Nearest Neighbor Queries

Instructor: Cyrus Shahabi
Nearest Neighbor Search

- Retrieve the nearest neighbor of query point Q
- Simple Strategy:
  - convert the nearest neighbor search to range search.
  - Guess a range around Q that contains at least one object say O
    - if the current guess does not include any answers, increase range size until an object found.
  - Compute distance $d'$ between Q and O
  - re-execute the range query with the distance $d'$ around Q.
  - Compute distance of Q from each retrieved object. The object at minimum distance is the nearest neighbor!!!
Naïve Approach

Issues: how to guess range?
The retrieval may be sub-optimal if incorrect range guessed.
Would be a problem in high dimensional spaces.
• Given a query location $q$, find the nearest object.

• Depth First and Best-First Search using R-trees
• Goal: avoid visiting nodes that cannot contain results
Basic Pruning Metric: MINDIST

- Minimum distance between $q$ and an MBR.

$$\text{mindist}(E_1, q)$$

- $\text{mindist}(E_1, q)$ is a lower bound of $d(o, q)$ for every object $o$ in $E_1$.

- If we have found a candidate NN $p$, we can prune every MBR whose $\text{mindist} > d(p, q)$.
MINDIST Property

- MINDIST is a lower bound of any k-NN distance

$$\forall o \in O, \text{MINDIST}(Q, R) \leq \|(Q, o)\|$$
Depth-First (DF) NN Algorithm
Roussoulos et al., SIGMOD, 1995

Main idea
Starting from the root visit nodes according to their mindist in depth-first manner

Note: distances not actually stored inside nodes. Only for illustration
DF Search – Visit $E_1$
DF Search – Find Candidate NN

First Candidate NN: a with distance $\sqrt{5}$
DF Search – Backtrack to Root and Visit $E_2$

First Candidate NN: $a$ with distance $\sqrt{5}$
DF Search – Find Actual NN \( i \)

First Candidate NN: \( a \) with distance \( \sqrt{5} \)

Actual NN: \( i \) with distance \( \sqrt{2} \)
• Question: Which is the minimal set of nodes that must be visited by any NN algorithm?
• Answer: The set of nodes whose MINDIST is smaller than or equal to the distance between $q$ and its NN (e.g., $E_1$, $E_2$, $E_6$).
A Better Strategy for KNN search

• A sorted priority queue based on MINDIST;
• Nodes traversed in order;
• Stops when there is an object at the top of the queue; (1-NN found)
• k-NN can be computed incrementally;

I/O optimal
Priority Queue
Best-First (BF) NN Algorithm (Optimal)
Hjaltason and Samet, TODS, 1998

- Keep a heap $H$ of index entries and objects, ordered by MINDIST.
- Initially, $H$ contains the root.
- While $H \neq \emptyset$
  - Extract the element with minimum MINDIST
    - If it is an index entry, insert its children into $H$.
    - If it is an object, return it as NN.
- End while
BF Search – Visit root

Action

Heap

Visit Root

E₁

E₂

E₃

E₄

E₅

E₆

E₇

x axis

y axis

Root

query

16

12

13
BF Search – Visit $E_1$

Action

Visit Root
follow $E_1$

Heap

$E_1 \sqrt{1}$ $E_2 \sqrt{2}$

$E_2 \sqrt{2}$ $E_3 \sqrt{5}$ $E_5 \sqrt{5}$ $E_4 \sqrt{9}$
BF Search – Visit $E_2$

**Action**
- Visit Root
- follow $E_1$
- follow $E_2$

**Heap**

<table>
<thead>
<tr>
<th>Action</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visit Root</td>
<td>$E_1\sqrt{1}$, $E_2\sqrt{2}$</td>
</tr>
<tr>
<td>follow $E_1$</td>
<td>$E_2\sqrt{2}$, $E_3\sqrt{5}$, $E_5\sqrt{5}$, $E_4\sqrt{9}$</td>
</tr>
<tr>
<td>follow $E_2$</td>
<td>$E_6\sqrt{2}$, $E_3\sqrt{5}$, $E_5\sqrt{5}$, $E_4\sqrt{9}$, $E_7\sqrt{13}$</td>
</tr>
</tbody>
</table>
BF Search – Visit $E_6$

**Action**
- Visit Root
- follow $E_1$
- follow $E_2$
- follow $E_6$

**Heap**

<table>
<thead>
<tr>
<th>Action</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visit Root</td>
<td>$E_1 \sqrt{1}$ $E_2 \sqrt{2}$</td>
</tr>
<tr>
<td>follow $E_1$</td>
<td>$E_2 \sqrt{2}$ $E_3 \sqrt{5}$ $E_5 \sqrt{5}$ $E_4 \sqrt{9}$</td>
</tr>
<tr>
<td>follow $E_2$</td>
<td>$E_6 \sqrt{2}$ $E_3 \sqrt{5}$ $E_5 \sqrt{5}$ $E_4 \sqrt{9}$ $E_7 \sqrt{13}$</td>
</tr>
<tr>
<td>follow $E_6$</td>
<td>$i \sqrt{2}$ $E_3 \sqrt{5}$ $E_5 \sqrt{5}$ $E_4 \sqrt{9}$ $j \sqrt{10}$ $E_7 \sqrt{13}$ $k \sqrt{13}$</td>
</tr>
</tbody>
</table>

