Carnegie Mellon

15-826: Multimedia Databases and Data Mining

Lecture #6: Spatial Access Methods

Part III: R-trees

C. Faloutsos

Carnegie Mellon

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.

Carnegie Mellon

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

15-826 Copyright: C. Faloutsos (2019) #3

Carnegie Mellon

Outline

Goal: 'Find similar / interesting things'

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

15-826 Copyright: C. Faloutsos (2019) #4

Carnegie Mellon

Indexing - Detailed outline

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

Carnegie Mellon

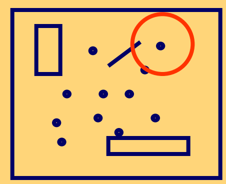
Indexing - more detailed outline

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

Carnegie Mellon

Spatial Access Methods - problem

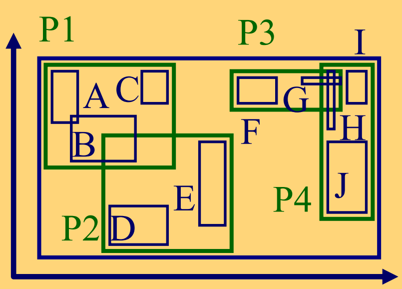
- Given a collection of geometric objects (points, lines, polygons, ...)
- Find cities within 100mi from Pittsburgh



Carnegie Mellon

Solution#2: R-trees

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



Carnegie Mellon

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

Carnegie Mellon

R-trees

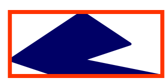
- [Guttman 84] Main idea: allow parents to overlap!

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

Carnegie Mellon

R-trees

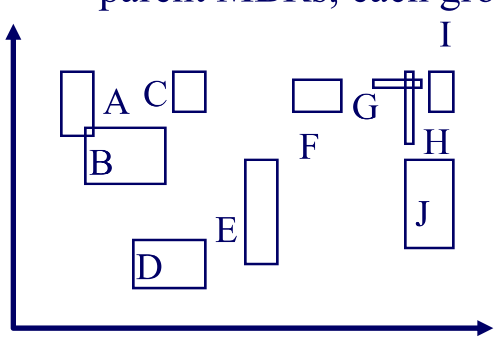
- [Guttman 84] Main idea: allow parents to overlap!
  - => guaranteed 50% utilization
  - => easier insertion/split algorithms.
  - (only deal with Minimum Bounding Rectangles - **MBR**s)



Carnegie Mellon

R-trees

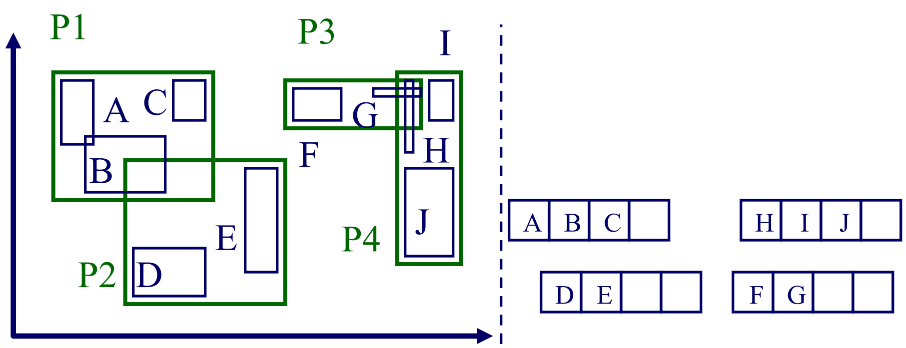
- eg., w/ fanout 4: group nearby rectangles to parent MBRs; each group -> disk page



Carnegie Mellon

R-trees

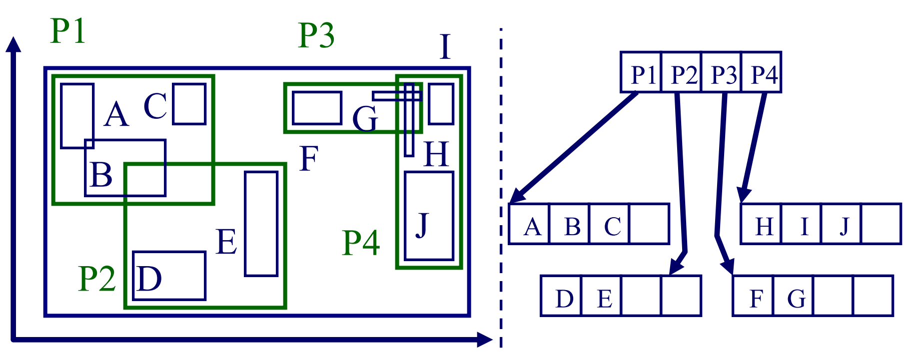
- eg., w/ fanout 4:



Carnegie Mellon

R-trees

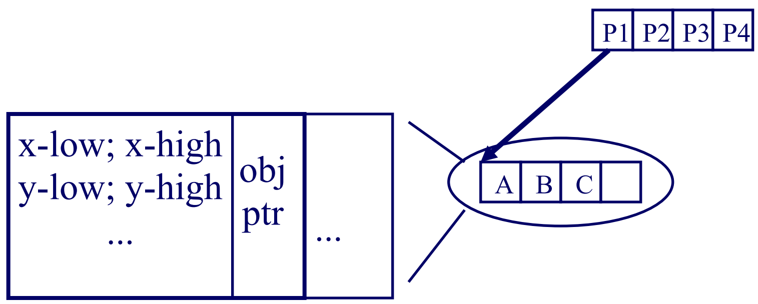
- eg., w/ fanout 4:



