Spatial Access Methods - problem

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

• Given a collection of geometric objects (points, lines, polygons, ...)
• organize them on disk, to answer
  – point queries
  – range queries
  – k-nn queries
  – spatial joins ('all pairs' queries)
Spatial Access Methods - problem

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

SAMs - motivation

- Q: applications?
SAMs - motivation

traditional DB

GIS

age

salary

SAMs - motivation

traditional DB

GIS

age

salary

SAMs - motivation

CAD/CAM

find elements too close to each other
SAMs - motivation

CAD/CAM

SAMs - motivation

eg. std
eg. avg

F(S1)
F(Sn)

S1
Sn

1 365 day

1 365 day

SAMs - Detailed outline

• spatial access methods
  – problem dfn
  – z-ordering
  – R-trees
SAMs: solutions

- z-ordering
- R-trees

Q: how would you organize, e.g., $n$-dim points, on disk? ($C$ points per disk page)

z-ordering

Q: how would you organize, e.g., $n$-dim points, on disk? ($C$ points per disk page)

Hint: reduce the problem to 1-d points (!!)

Q1: why?
A:
Q2: how?

z-ordering

Q: how would you organize, e.g., $n$-dim points, on disk? ($C$ points per disk page)

Hint: reduce the problem to 1-d points (!!)

Q1: why?
A: B-trees!
Q2: how?
z-ordering

Q2: how?
A: assume finite granularity; z-ordering = bit-shuffling = N-trees = Morton keys = geo-coding = ...

Q2.1: how to map n-d cells to 1-d cells?
z-ordering

Q2.1: how to map $n$-d cells to 1-d cells?
A: row-wise
Q: is it good?

Great for 'x' axis; bad for 'y' axis

Q: How about the 'snake' curve?
z-ordering
Q: How about the ‘snake’ curve?
A: still problems:

Q: Why are those curves ‘bad’?
A: no distance preservation (~ clustering)
Q: solution?

z-ordering
Q: solution? (w/ good clustering, and easy to compute, for 2-d and n-d?)
z-ordering

Q: solution? (w/ good clustering, and easy to compute, for 2-d and n-d?)
A: z-ordering/bit-shuffling/linear-quadtrees

'looks' better:
• few long jumps;
• scoops out the whole quadrant before leaving it
• a.k.a. space filling curves

z-ordering

z-ordering/bit-shuffling/linear-quadtrees

Q: How to generate this curve (z = f(x,y))?
A: 3 (equivalent) answers!

z-ordering

z-ordering/bit-shuffling/linear-quadtrees

Q: How to generate this curve (z = f(x,y))?
A1: ‘z’ (or ‘N’) shapes, RECURSIVELY

z-ordering

z-ordering/bit-shuffling/linear-quadtrees

Q: How to generate this curve (z = f(x,y))?
A1: ‘z’ (or ‘N’) shapes, RECURSIVELY

z-ordering

z-ordering/bit-shuffling/linear-quadtrees

Q: How to generate this curve (z = f(x,y))?
A1: ‘z’ (or ‘N’) shapes, RECURSIVELY
z-ordering

Notice:
• self similar (we'll see about fractals, soon)
• method is hard to use: $z = f(x,y)$

order-1

order-2

... order (n+1)

Q: How to generate this curve ($z = f(x,y)$)?
A: 3 (equivalent) answers!

Method #2?

bit-shuffling

$z = (0101)_2 = 5$
z-ordering

bit-shuffling

\[
\begin{array}{cccc}
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 \\
\end{array}
\]

\[z = (0101)_2 = 5\]

How about the reverse:

\[(x,y) = g(z)\]?

z-ordering

bit-shuffling

\[
\begin{array}{cccc}
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 \\
\end{array}
\]

\[z = (0101)_2 = 5\]

How about n-d spaces?

z-ordering

z-ordering/bit-shuffling/linear-quadtrees

Q: How to generate this curve \((z = f(x,y))\)?

A: 3 (equivalent) answers!

Method #3?
**z-ordering**

**linear-quadtrees**: assign N->1, S->0 e.t.c.

- z-ordering
- ... and repeat recursively. Eg.: $z_{blue-cell} = WN; WN = (0101)_2 = 5$
- Drill: z-value of magenta cell, with the three methods?
z-ordering

Drill: z-value of magenta cell, with the three methods?

method#1: 14
method#2: shuffle(11;10) = (1110)_2 = 14
method#3: EN_ES = ... = 14

z-ordering - Detailed outline

- spatial access methods
  - z-ordering
    - main idea - 3 methods
    - use w/ B-trees; algorithms (range, kNN queries ...)
    - analysis; variations
  - R-trees
z-ordering - usage & algo’s

Q1: How to store on disk?
A: treat z-value as primary key; feed to B-tree

z-ordering - usage & algo’s

Q1: How to store on disk?
A: treat z-value as primary key; feed to B-tree

MAJOR ADVANTAGES w/ B-tree:
• already inside commercial systems (no coding/debugging!)
• concurrency & recovery is ready

z-ordering - usage & algo’s

MAJOR ADVANTAGES w/ B-tree:
• already inside commercial systems (no coding/debugging!)
• concurrency & recovery is ready
z-ordering - usage & algo’s

Q2: queries? (eg.: find city at (0,3) )?