Root

```
E_1  E_2
√1   √2
```

```
<table>
<thead>
<tr>
<th>E_1</th>
<th>E_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_3</td>
<td>E_4</td>
</tr>
<tr>
<td>√5</td>
<td>√9</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>E_1</th>
<th>E_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_3</td>
<td>E_4</td>
</tr>
<tr>
<td>√5</td>
<td>√9</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>E_6</th>
<th>E_7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>√2</td>
<td>√13</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>E_6</th>
<th>E_7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>j</td>
</tr>
<tr>
<td>√2</td>
<td>√10</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>E_6</th>
<th>E_7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>m</td>
</tr>
</tbody>
</table>
```

- $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$, $i$, $j$, $k$, $l$, $m$
- $19$, $2$, $4$, $6$, $8$, $10$, $y$ axis, $x$ axis
DF Search – Find Actual NN \( i \)

**Action**
- Visit Root
- follow \( E_1 \)
- follow \( E_2 \)
- follow \( E_6 \)

**Heap**
- \( E_1 \sqrt{1} \)
- \( E_2 \sqrt{2} \)
- \( E_3 \sqrt{5} \)
- \( E_4 \sqrt{9} \)
- \( E_5 \sqrt{5} \)
- \( E_6 \sqrt{2} \)
- \( E_7 \sqrt{13} \)
- \( \sqrt{i} \)
- \( \sqrt{j} \)
- \( \sqrt{k} \)

Report \( i \) and terminate
Generalizations

- Both DF and BF can be easily adapted to (i) extended (instead of point) objects and (ii) retrieval of $k (>1)$ NN.
- BF can be made incremental; i.e., it can report the NN in increasing order of distance without a given value of $k$.
  - Example: find the 10 closest cities to HK with population more than 1 million. We may have to retrieve many ($>>10$) cities around Hong Kong in order to answer the query.
Generalize to k-NN

- Keep a sorted buffer of at most \( k \) current nearest neighbors
- Pruning is done according to the distance of the furthest nearest neighbor in this buffer
- Example:

\[ \text{The } k\text{-th object in the buffer} \]

\[ \text{Actual\_dist} \]

\[ \text{MINDIST} \]

\[ P \]

\[ R \]
Another filter idea based on MBR Face Property

• **MBR** is an n-dimensional **Minimal Bounding Rectangle** used in R trees, which is the minimal bounding n-dimensional rectangle bounds its corresponding objects.

• **MBR face property**: Every face of any MBR contains at least one point of some object in the database.
MBR Face Property – 2D
MBR Face Property – 3D
Improving the KNN Algorithm

• While the MinDist based algorithm is I/O optimal, its performance may be further improved by pruning nodes from the priority queue.
Properties of MINMAXDIST

- **MINMAXDIST(P,R)** is the minimum over all dimensions distances from P to the furthest point of the closest face of R.
- **MINMAXDIST** is the smallest possible upper bound of distances from the point P to the rectangle R.
- **MINMAXDIST** guarantees there is an object within the R at a distance to P less than or equal to minmaxdist.

$$\exists o \in O, \|(Q, o)\| \leq \text{MINMAXDIST}(Q, R)$$

- MINMAXDIST is an upper bound of the 1-NN distance
MINDIST & MINMAXDIST

\[ \text{MINDIST}(P, R) \leq \text{NN}(P) \leq \text{MINMAXDIST}(P, R) \]
MinDist & MinMaxDist – 3D
Pruning 1

- **Downward pruning**: An MBR $R$ is discarded

  If there exists another $R'$ such that $\text{MINDIST}(P,R) > \text{MINMAXDIST}(P,R')$
Pruning 2

- **Downward pruning**: An object $O$ is discarded

  If there exists an $R$ such that $\text{Actual\_dist}(P,O) > \text{MINMAXDIST}(P,R)$
Pruning 3

- **Upward pruning**: An MBR R is discarded

  If an object O is found such that $\text{MINDIST}(P, R) > \text{Actual_dist}(P, O)$
MINDIST vs MINMAXDIST Ordering

- MINDIST: optimistic (the box that is closer)
- MINMAXDIST: pessimistic (the box that has at least one object at that distance)

Example: MINDIST ordering finds the 1-NN first
MINDIST vs MINMAXDIST Ordering

- Example: MINMAXDIST ordering finds the 1-NN first
NN-search Algorithm using the mentioned pruning rules (branch and bound)

1. Initialize the nearest distance as infinite distance
2. Traverse the tree depth-first starting from the root. At each Index node, sort all MBRs using an ordering metric and put them in an **Active Branch List (ABL)**.
3. Apply pruning rules 1 and 2 to ABL
4. Visit the MBRs from the ABL following the order until it is empty
5. If Leaf node, compute actual distances, compare with the best NN so far, update if necessary.
6. At the return from the recursion, use pruning rule 3
7. When the ABL is empty, the NN search returns.
Best-First vs Branch and Bound

• Best-First is the “optimal” algorithm in the sense that it visits all the necessary nodes and nothing more!

• But needs to store a large Priority Queue in main memory. If PQ becomes large, we have thrashing…

• BB uses small Lists for each node. Also uses MINMAXDIST to prune some entries
References


• 2- G. R. Hjaltason and H. Samet, Distance browsing in spatial databases, ACM Transactions on Database Systems 24, 2 (June 1999), 265-318

• STDBM06 keynote slides by Dimitris Papadias “Novel Forms of Nearest Neighbor Search in Spatial and Spatiotemporal Databases”