Carnegie Mellon

R-trees - format of nodes

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

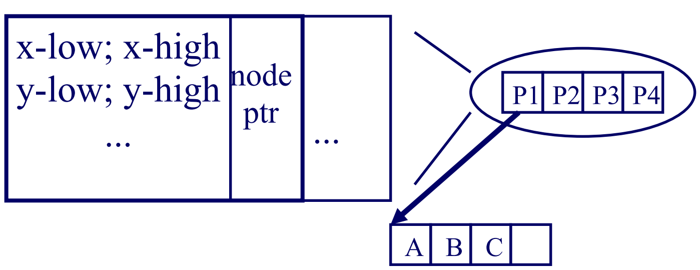


15-826 Copyright: C. Faloutsos (2019) #15

Carnegie Mellon

R-trees - format of nodes

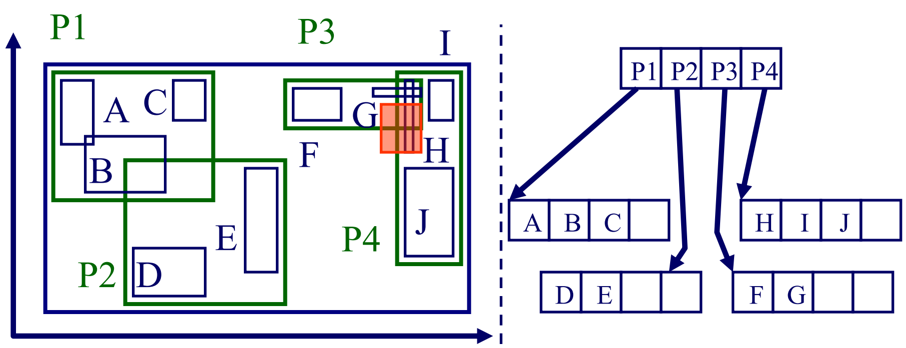
- {(MBR; node-ptr)} for non-leaf nodes



15-826 Copyright: C. Faloutsos (2019) #16

Carnegie Mellon

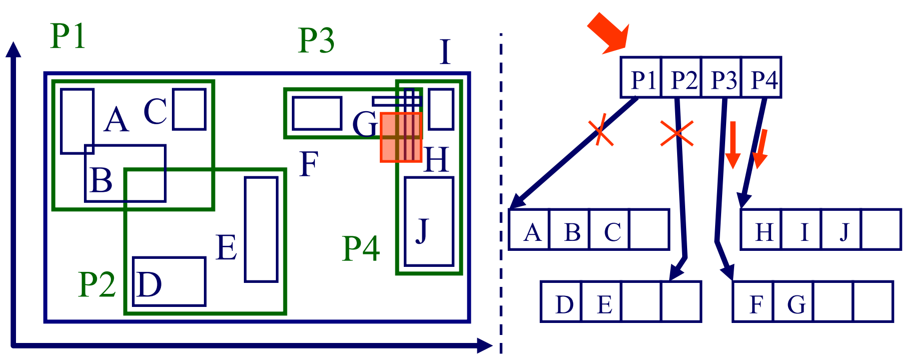
R-trees - range search?



15-826 Copyright: C. Faloutsos (2019) #17

Carnegie Mellon

R-trees - range search?



15-826 Copyright: C. Faloutsos (2019) #18

Carnegie Mellon

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.)

