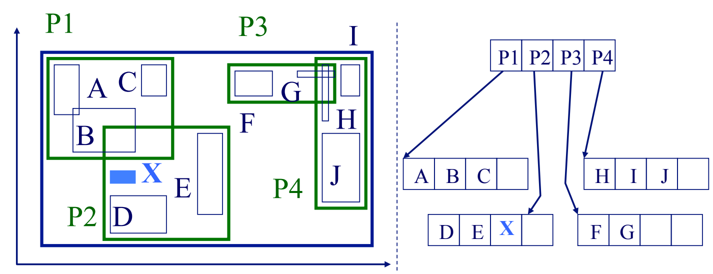
- eg., rectangle ‘X’



CMU SCS

R-trees - insertion

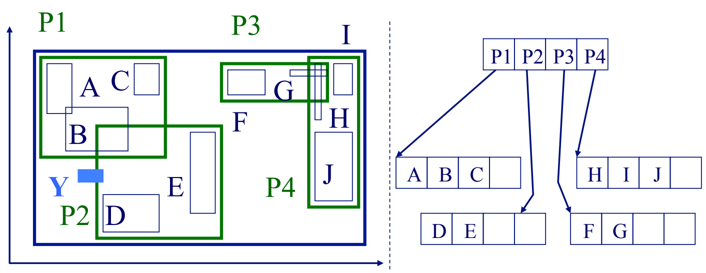
- eg., rectangle ‘X’



CMU SCS

R-trees - insertion

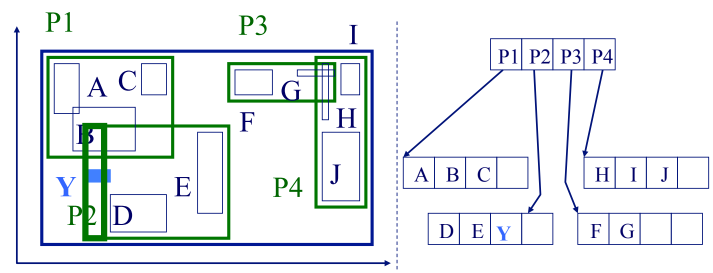
- eg., rectangle 'Y'



CMU SCS

R-trees - insertion

- eg., rectangle 'Y' : extend suitable parent.



CMU SCS

R-trees - insertion

- eg., rectangle 'Y' : extend suitable parent.
- Q: how to measure 'suitability' ?

CMU SCS

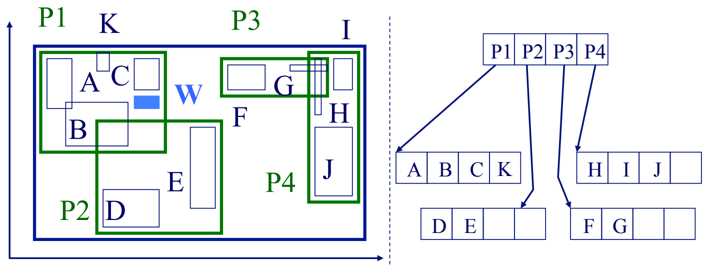
R-trees - insertion

- 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?

CMU SCS

R-trees - insertion

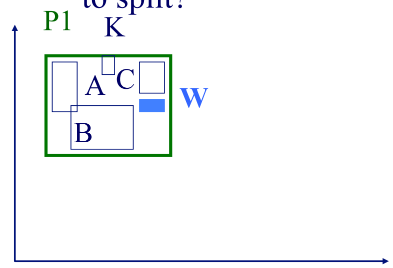
- eg., rectangle ‘W’



CMU SCS

R-trees - insertion

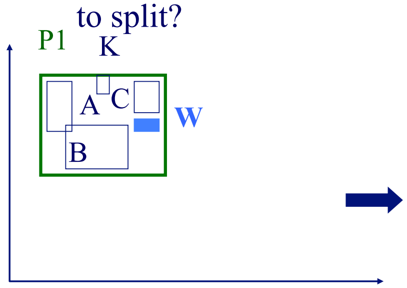
- eg., rectangle ‘W’ - focus on ‘P1’ - how to split?



