15-721 DB Sys. Design & Impl.

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
- ...

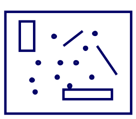
3) Transactions: locking, recovery

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;...)

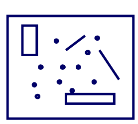
Spatial Access Methods - problem

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



Spatial Access Methods - problem

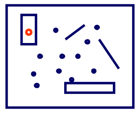
- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer
  - point queries
  - range queries
  - k-nn queries
  - spatial joins (‘all pairs’ queries)



CMU SCS

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)

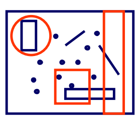


15-721 C. Faloutsos 7

CMU SCS

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)

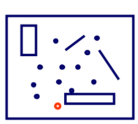


15-721 C. Faloutsos 8

CMU SCS

Spatial Access Methods - problem

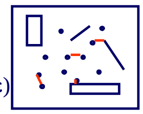
- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer
  - point queries
  - range queries
  - k-nn queries
  - spatial joins (‘all pairs’ queries)



15-721 C. Faloutsos 9

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)



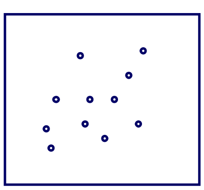
SAMs - motivation

- Q: applications?

SAMs - motivation

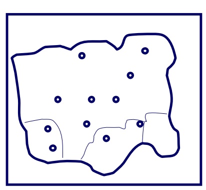
traditional DB

age

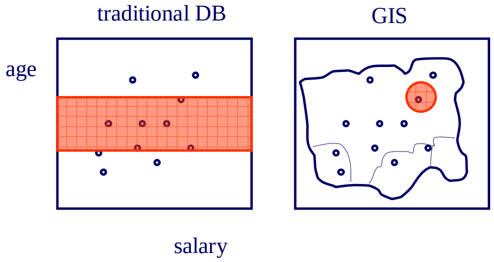


salary

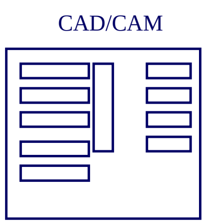
GIS



SAMs - motivation

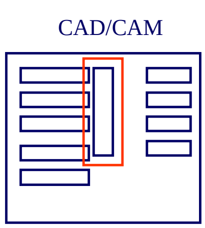


SAMs - motivation



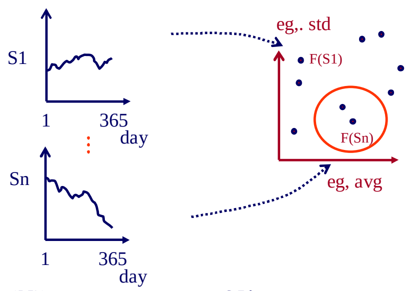
find elements
too close
to each other

SAMs - motivation



CMU SCS

SAMs - motivation

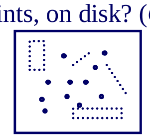


CMU SCS

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)



CMU SCS

Outline

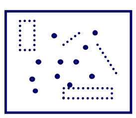
- R-trees
  - – Problem definition
  - – main idea; file structure
  - – algorithms: insertion/split
  - – deletion
  - – search: range, nn, spatial joins
  - – performance analysis
  - – variations (packed; hilbert;...)

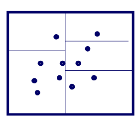
R-trees

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

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

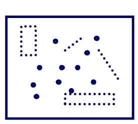
- [Robinson, 81]: if *f* is the fanout, split point-set in *f* parts; and so on, recursively

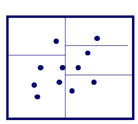




(first attempt: k-d-B-trees)

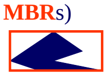
- But: insertions/deletions are tricky (splits may propagate downwards **and** upwards)
- no guarantee on space utilization