Carnegie Mellon

R-trees - range search

Observations - cont'd

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

Carnegie Mellon

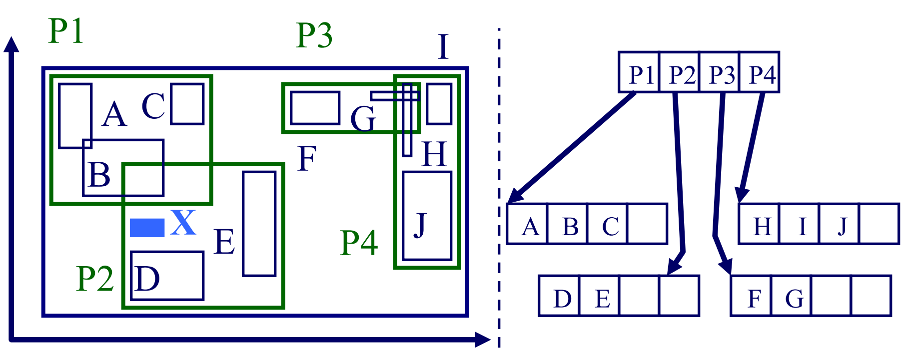
Indexing - more detailed outline

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

Carnegie Mellon

R-trees - insertion

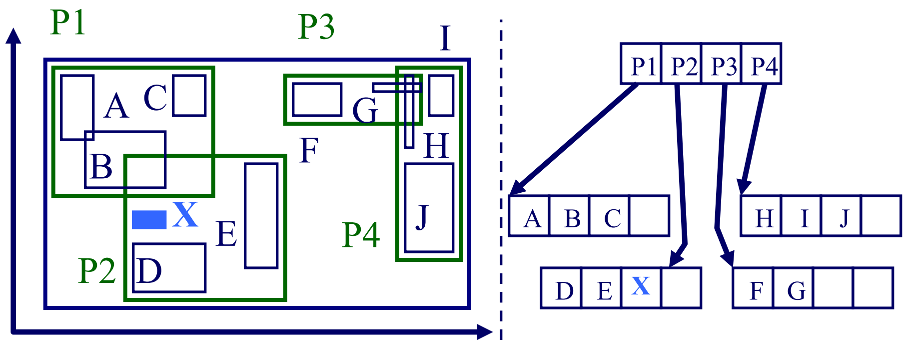
- eg., rectangle 'X'



Carnegie Mellon

R-trees - insertion

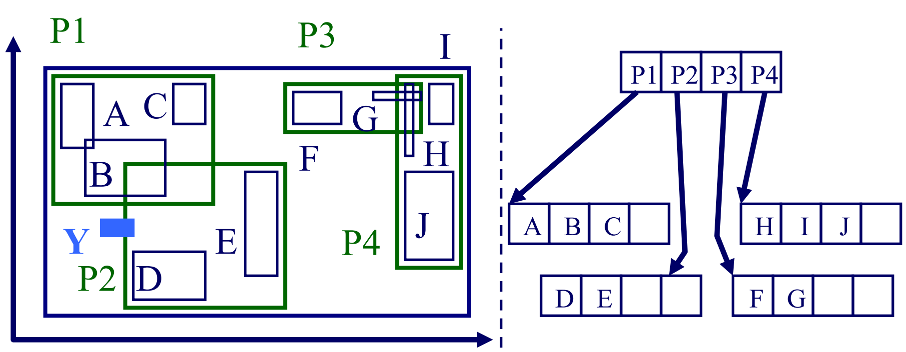
- eg., rectangle 'X'



Carnegie Mellon

R-trees - insertion

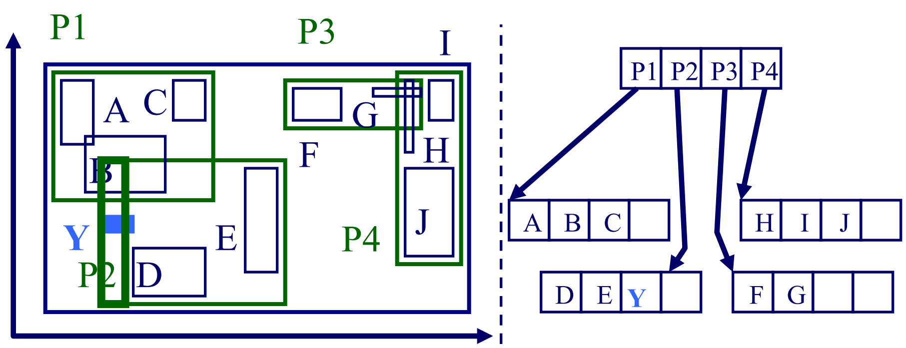
- eg., rectangle 'Y'



Carnegie Mellon

R-trees - insertion

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



Carnegie Mellon

R-trees - insertion

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

Carnegie Mellon

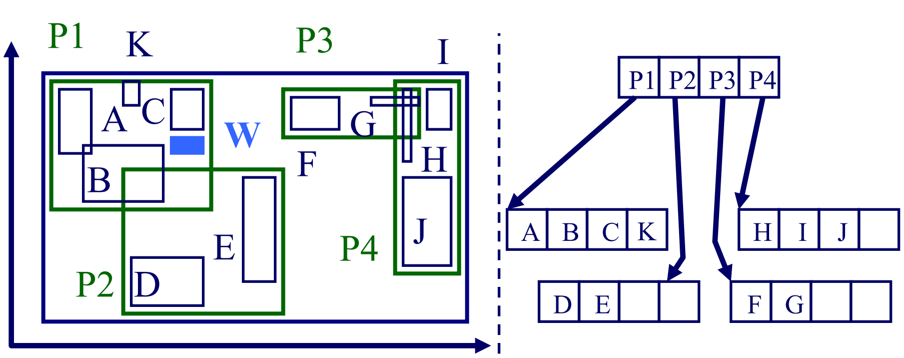
R-trees - insertion

- 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?

Carnegie Mellon

R-trees - insertion

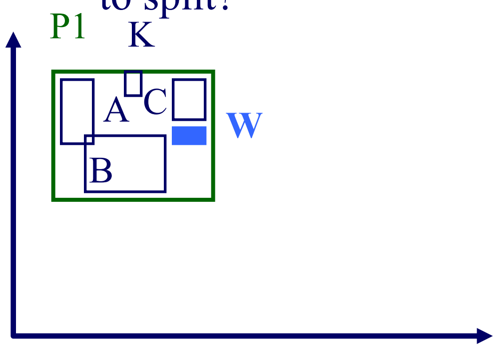
- eg., rectangle 'W'



Carnegie Mellon

R-trees - insertion

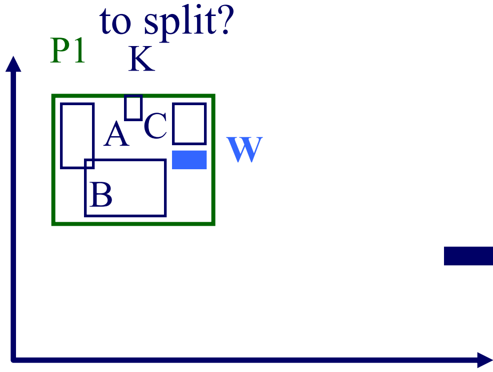
- eg., rectangle ‘W’ - focus on ‘P1’ - how to split?