CMU SCS

R-trees - insertion

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

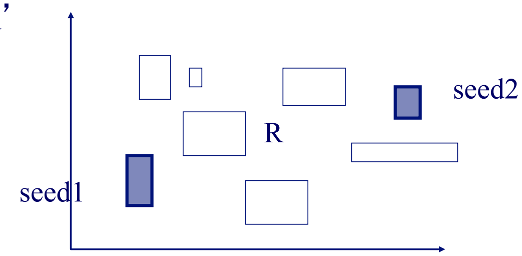


- (A1: plane sweep, until 50% of rectangles)
- A2: ‘linear’ split
- A3: quadratic split
- A4: exponential split

CMU SCS

R-trees - insertion & split

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



CMU SCS

R-trees - insertion & split

- pick two rectangles as 'seeds';
- assign each rectangle 'R' to the 'closest' 'seed'
- Q: how to measure 'closeness'?

 Copyright: C. Faloutsos (2017)

CMU SCS

R-trees - insertion & split

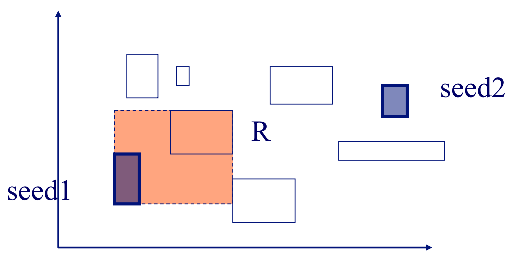
- pick two rectangles as 'seeds';
- assign each rectangle 'R' to the 'closest' 'seed'
- Q: how to measure 'closeness'?
- A: by increase of area (volume)

 Copyright: C. Faloutsos (2017)

CMU SCS

R-trees - insertion & split

- pick two rectangles as 'seeds';
- assign each rectangle 'R' to the 'closest' 'seed'

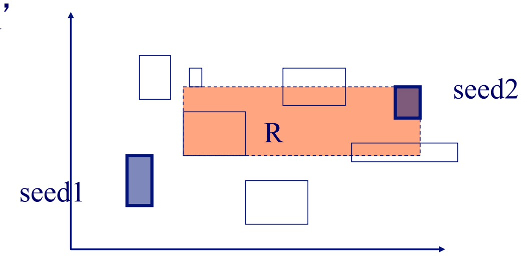


 Copyright: C. Faloutsos (2017)

CMU SCS

R-trees - insertion & split

- pick two rectangles as 'seeds';
- assign each rectangle 'R' to the 'closest' 'seed'



 Copyright: C. Faloutsos (2017)

CMU SCS

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!)

CMU SCS

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.

CMU SCS

R-trees - insertion - observations

- **many** more split algorithms exist (next!)

CMU SCS

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 - deletion

- delete rectangle
- if underflow
  - ??

CMU SCS

R-trees - deletion

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

CMU SCS

R-trees - deletion

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

CMU SCS

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

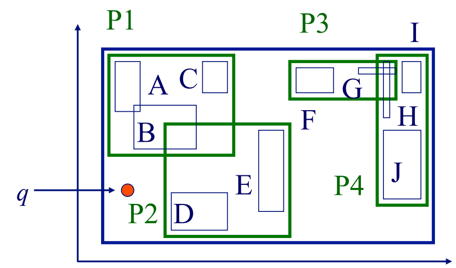
for each branch,

if its MBR intersects the query rectangle

apply range-search (or print out, if this is a leaf)

CMU SCS

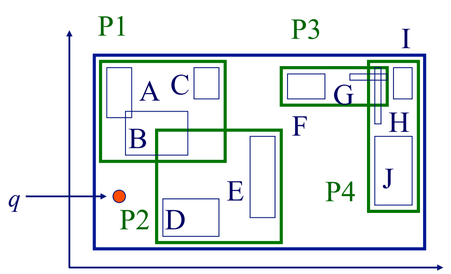
R-trees - nn search



CMU SCS

R-trees - nn search

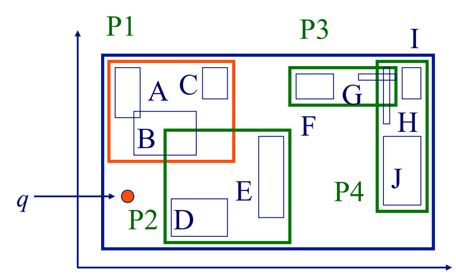
- Q: How? (find near neighbor; refine...)



