15-826: Multimedia Databases and Data Mining

Lecture#5: Multi-key and Spatial Access Methods - II

C. Faloutsos

Must-read material

- Textbook, Chapter 5.1
- Ramakrishnan+Gehrke, Chapter 28.4

Outline

Goal: ‘Find similar / interesting things’
- Intro to DB
- Indexing - similarity search
- Data Mining
Indexing - Detailed outline

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

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)
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

<table>
<thead>
<tr>
<th>Traditional DB</th>
<th>GIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td></td>
</tr>
<tr>
<td>Salary</td>
<td></td>
</tr>
</tbody>
</table>
SAMs - motivation

traditional DB

age

salary

GIS

find elements too close to each other
### SAMs - motivation

- \(S_1\)
- \(S_n\)
- 1 day
- 365 day
- \(F(S_1)\)
- \(F(S_n)\)
- eg, std
- eg, avg

### Indexing - Detailed outline

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

### 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)
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?

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

Q2: how?
A: assume finite granularity (e.g., $2^{32} \times 2^{32}$; 4x4 here)

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

\[ \begin{array}{cccc}
... & ... & ... & ...\\
... & ... & ... & ...\\
\end{array} \rightarrow \begin{array}{cccc}
* & * & * & * \\
* & * & * & * \\
\end{array} \]
**z-ordering**

Q: is it good?
A: great for ‘x’ axis; bad for ‘y’ axis

---

**z-ordering**

Q: How about the ‘snake’ curve?

---

**z-ordering**

Q: How about the ‘snake’ curve?
A: still problems:

\[2^{32}\]
z-ordering

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

2^32

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!

\[ \begin{array}{ccc}
  & & \\
  & 1 & \\
 1 & & 1 \\
 & & \\
\end{array} \]

---

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

A1: ‘z’ (or ‘N’) shapes, RECURSIVELY

\[ \begin{array}{ccc}
  & & \\
  & 1 & \\
 1 & & 1 \\
 & & \\
\end{array} \]

---

Notice:

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

\[ \begin{array}{ccc}
  & & \\
  & 1 & \\
 1 & & 1 \\
 & & \\
\end{array} \]
z-ordering

z-ordering / bit-shuffling / linear-quadtrees

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

A: 3 (equivalent) answers!

Method #2?

How about the reverse:

\((x, y) = g(z)\)?
z-ordering

bit-shuffling

x
0 0
0 1
1 0
1 1

y
1 1
1 0
0 1
0 0

z = (0 1 0 1)_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?

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

... and repeat recursively. E.g.: \( z_{\text{blue-cell}} = WN;WN = (0101)_2 = 5 \)

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

- Method #1: 14
- Method #2: shuffle(11;10) = (1110)\(_2\) = 14
**z-ordering**

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

```
  1  0  0  1
 E N S W
```

- **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 ...)
    - non-point (eg., region) data
    - analysis; variations
  - R-trees
  - ...

---

**z-ordering - usage & algo’s**

Q1: How to store on disk?
A: 

Q2: How to answer range queries etc
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

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

**A:** find z-value; search B-tree

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SF</td>
<td>PGH</td>
<td></td>
</tr>
</tbody>
</table>

### Q2: range queries?

**A:** compute ranges of z-values; use B-tree

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SF</td>
<td>PGH</td>
<td></td>
</tr>
</tbody>
</table>

**z**-ordering - usage & algo’s

**z**-ordering - usage & algo’s

**z**-ordering - usage & algo’s

**z**-ordering - usage & algo’s
z-ordering - usage & algo’s

Q2’: range queries - how to reduce # of qualifying of ranges?

SF  PGH  9,11-15

A: Augment the query!