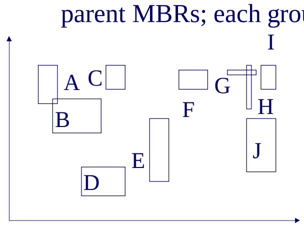
R-trees

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



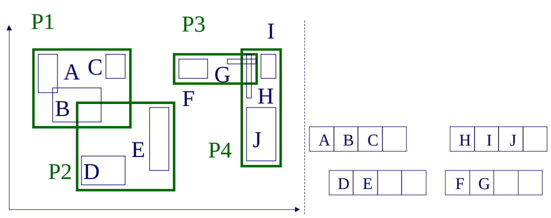
R-trees

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



R-trees

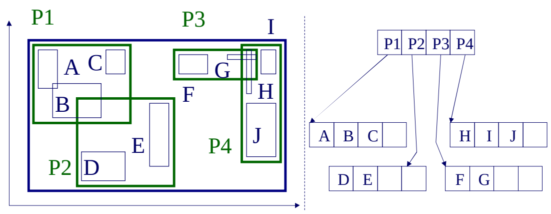
- eg., w/ fanout 4:



CMU SCS

R-trees

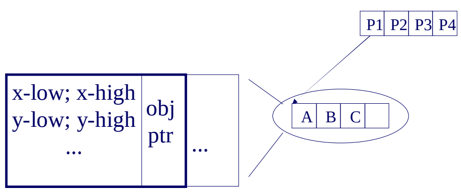
- eg., w/ fanout 4:



CMU SCS

R-trees - format of nodes

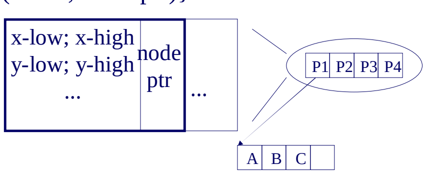
- {(MBR; obj-ptr)} for leaf nodes



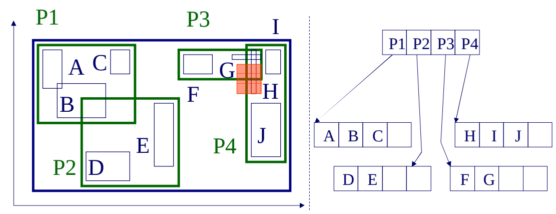
CMU SCS

R-trees - format of nodes

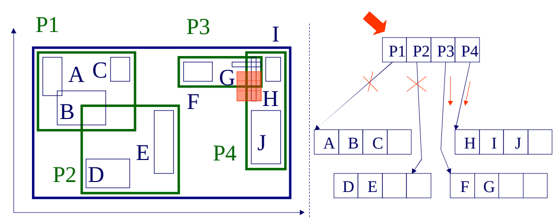
- {(MBR; node-ptr)} for non-leaf nodes



R-trees - range search?



R-trees - range search?



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

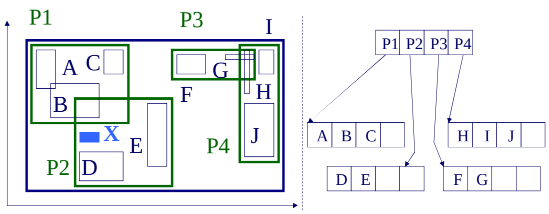
- a point query may follow multiple branches.
- everything works for **any** dimensionality

Outline

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

R-trees - insertion

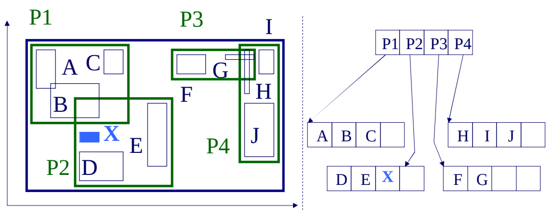
- eg., rectangle 'X'



CMU SCS

R-trees - insertion

- eg., rectangle ‘X’

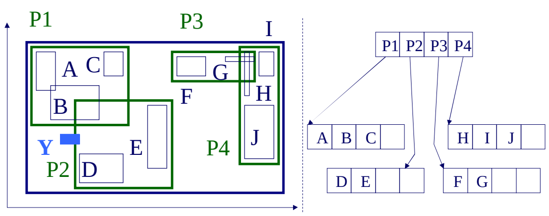


15-721 C. Faloutsos 34

CMU SCS

R-trees - insertion

- eg., rectangle ‘Y’