CMU SCS

R-trees - nn search

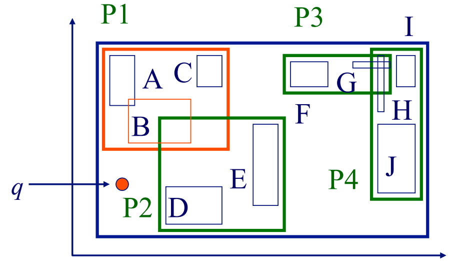
- A1: depth-first search; then, range query



CMU SCS

R-trees - nn search

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

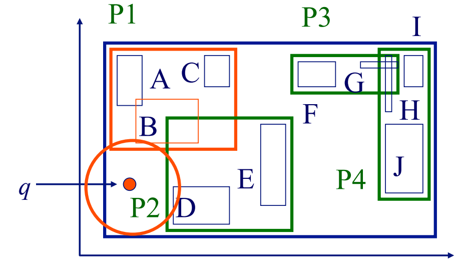


Copyright: C. Faloutsos (2017)

CMU SCS

R-trees - nn search

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



Copyright: C. Faloutsos (2017)

CMU SCS

R-trees - nn search

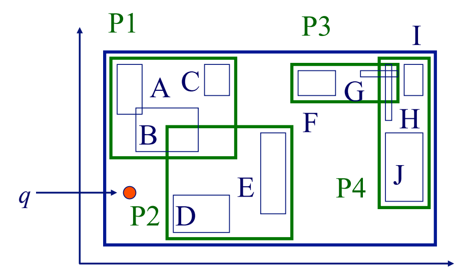
- A2: [Roussopoulos+, sigmod95]:
  - priority queue, with promising MBRs, and their best and worst-case distance
- main idea:

Copyright: C. Faloutsos (2017)

CMU SCS

R-trees - nn search

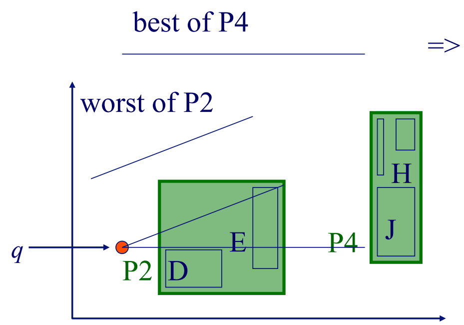
consider only P2 and P4, for illustration



Copyright: C. Faloutsos (2017)