SF  PGH  9,11-15 -> 8-15

z-ordering - usage & algo’s

Q2’’: range queries - how to break a query into ranges?
z-ordering - usage & algo’s

Q2’’': range queries - how to break a query into ranges?
A: recursively, quadtree-style; decompose only non-full quadrants

z-ordering - usage & algo’s

Q2’’': range queries - how to break a query into ranges?
A: recursively, quadtree-style; decompose only non-full quadrants

z-ordering - Detailed outline

• spatial access methods
  – z-ordering
    • main idea - 3 methods
    • use w/ B-trees; algorithms (range, knn queries ...)
    • non-point (eg., region) data
    • analysis; variations
  – R-trees
  – ...
z-ordering - usage & algo’s

Q3: k-nn queries? (say, 1-nn)?

A: traverse B-tree; find nn wrt z-values and ...

... ask a range query.
z-ordering - usage & algo’s

... ask a range query.

Q4: all-pairs queries? (all pairs of cities within 10 miles from each other?)

(all PA counties that intersect a lake)

z-ordering - Detailed outline

- spatial access methods
  - z-ordering
    - main idea - 3 methods
    - use w/ B-trees; algorithms (range, knn queries ...)
    - non-point (eg., region) data
      - analysis; variations
    - R-trees
      - ...
z-ordering - regions

Q: z-value for a region?

A

B

C

z_B = ??
z_C = ??

z-ordering - regions

Q: z-value for a region?
A: 1 or more z-values; by quadtree decomposition

A

B

C

z_B = ??
z_C = ??

z-ordering - regions

Q: z-value for a region?

A

B

C

w 0 1
0 1
00...
01...
10...
11...

W E N S

00
11

z_B = 11**

z_C = ??

“don’t care”
**z-ordering - regions**

Q: z-value for a region?

```
A   W   E   B
1   01   11...
0   00... 10...
```

```
z_B = 11**
z_C = {0010, 1000}
```

**“don’t care”**

---

**z-ordering - regions**

Q: How to store in B-tree?

Q: How to search (range etc queries)

---

**z-ordering - regions**

Q: How to store in B-tree? *A: sort (*<0<1)*

Q: How to search (range etc queries)

```
A   B
```

```
z   obj-id  etc
0010  C
0101  A
1000  C
11**  B
```
z-ordering - regions

Q: How to search (range etc queries) - eg 'red' range query

A: break query in z-values; check B-tree

Almost identical to range queries for point data, except for the “don’t cares” - i.e.,
**z-ordering - regions**

Almost identical to range queries for point data, except for the "don't cares" - i.e.,
\[ z_1 = 1100 \] \[ ?? \] \[ 11** = z_2 \]

Specifically: does \( z_1 \) contain/avoid/intersect \( z_2 \)?
Q: what is the criterion to decide?

**A:** Prefix property: let \( r_1, r_2 \) be the corresponding regions, and let \( r_1 \) be the smallest \((=> r_1 \text{ has fewest } **\text{'s})\). Then:

- \( r_2 \) will either contain completely, or avoid completely \( r_1 \).
- it will contain \( r_1 \), if \( z_2 \) is the prefix of \( z_1 \)

\[ \begin{array}{c}
A \quad \text{region of } z_1: \\
\text{completely contained in} \\
\text{region of } z_2
\end{array} \]
z-ordering - regions

Drill (True/False). Given:
• z1 = 011001**
• z2 = 01******
• z3 = 0100****
T/F r2 contains r1
T/F r3 contains r1
T/F r3 contains r2

T/F r2 contains r1 - TRUE (prefix property)
T/F r3 contains r1 - FALSE (disjoint)
T/F r3 contains r2 - FALSE (r2 contains r3)
z-ordering - regions

Drill (True/False). Given:

• \( z1 = 011001** \)
• \( z2 = 01****** \)
• \( z3 = 0100**** \)

T/F \( r2 \) contains \( r1 \) - TRUE (prefix property)
T/F \( r3 \) contains \( r1 \) - FALSE (disjoint)
T/F \( r3 \) contains \( r2 \) - FALSE (\( r2 \) contains \( r3 \))

z-ordering - regions

Spatial joins: find (quickly) all counties intersecting lakes

z-ordering - regions

Spatial joins: find (quickly) all counties intersecting lakes
**z-ordering - regions**