Carnegie Mellon

R-trees - insertion

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

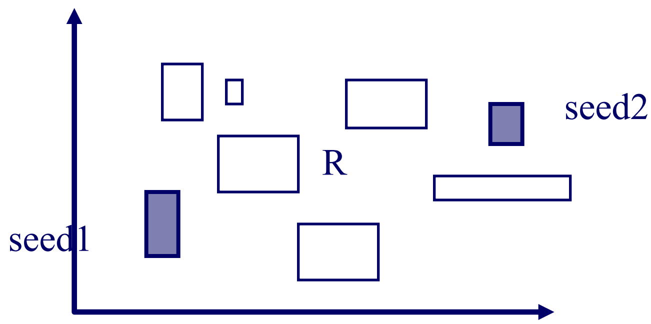


- (A1: plane sweep, until 50% of rectangles)
- A2: ‘linear’ split
- A3: quadratic split
- A4: exponential split

Carnegie Mellon

R-trees - insertion & split

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



Carnegie Mellon

R-trees - insertion & split

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

Carnegie Mellon

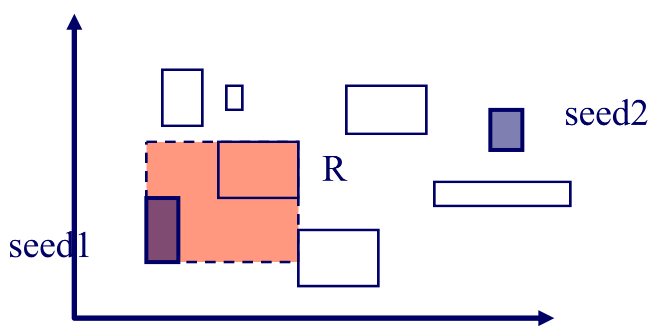
R-trees - insertion & split

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

Carnegie Mellon

R-trees - insertion & split

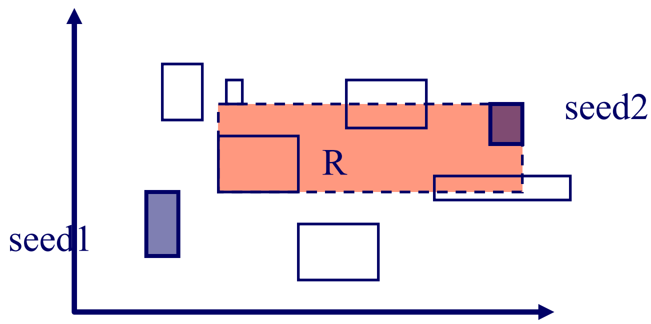
- pick two rectangles as 'seeds';
- assign each rectangle 'R' to the 'closest' 'seed'



Carnegie Mellon

R-trees - insertion & split

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



Carnegie Mellon

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!)

Carnegie Mellon

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.

Carnegie Mellon

R-trees - insertion - observations

- **many** more split algorithms exist (see refs)

Carnegie Mellon

Indexing - more detailed outline

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

Carnegie Mellon

R-trees - deletion

- delete rectangle
- if underflow
  - ??

Carnegie Mellon

R-trees - deletion

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

Carnegie Mellon

Indexing - more detailed outline

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

Carnegie Mellon

R-trees - range search

pseudocode:

check the root

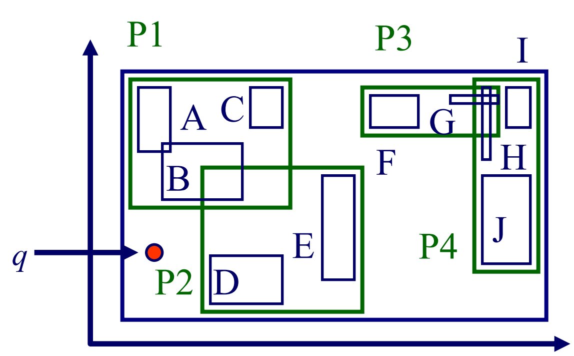
for each branch,

if its MBR intersects the query rectangle

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

Carnegie Mellon

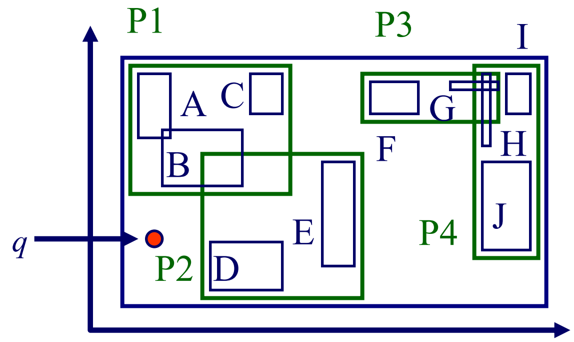
R-trees - nn search



Carnegie Mellon

R-trees - nn search

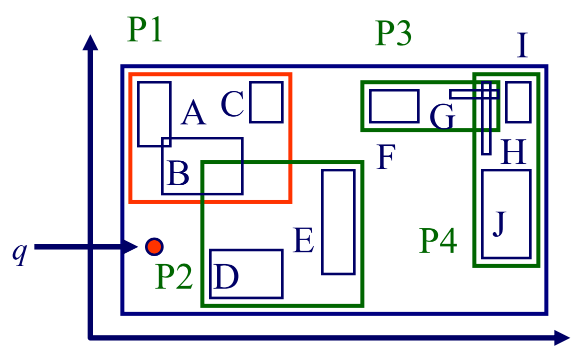
- Q: How? (find near neighbor; refine...)



Carnegie Mellon

R-trees - nn search

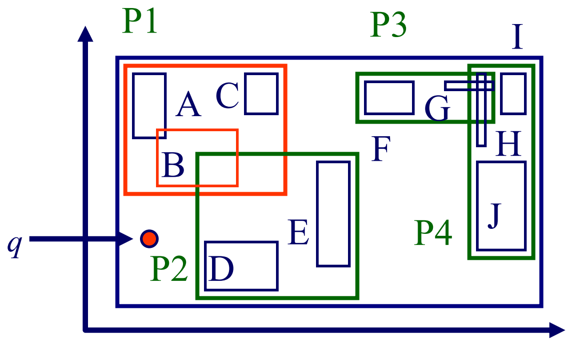
- A1: depth-first search; then, range query



Carnegie Mellon

R-trees - nn search

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

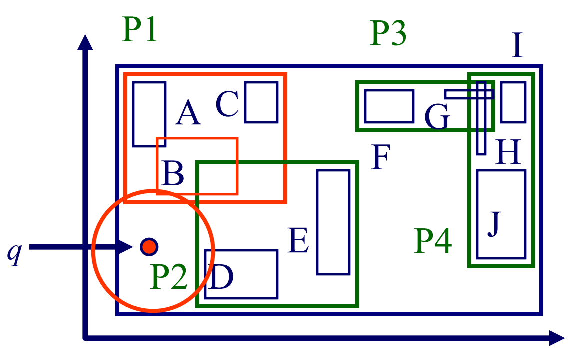


15-826 Copyright: C. Faloutsos (2019) #47

Carnegie Mellon

R-trees - nn search

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



15-826 Copyright: C. Faloutsos (2019) #48

Carnegie Mellon

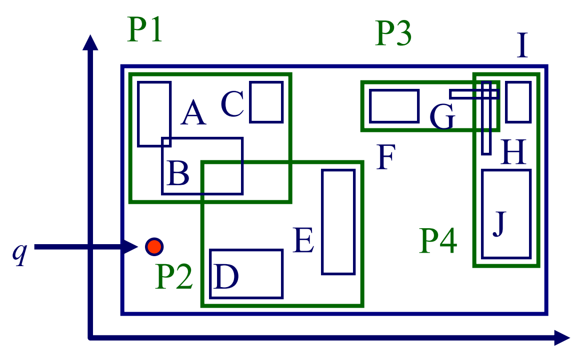
R-trees - nn search

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

Carnegie Mellon

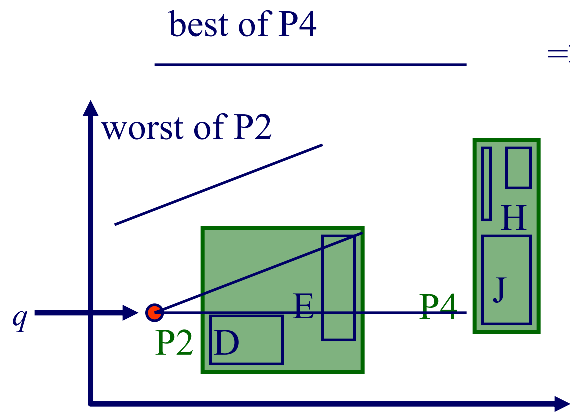
R-trees - nn search

consider only P2 and P4, for illustration



Carnegie Mellon

R-trees - nn search

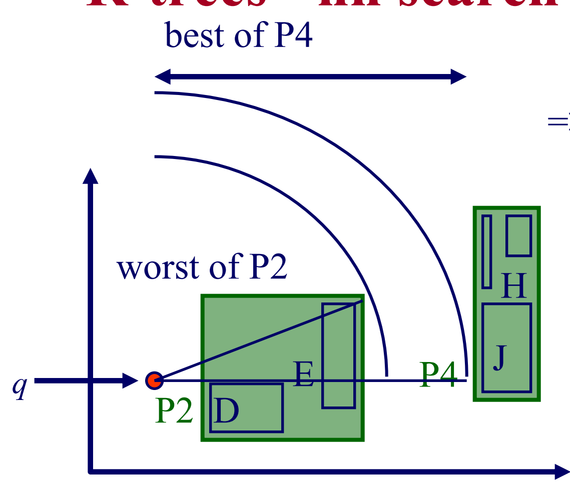


=> P4 is useless for 1-nn

15-826 Copyright: C. Faloutsos (2019) #51

Carnegie Mellon

R-trees - nn search



=> P4 is useless for 1-nn

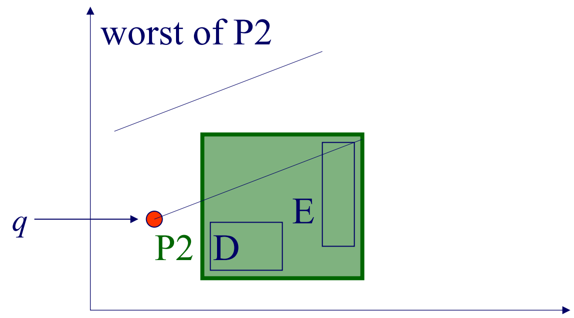
15-826 Copyright: C. Faloutsos (2019) #52