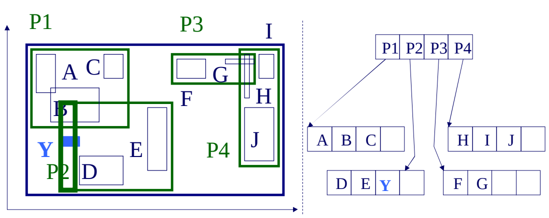


15-721 C. Faloutsos 35

CMU SCS

R-trees - insertion

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



15-721 C. Faloutsos 36

CMU SCS

R-trees - insertion

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

CMU SCS

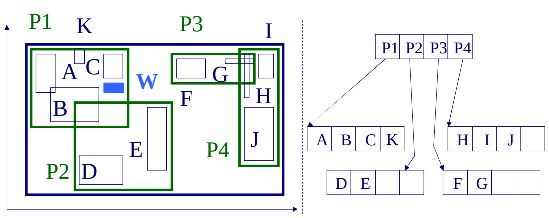
R-trees - insertion

- 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

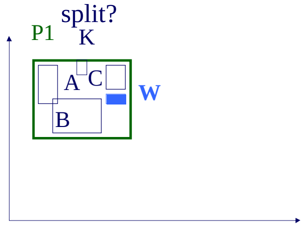
R-trees - insertion

- eg., rectangle 'W'



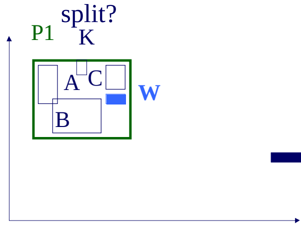
R-trees - insertion

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



R-trees - insertion

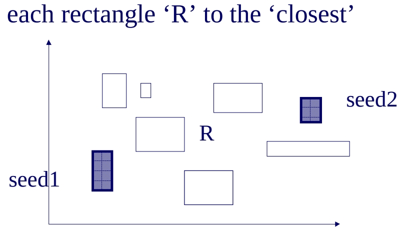
- eg., rectangle ‘W’ - focus on ‘P1’ - how to split?



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

R-trees - insertion & split

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



R-trees - insertion & split

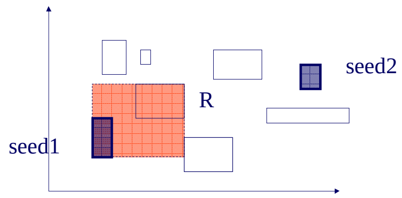
- pick two rectangles as ‘seeds’;
- assign each rectangle ‘R’ to the ‘closest’ ‘seed’
- Q: how to measure ‘closeness’?

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

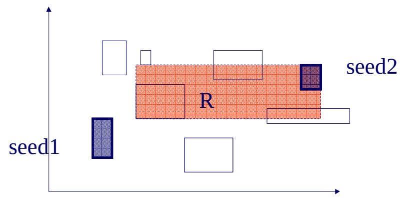
- pick two rectangles as ‘seeds’;
- assign each rectangle ‘R’ to the ‘closest’ ‘seed’



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’
- 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!)

15-721 C. Faloutsos 49

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;...)

15-721 C. Faloutsos 50

CMU SCS

R-trees - deletion

- delete rectangle
- if underflow
  - ??

15-721 C. Faloutsos 51

CMU SCS

R-trees - deletion

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

15-721 C. Faloutsos 52

CMU SCS

R-trees - deletion

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

15-721 C. Faloutsos 53

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;...)

15-721 C. Faloutsos 54

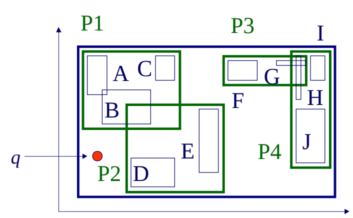
CMU SCS

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

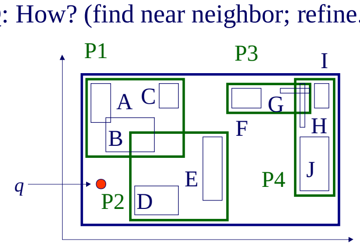
R-trees - nn search



CMU SCS

R-trees - nn search

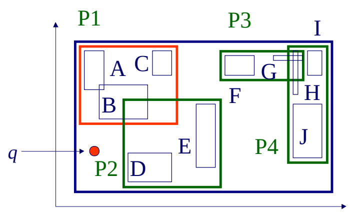
- Q: How? (find near neighbor; refine...)