CMU SCS

R-trees - nn search

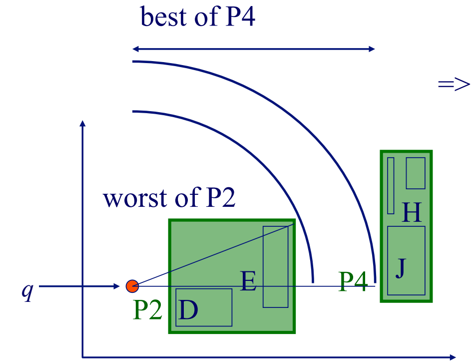


=> P4 is useless

for 1-nn

CMU SCS

R-trees - nn search



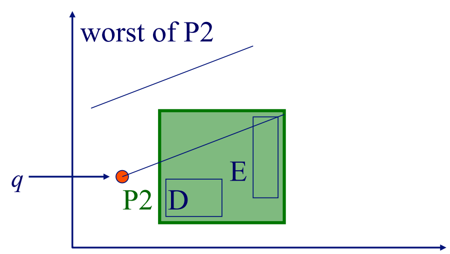
=> P4 is useless

for 1-nn

CMU SCS

R-trees - nn search

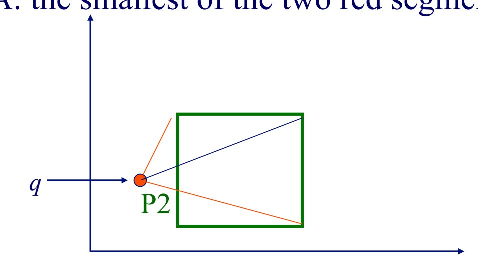
- what is really the worst of, say, P2?



CMU SCS

R-trees - nn search

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



CMU SCS

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)

15-826 Copyright: C. Faloutsos (2017) #57

CMU SCS

Indexing - more detailed outline

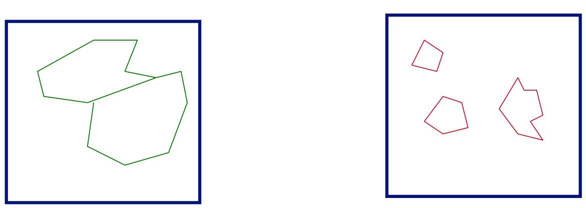
- R-trees
  - main idea; file structure
  - algorithms: insertion/split
  - deletion
  - search: range, nn, **spatial joins**
  - performance analysis
  - variations (packed; hilbert;...)

15-826 Copyright: C. Faloutsos (2017) #58

CMU SCS

R-trees - spatial joins

Spatial joins: find (quickly) all counties intersecting lakes

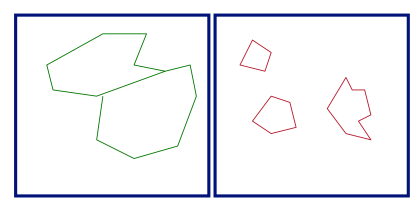


15-826 Copyright: C. Faloutsos (2017) #59

CMU SCS

R-trees - spatial joins

Spatial joins: find (quickly) all counties intersecting lakes

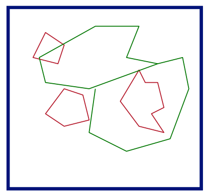


15-826 Copyright: C. Faloutsos (2017) #60

CMU SCS

R-trees - spatial joins

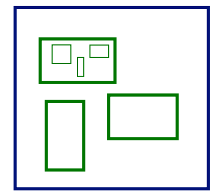
Spatial joins: find (quickly) all
counties intersecting lakes

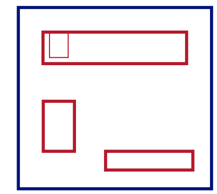


CMU SCS

R-trees - spatial joins

Assume that they are both organized in R-trees:

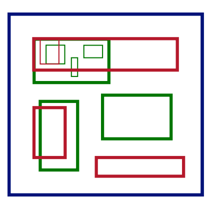




CMU SCS

R-trees - spatial joins

Assume that they are both organized in R-trees:



CMU SCS

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* * *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

CMU SCS

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

CMU SCS

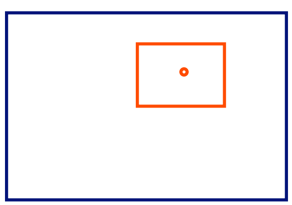
R-trees - performance analysis

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

CMU SCS

R-trees - performance analysis

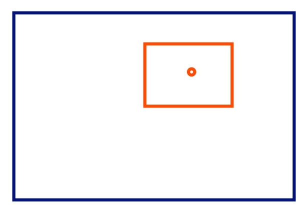
- How many disk accesses for range queries?
  - query distribution wrt location?
  - " " wrt size?