CMU SCS

DETAILS

R-trees - nn search

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

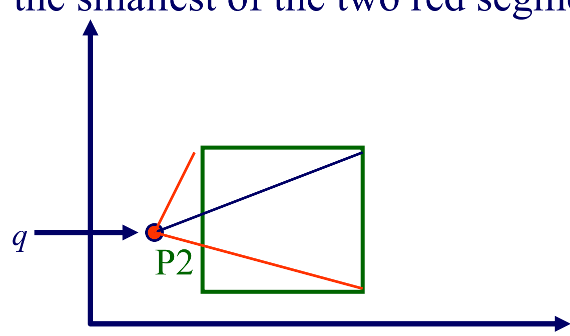


Carnegie Mellon

DETAILS

R-trees - nn search

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



Carnegie Mellon

Indexing - more detailed outline

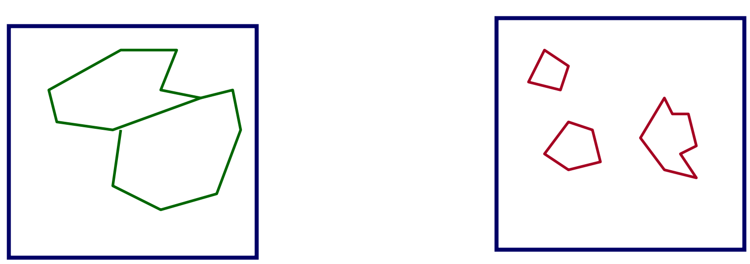
- R-trees
  - main idea; file structure
  - algorithms: insertion/split
  - deletion
  - search: range, nn, **spatial joins**
  - performance analysis
  - variations (packed; hilbert;...)

Carnegie Mellon

R-trees - spatial joins

Spatial joins: find (quickly) all

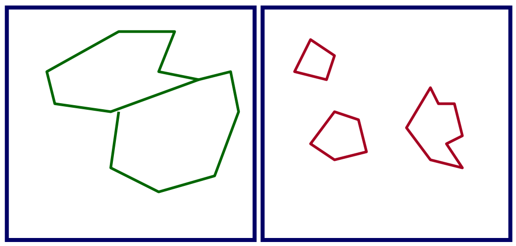
counties intersecting lakes



Carnegie Mellon

R-trees - spatial joins

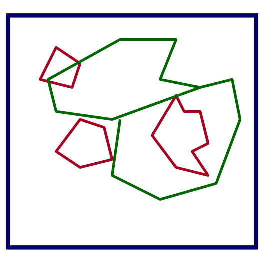
Spatial joins: find (quickly) all
counties intersecting lakes



Carnegie Mellon

R-trees - spatial joins

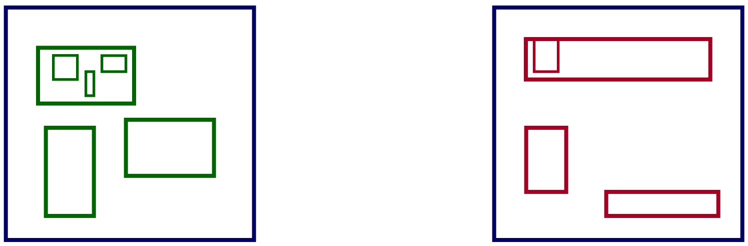
Spatial joins: find (quickly) all
counties intersecting lakes



Carnegie Mellon

R-trees - spatial joins

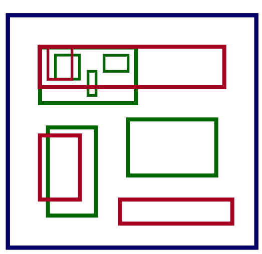
Assume that they are both organized in R-trees:



Carnegie Mellon

R-trees - spatial joins

Assume that they are both organized in R-trees:



Carnegie Mellon

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)

Carnegie Mellon

DETAILS

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.)

Carnegie Mellon

Indexing - more detailed outline

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

Carnegie Mellon

R-trees - performance analysis

- How many disk (=node) accesses we'll need for
  - range
  - nn
  - spatial joins
- why does it matter?

Carnegie Mellon

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

15-826 Copyright: C. Faloutsos (2019) #65

Carnegie Mellon

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

15-826 Copyright: C. Faloutsos (2019) #66

Carnegie Mellon

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?