CMU SCS

R-trees - nn search

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

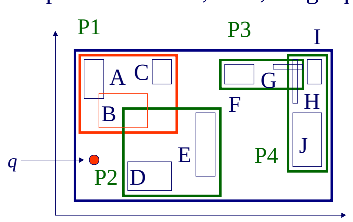


15-721 C. Faloutsos 58

CMU SCS

R-trees - nn search

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

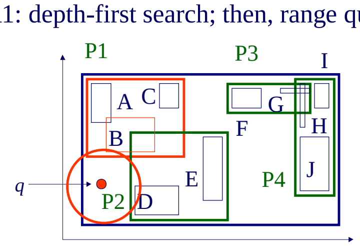


15-721 C. Faloutsos 59

CMU SCS

R-trees - nn search

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



15-721 C. Faloutsos 60

CMU SCS

R-trees - nn search

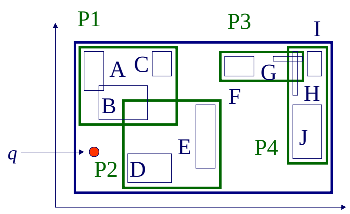
- A2: [Roussopoulos+, sigmod95]:
  - priority queue, with promising MBRs, and their best and worst-case distance
- main idea:

15-721 C. Faloutsos 61

CMU SCS

R-trees - nn search

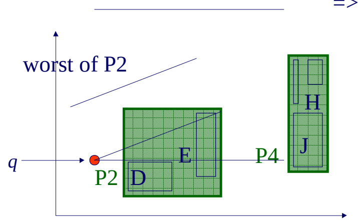
consider only P2 and P4, for illustration



15-721 C. Faloutsos 62

CMU SCS

R-trees - nn search

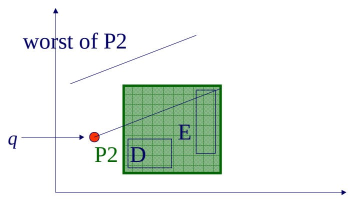


15-721 C. Faloutsos 63

CMU SCS

R-trees - nn search

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

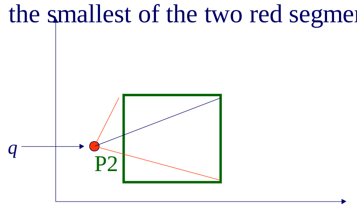


15-721 C. Faloutsos 64

CMU SCS

R-trees - nn search

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



15-721 C. Faloutsos 65

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)

15-721 C. Faloutsos 66

CMU SCS

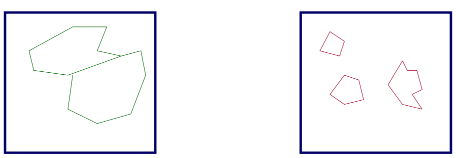
Indexing - more detailed outline

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

CMU SCS

R-trees - spatial joins

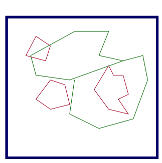
Spatial joins: find (quickly) all
counties intersecting lakes



CMU SCS

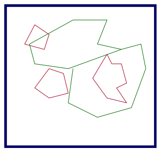
R-trees - spatial joins

Spatial joins: find (quickly) all
counties intersecting lakes



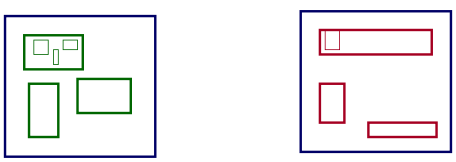
R-trees - spatial joins

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)

CMU SCS

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.)

15-721 C. Faloutsos 73

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;...)

15-721 C. Faloutsos 74

CMU SCS

Advanced - skip

R-trees - performance analysis

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

15-721 C. Faloutsos 75

CMU SCS

Advanced - skip

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

15-721 C. Faloutsos 76

CMU SCS

Advanced - skip

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?