CMU SCS

R-trees - performance analysis

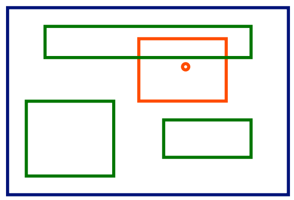
- How many disk accesses for range queries?
  - query distribution wrt location? uniform; (biased)
  - " " wrt size? uniform



CMU SCS

R-trees - performance analysis

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

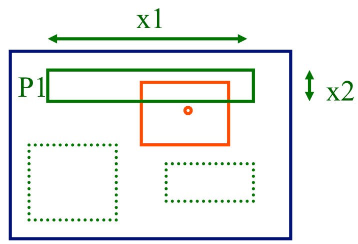


 Copyright: C. Faloutsos (2017)

CMU SCS

R-trees - performance analysis

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

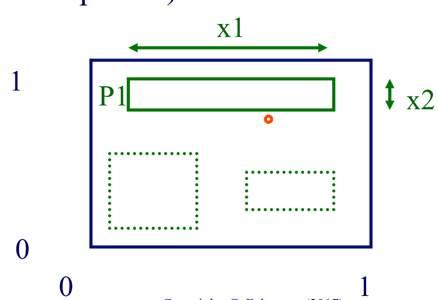


 Copyright: C. Faloutsos (2017)

CMU SCS

R-trees - performance analysis

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

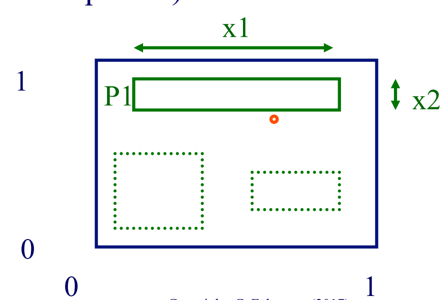


 Copyright: C. Faloutsos (2017)

CMU SCS

R-trees - performance analysis

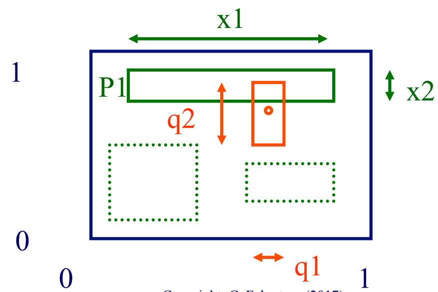
- How many times will P1 be retrieved (unif. POINT queries)? A: x1*x2



 Copyright: C. Faloutsos (2017)

R-trees - performance analysis

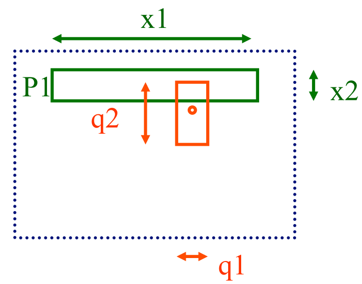
- How many times will P1 be retrieved (unif. queries of size q1xq2)?



 Copyright: C. Faloutsos (2017)

R-trees - performance analysis

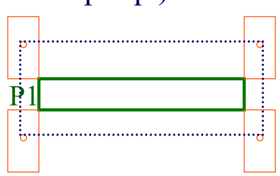
- How many times will P1 be retrieved (unif. queries of size q1xq2)?



 Copyright: C. Faloutsos (2017)

R-trees - performance analysis

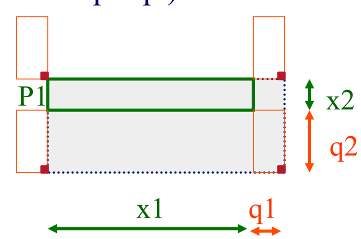
- How many times will P1 be retrieved (unif. queries of size q1xq2)?



 Copyright: C. Faloutsos (2017)

R-trees - performance analysis

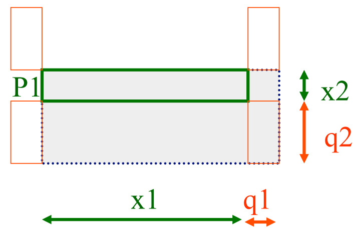
- How many times will P1 be retrieved (unif. queries of size q1xq2)?