Carnegie Mellon

R-trees - performance analysis

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

#DiskAccesses(q1,q2) =

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

$$= \text{sum} (x_{i,1} * x_{i,2}) +$$ → 'volume'

$$q2 * \text{sum} (x_{i,1}) +$$ → surface area

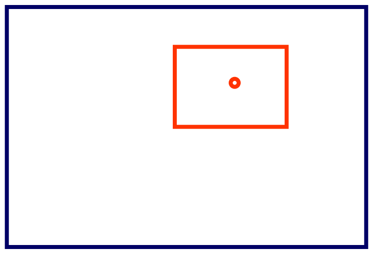
$$q1 * \text{sum} (x_{i,2})$$ → surface area

$$q1 * q2 * N$$ → count

Carnegie Mellon

R-trees - performance analysis

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

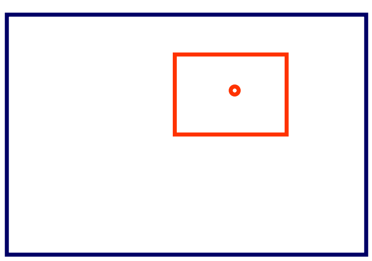


Carnegie Mellon

Proof

R-trees - performance analysis

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

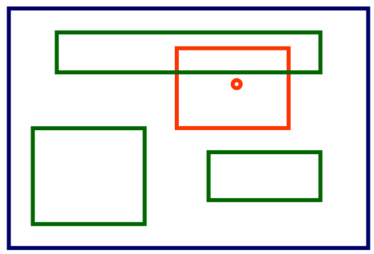


Carnegie Mellon

Proof

R-trees - performance analysis

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

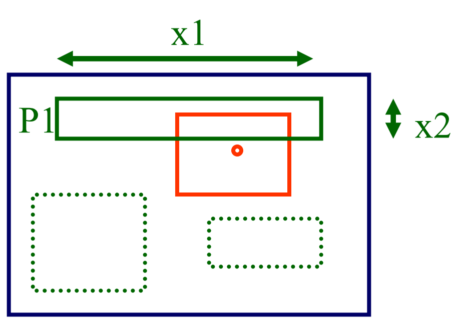


Carnegie Mellon

Proof

R-trees - performance analysis

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

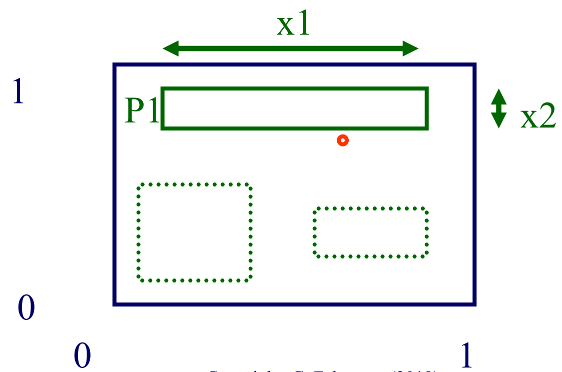


Carnegie Mellon

Proof

R-trees - performance analysis

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



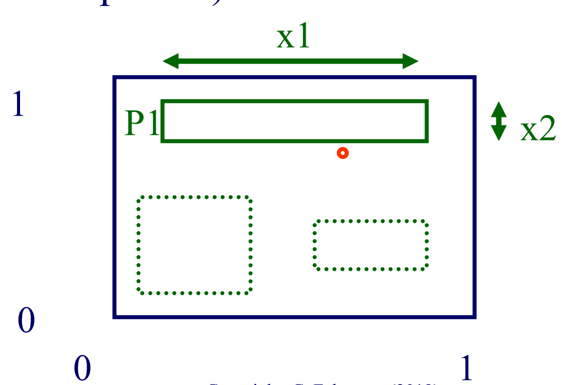
15-826 Copyright: C. Faloutsos (2019) #73

Carnegie Mellon

Proof

R-trees - performance analysis

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



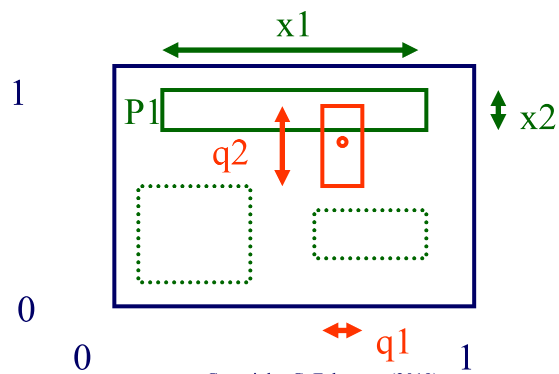
15-826 Copyright: C. Faloutsos (2019) #74