**Spatial joins**: find (quickly) all counties intersecting lakes

Naive algorithm: $O(N \times M)$

Something faster?
**z-ordering - regions**

Spatial joins: find (quickly) all counties intersecting lakes

Solution: merge the lists of (sorted) z-values, looking for the prefix property

footnote#1: '*' needs careful treatment
footnote#2: need dup. elimination

---

**z-ordering - Detailed outline**

- spatial access methods
  - z-ordering
    - main idea - 3 methods
    - use w/ B-trees; algorithms (range, knn queries ...)
    - non-point (eg., region) data
    - analysis; variations
  - R-trees
  - ...

---

**z-ordering - variations**

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

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

Gray codes

Ingenious way to spot flickering LED – binary:

- 000 0
- 001 1
- 010 2
- 011 3
- 100 4
- 101 5
- 110 6
- 111 7

F. Gray. Pulse code communication, March 17, 1953
U.S. Patent 2,632,058
(Gray codes)

• Ingenious way to spot flickering LED

0
1
(Gray codes)

- Ingenious way to spot flickering LED

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>000 0</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>001 1</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>011 2</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>010 3</td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>110 4</td>
</tr>
<tr>
<td></td>
<td>111</td>
<td>111 5</td>
</tr>
<tr>
<td></td>
<td>101</td>
<td>101 6</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>100 7</td>
</tr>
</tbody>
</table>
z-ordering - variations

Q: is z-ordering the best we can do?
A: probably not - occasional long 'jumps'
Q: then? A1: Gray codes – CAN WE DO BETTER?

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

(break)

David Hilbert (1862-1943)  Giuseppe Peano (1858-1932)
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 textbook, for pointers to code/algorithms (eg., [Jagadish, 90])
z-ordering - variations

Q: how about Hilbert curve in 3-d? n-d?
A: Exists (and is not unique!). Eg., 3-d, order-1 Hilbert curves (Hamiltonian paths on cube)

z-ordering - Detailed outline

• spatial access methods
  – z-ordering
    • main idea - 3 methods
    • use w/ B-trees; algorithms (range, knn queries ...)
    • non-point (eg., region) data
    • analysis; variations
  – R-trees
  – ...

z-ordering - analysis

Q: How many pieces (‘quad-tree blocks’) per region?
A: proportional to perimeter (surface etc)
z-ordering - analysis

(How long is the coastline, say, of England? Paradox: The answer changes with the yardstick -> fractals ...)

Q: Should we decompose a region to full detail (and store in B-tree)?

A: NO! approximation with 1-3 pieces/z-values is best [Orenstein90]
z-ordering - analysis

Q: how to measure the ‘goodness’ of a curve?

A: e.g., avg. # of runs, for range queries

4 runs 3 runs
(#runs ~ #disk accesses on B-tree)

Q: So, is Hilbert really better?
A: 27% fewer runs, for 2-d (similar for 3-d)

Q: are there formulas for #runs, #of quadtree blocks etc?
A: Yes ([Jagadish; Moon+ etc] see textbook)
z-ordering - fun observations

Hilbert and z-ordering curves: “space filling curves”: eventually, they visit every point in n-d space - therefore:

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

... they show that the plane has as many points as a line (-> headaches for 1900’s mathematics/topology). (fractals, again!)

Observation #2: Hilbert (like) curve for video encoding [Y. Matias+, CRYPTO ‘87]:
Given a frame, visit its pixels in randomized hilbert order; compress; and transmit
z-ordering - fun observations

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

Indexing - Detailed outline

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

Conclusions

- z-ordering is a great idea (n-d points -> 1-d points; feed to B-trees)
- used by TIGER system
  http://www.census.gov/geo/www/tiger/
- and (most probably) by other GIS products
- works great with low-dim points