 Copyright: C. Faloutsos (2017)

R-trees - performance analysis

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



R-trees - performance analysis

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

#DiskAccesses(q1,q2) =

$$\text{sum}\ (x_{i,1} + q1) * (x_{i,2} + q2)$$
$$= \text{sum}\ (x_{i,1} * x_{i,2}) +$$
$$q2 * \text{sum}\ (x_{i,1}) +$$
$$q1 * \text{sum}\ (x_{i,2})$$
$$q1 * q2 * N$$

R-trees - performance analysis

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

#DiskAccesses(q1,q2) =

$$\text{sum}\ (x_{i,1} + q1) * (x_{i,2} + q2)$$
$$= \text{sum}\ (x_{i,1} * x_{i,2}) + \quad \rightarrow \text{'volume'}$$
$$q2 * \text{sum}\ (x_{i,1}) + \quad \rightarrow \text{surface area}$$
$$q1 * \text{sum}\ (x_{i,2}) \quad \rightarrow \text{surface area}$$
$$q1 * q2 * N \quad \rightarrow \text{count}$$

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 >> 0): the count N matters

CMU SCS

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

CMU SCS

R-trees - performance analysis

Conclusions:

- splits should try to minimize area and perimeter
- ie., we want **few**, **small**, **square-like** parent MBRs
- rule of thumb: shoot for queries with q1=q2 = 0.1 (or =0.5 or so).

CMU SCS

R-trees - performance analysis

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

CMU SCS

R-trees - performance analysis

Range queries - how many disk accesses, if we just now that we have

- *N* points in *n*-d space?

A: ?

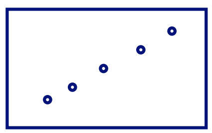
CMU SCS

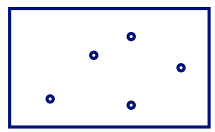
R-trees - performance analysis

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





CMU SCS

R-trees - performance analysis

What are obvious and/or realistic distributions?

CMU SCS

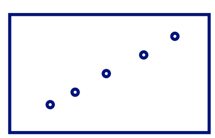
R-trees - performance analysis

What are obvious and/or realistic distributions?

A: uniform

A: Gaussian / mixture of Gaussians

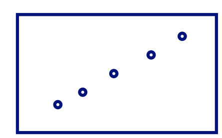
A: self-similar / fractal. Fractal dimension ~ intrinsic dimension



CMU SCS

R-trees - performance analysis

Formulas for range queries and k-nn queries: use fractal dimension [Kamel+, PODS94], [Korn+ ICDE2000] [Kriegel+, PODS97]



CMU SCS

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 - 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?

CMU SCS

R-trees - variations

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

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

CMU SCS

Popular

R-trees - variations

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

Norbert Beckmann

Hans Peter Kriegel →

Ralf Schneider

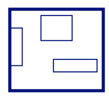
Bernhard Seeger →

CMU SCS

R-trees - variations

A: R*-trees [Beckmann+, 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?



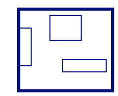
15-826 Copyright: C. Faloutsos (2017) #97

CMU SCS

R-trees - variations

A: R*-trees [Beckmann+, 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%



15-826 Copyright: C. Faloutsos (2017) #98

CMU SCS

R-trees - variations

Q: Other ways to defer splits?

15-826 Copyright: C. Faloutsos (2017) #99

CMU SCS

R-trees - variations

Q: Other ways to defer splits?

A: Push a few keys to the closest sibling node (closest = ??)

15-826 Copyright: C. Faloutsos (2017) #100

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.

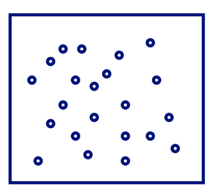
R-trees - variations

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

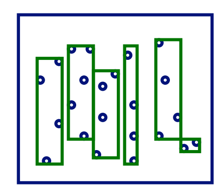
- what about static datasets (no ins/del/upd)?
- Q: Best way to pack points?