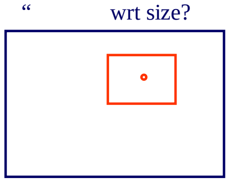
15-721 C. Faloutsos 77

CMU SCS

Advanced - skip

R-trees - performance analysis

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



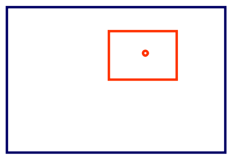
15-721 C. Faloutsos 78

CMU SCS

Advanced - skip

R-trees - performance analysis

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



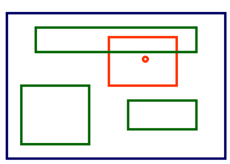
15-721 C. Faloutsos 79

CMU SCS

Advanced - skip

R-trees - performance analysis

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



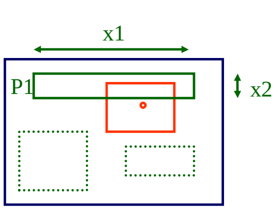
15-721 C. Faloutsos 80

CMU SCS

Advanced - skip

R-trees - performance analysis

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

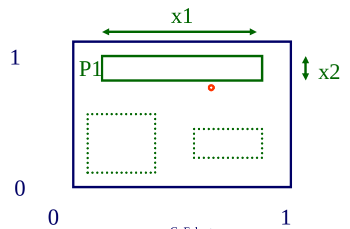


15-721 C. Faloutsos 81

Advanced - skip

R-trees - performance analysis

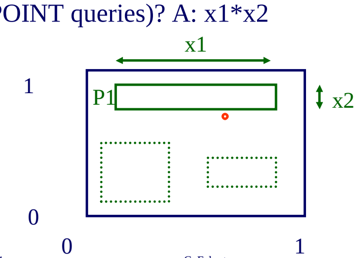
- How many times will P1 be retrieved (unif. POINT queries)?



Advanced - skip

R-trees - performance analysis

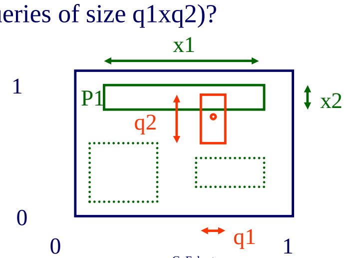
- How many times will P1 be retrieved (unif. POINT queries)? A: x1*x2



Advanced - skip

R-trees - performance analysis

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

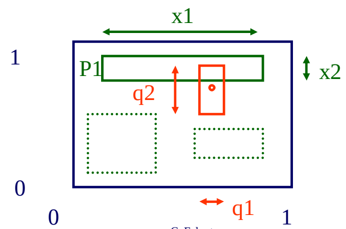


CMU SCS

Advanced - skip

R-trees - performance analysis

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



15-721 C. Faloutsos 85

CMU SCS

Advanced - skip

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

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

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

  q1* q2 * N

15-721 C. Faloutsos 86

CMU SCS

Advanced - skip

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 ($x_{i,2}$)

  q1* q2 * N → count

15-721 C. Faloutsos 87

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

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
- 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).

Advanced - skip

R-trees - performance analysis

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

Advanced - skip

R-trees - performance analysis

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

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

A: ?

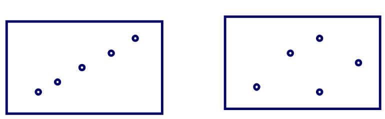
Advanced - skip

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

Advanced - skip

R-trees - performance analysis

What are obvious and/or realistic distributions?

15-721 C. Faloutsos 94

CMU SCS

Advanced - skip

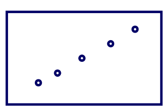
R-trees - performance analysis

What are obvious and/or realistic distributions?

A: uniform

A: Gaussian / mixture of Gaussians

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



15-721 C. Faloutsos 95

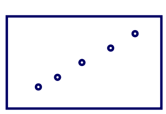
CMU SCS

Advanced - skip

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



15-721 C. Faloutsos 96

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

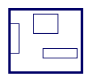
Guttman's R-trees sparked much follow-up work

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

R-trees - variations

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

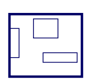
- defer splits, by forced-reinsert, i.e.: instead of splitting, temporarily delete some entries, shrink overflowing MBR, and re-insert those entries
- Which ones to re-insert?
- How many?