SF

PGH

z  cname  etc
5  SF
12  PGH

A: find z-value; search B-tree

Q2: range queries?
Q2: range queries?
A: compute ranges of z-values; use B-tree

SF 9,11-15
PGH

Q2': range queries - how to reduce # of qualifying ranges?
A: Augment the query!
z-ordering - Detailed outline

• spatial access methods
  – z-ordering
    • main idea - 3 methods
    • use w/ B-trees; algorithms (range, km queries ...)
  • variations
    – R-trees

Q: is z-ordering the best we can do?

A: probably not - occasional long ‘jumps’
Q: then?
z-ordering - variations

Q: is z-ordering the best we can do?
A: probably not - occasional long ‘jumps’
Q: then? A1: Gray codes

z-ordering - variations

A2: Hilbert curve! (a.k.a. Hilbert-Peano curve)

z-ordering - variations

‘Looks’ better (never long jumps). How to derive it?
z-ordering - variations

‘Looks’ better (never long jumps). How to derive it?

order-1  order-2  ...  order (n+1)

Q: function for the Hilbert curve (h = f(x,y))?
A: bit-shuffling, followed by post-processing, to account for rotations. Linear on # bits. See, eg., [Jagadish, 90]

In general, Hilbert curve is great for preserving distances, clustering, vector quantization etc
Conclusions

- z-ordering is a great idea (n-d points -> 1-d points; feed to B-trees)
- used by TIGER system and (most probably) by other GIS products
- works great with low-dim points

SAMs - Detailed outline

- spatial access methods
  - problem dfn
  - z-ordering
  - R-trees

SAMs - more detailed outline

- R-trees
  - main idea; file structure
  - (algorithms: insertion/split)
  - (deletion)
  - search: range, (nn, spatial joins)
  - Variations: R*-trees, packed R-trees
Reminder: problem

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

R-trees

- z-ordering: cuts regions to pieces -> dup. elim.
- how could we avoid that?
- Idea: Minimum Bounding Rectangles

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

A
B
C
D
E
F
G
H
I
J

R-trees

• eg., w/ fanout 4:

P1
A
B
C
D
E

P2
F
G
H
I
J

P3
A
B
C
D
E
F
G
H
I
J

P4
A
B
C
D
E
F
G
H
I
J

R-trees

• eg., w/ fanout 4:

P1
A
B
C
D
E

P2
F
G
H
I
J

P3
A
B
C
D
E
F
G
H
I
J

P4
A
B
C
D
E
F
G
H
I
J

R-trees - format of nodes

• \{(MBR; obj-ptr)\} for leaf nodes

R-trees - format of nodes

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

R-trees - range search?
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
SAMs - more detailed outline

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

R-trees - insertion

- eg., rectangle ‘X’
SAMs - more detailed outline

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

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)

SAMs - more detailed outline

- R-trees
  - main idea; file structure
  - (algorithms: insertion/split)
  - (deletion)
  - search: range, (nn, spatial joins)
  - Variations: R*-trees, packed R-trees
Guttman’s R-trees sparked much follow-up work

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

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

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: Also try to minimize area AND perimeter, in their split.
Performance: higher space utilization; faster than plain R-trees. One of the most successful R-tree variants.

Guttman’s R-trees sparked much follow-up work
• can we do better splits?
• what about static datasets (no ins/del/upd)?
  – Hilbert R-trees
• what about other bounding shapes?
R-trees - variations

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

A1: plane-sweep
great for queries on ‘x’;
terrible for ‘y’
R-trees - variations

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

A: plane-sweep on HILBERT curve!

In fact, it can be made dynamic (how?), as well
as to handle regions (how?)

A: [Kamel+, VLDB94]
R-trees - variations

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

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

R-trees - variations

• what about other bounding shapes? (and why?)

• A1: arbitrary-orientation lines (cell-tree, [Guenther])

• A2: P-trees (polygon trees) (MB polygon: 0, 90, 45, 135 degree lines)

R-trees - variations

• A3: L-shapes; holes (hB-tree)

• A4: TV-trees [Lin+, VLDB-Journal 1994]

• A5: SR-trees [Katayama+, SIGMOD97] (used in Informedia)
R-trees - conclusions

- Popular method; like multi-d B-trees
- Guaranteed utilization
- Good search times (for low-dim. at least)
- R*, Hilbert- and SR-trees: still used
- Informix/DB2 ships DataBlade with R-trees
  - Also in postgres (GiST)
  - And sqlite3 (separate module: R*-tree)

Overall conclusions

- For spatial data:
  - Z-ordering (maps to 1-d points)
  - R-trees (overlapping MBRs)
- Both have been implemented in some commercial systems
- Both work well for low-dimensionalities (<10 or so) - in high-d, it depends on 'intrinsic' dimensionality.

References

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