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'



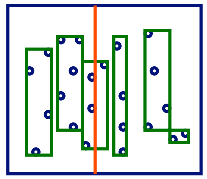
CMU SCS

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'



15-826 Copyright: C. Faloutsos (2017) #105

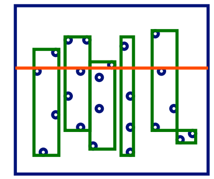
CMU SCS

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?

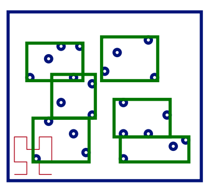


15-826 Copyright: C. Faloutsos (2017) #106

CMU SCS

R-trees - variations

- A: plane-sweep on HILBERT curve!

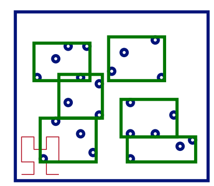


15-826 Copyright: C. Faloutsos (2017) #107

CMU SCS

R-trees - variations

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



15-826 Copyright: C. Faloutsos (2017) #108

CMU SCS

Details

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 - main idea
  - ➡ handling regions
  - performance/discusion
- what about other bounding shapes?

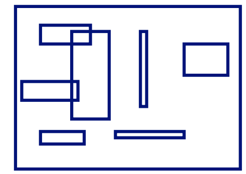
15-826 Copyright: C. Faloutsos (2017) #109

CMU SCS

Details

R-trees - variations

- What if we have regions, instead of points?
- I.e., how to impose a linear ordering ('h-value') on rectangles?



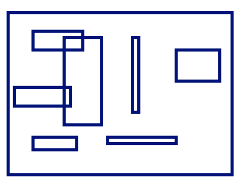
15-826 Copyright: C. Faloutsos (2017) #110

CMU SCS

Details

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: ...



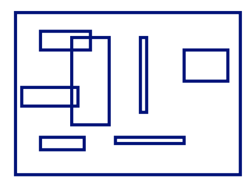
15-826 Copyright: C. Faloutsos (2017) #111

CMU SCS

Details

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: ...



15-826 Copyright: C. Faloutsos (2017) #112

CMU SCS

Details

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])

CMU SCS

Details

R-trees - variations

Guttman's R-trees sparked **much** follow-up work

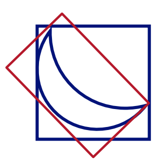
- can we do better splits?
- what about static datasets (no ins/del/upd)?
- what about other bounding shapes?

CMU SCS

Details

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)

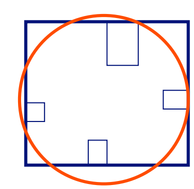


CMU SCS

Details

R-trees - variations

- A3: L-shapes; holes (hB-tree)
- A4: TV-trees [Lin+, VLDB-Journal 1994]
- A5: SR-trees [Katayama+, SIGMOD97] (used in Informedia)



CMU SCS

R-trees - conclusions

- 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`
- R* variation is popular

CMU SCS

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
- Guttman, A. (June 1984). *R-Trees: A Dynamic Index Structure for Spatial Searching*. Proc. ACM SIGMOD, Boston, Mass.

CMU SCS

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
- Ibrahim Kamel and Christos Faloutsos, *Hilbert R-tree: An improved R-tree using fractals* VLDB, Santiago, Chile, Sept. 12-15, 1994, pp. 500-509.
- Lin, K.-I., H. V. Jagadish, et al. (Oct. 1994). "The TV-tree - An Index Structure for High-dimensional Data." VLDB Journal 3: 517-542.

CMU SCS

References, cont'd

- Pagel, B., H. Six, et al. (May 1993). *Towards an Analysis of Range Query Performance*. Proc. of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Washington, D.C.
- Robinson, J. T. (1981). The k-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes. Proc. ACM SIGMOD.
- Roussopoulos, N., S. Kelley, et al. (May 1995). Nearest Neighbor Queries. Proc. of ACM-SIGMOD, San Jose, CA.

CMU SCS

Other resources

- Code, papers, datasets etc:
  www.rtreeportal.org/
- Java applets and more info:
  donar.umiacs.umd.edu/quadtree/points/rtrees.html

 Copyright: C. Faloutsos (2017)