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?

CMU SCS

R-trees - variations

Q: Other ways to defer splits?

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

15-721 C. Faloutsos 103

CMU SCS

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.

15-721 C. Faloutsos 104

CMU SCS

R-trees - variations

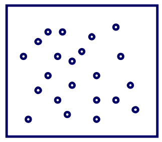
Guttman's R-trees sparked **much** follow-up work

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

15-721 C. Faloutsos 105

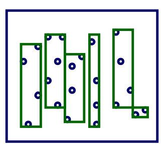
R-trees - variations

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



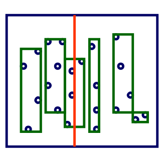
R-trees - variations

- 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’



R-trees - variations

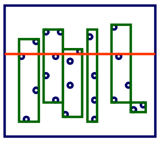
- what about static datasets (no ins/del/upd)?
- Q: Best way to pack points?
- A1: plane-sweep
  great for queries on ‘x’;
  bad for ‘y’



CMU SCS

R-trees - variations

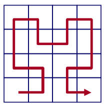
- what about static datasets (no ins/del/upd)?
- Q: Best way to pack points?
- A1: plane-sweep
  great for queries on ‘x’;
  terrible for ‘y’
- Q: how to improve?

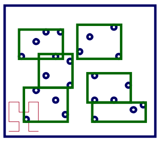


CMU SCS

R-trees - variations

- A: plane-sweep on HILBERT curve!

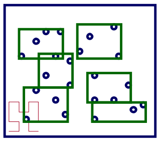




CMU SCS

R-trees - variations

- A: plane-sweep on HILBERT curve!
- In fact, it can be made dynamic (how?), as well as to handle regions (how?)

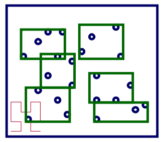


CMU SCS

Advanced - skip

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



15-721 C. Faloutsos 112

CMU SCS

R-trees - variations

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

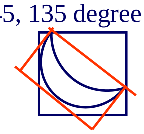
- can we do better splits?
- what about static datasets (no ins/del/upd)?
- what about other bounding shapes?

15-721 C. Faloutsos 113

CMU SCS

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)

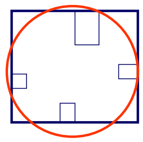


15-721 C. Faloutsos 114

CMU SCS

R-trees - variations

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



15-721 C. Faloutsos 115

CMU SCS

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

15-721 C. Faloutsos 116

CMU SCS

GiST: unifying the variants

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

15-721 C. Faloutsos 117

CMU SCS

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

CMU SCS

GiST

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

CMU SCS

Outline

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

CMU SCS

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

CMU SCS

References

- **Guttman, A. (June 1984). *R-Trees: A Dynamic Index Structure for Spatial Searching*. Proc. ACM SIGMOD, Boston, Mass.**
- **Joseph M. Hellerstein, Jeffrey F. Naughton, Avi Pfeffer: *Generalized Search Trees for Database Systems*. VLDB 1995: 562-573**

CMU SCS

References cont'd

- Edgar Chávez, Gonzalo Navarro, Ricardo A. Baeza-Yates, José L. Marroquín: *Searching in metric spaces*. ACM Comp. Surveys, 33,3, Sept. 2001, pp. 273-321
- Christian Böhm, Stefan Berchtold, Daniel A. Keim: *Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases*. ACM Comp. Surveys, 33,3, Sept. 2001, pp. 322-373

CMU SCS

References cont'd

- Volker Gaede, Oliver Günther: Multidimensional Access Methods. ACM Comp. Surveys, 30,2, June 1998, pp.170-231
- Jagadish, H. V. (May 23-25, 1990). *Linear Clustering of Objects with Multiple Attributes*. ACM SIGMOD Conf., Atlantic City, NJ.

CMU SCS

References, cont'd

- 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.
- 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.

CMU SCS

References, cont'd

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