Carnegie Mellon

Proof

R-trees - performance analysis

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

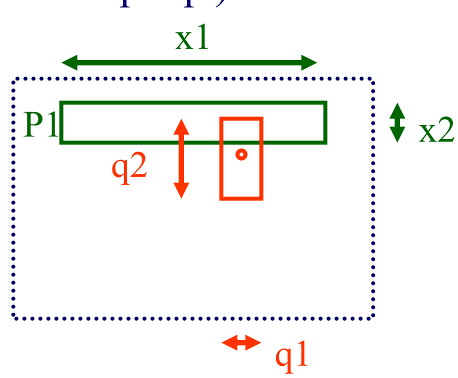


Carnegie Mellon

Proof

R-trees - performance analysis

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

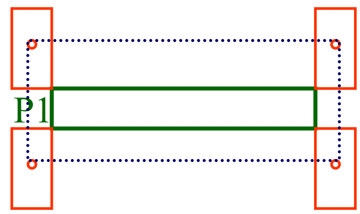


Carnegie Mellon

Proof

R-trees - performance analysis

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

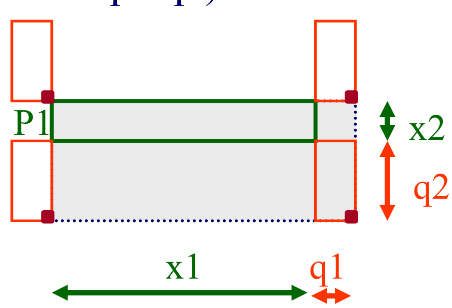


Carnegie Mellon

Proof

R-trees - performance analysis

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

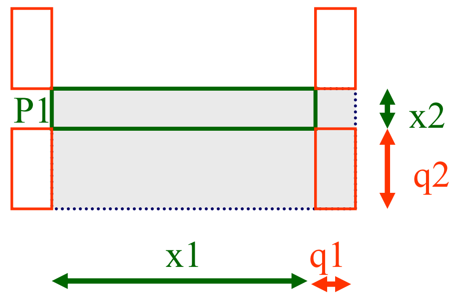


Carnegie Mellon

Proof

R-trees - performance analysis

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



15-826 Copyright: C. Faloutsos (2019) #79

Carnegie Mellon

Proof

R-trees - performance analysis

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

  #DiskAccesses(q1,q2) =

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

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

  $q2 * \text{sum} (x_{i,1}) +$

  $q1 * \text{sum} (x_{i,2})$

  $q1 * q2 * N$

15-826 Copyright: C. Faloutsos (2019) #80

Carnegie Mellon

R-trees - performance analysis

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

#DiskAccesses(q1,q2) =

sum ($x_{i,1}$ + q1) * ($x_{i,2}$ + q2)

= sum ($x_{i,1}$ * $x_{i,2}$) + → 'volume'

q2 * sum ($x_{i,1}$) + → surface area

q1* sum ($x_{i,2}$) → surface area

q1* q2 * N → count

15-826 Copyright: C. Faloutsos (2019) #81

Carnegie Mellon

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

15-826 Copyright: C. Faloutsos (2019) #82

Carnegie Mellon

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

15-826 Copyright: C. Faloutsos (2019) #83

Carnegie Mellon

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).

15-826 Copyright: C. Faloutsos (2019) #84

Carnegie Mellon

DETAILS

Indexing - more detailed outline

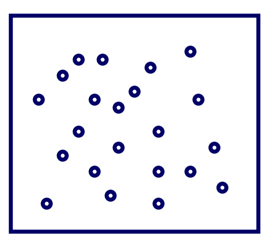
- R-trees
  - main idea; file structure
  - algorithms: insertion/split
  - deletion
  - search: range, nn, spatial joins
  - performance analysis
  - variations (packed; hilbert;...)

Carnegie Mellon

DETAILS

R-trees - variations

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

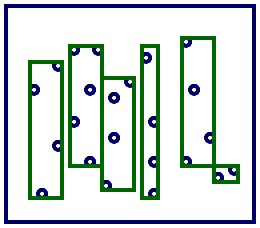


Carnegie Mellon

DETAILS

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'

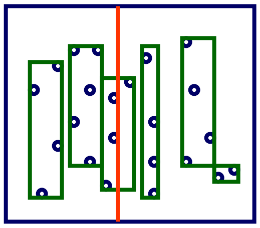


Carnegie Mellon

DETAILS

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'

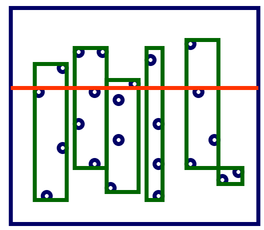


Carnegie Mellon

DETAILS

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?

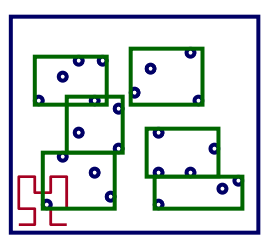


Carnegie Mellon

DETAILS

R-trees - variations

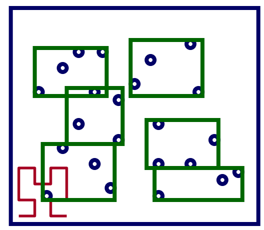
- A: plane-sweep on HILBERT curve!



DETAILS

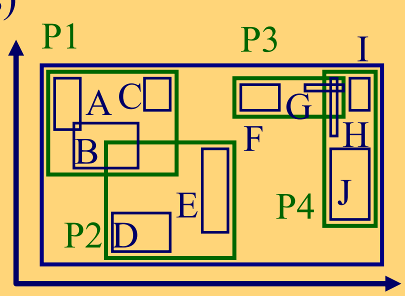
R-trees - variations

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

Carnegie Mellon

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

Carnegie Mellon

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.
- Roussopoulos, N., S. Kelley, et al. (May 1995). Nearest Neighbor Queries. Proc. of ACM-SIGMOD, San Jose, CA.

Carnegie Mellon

Other resources

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