

# Query Processing in Spatial Network Databases

## SL06

- ▶ Spatial network databases
- ▶ Shortest Path
- ▶ Incremental Euclidean Restriction
- ▶ Incremental Network Expansion

) 2 algorithms

# Literature and Acknowledgments

The slides were developed in collaboration with Johann Gamper from the Free University of Bozen-Bolzano and are based on the following material:

- ▶ Dimitris Papadias, Jun Zhang, Nikos Mamoulis, Yufei Tao, Query Processing in Spatial Network Databases, VLDB 2003, 802-813.

# Spatial Network Databases (SNDB)/1

## Definition (Spatial network databases)

In a **spatial network database** objects can move only on pre-defined trajectories as specified by the underlying network (representing e.g., roads, railways, rivers, etc.)

- ▶ The distance between two objects is the shortest trajectory between the two rather than the Euclidian distance

# Spatial Network Databases (SNDB)/1

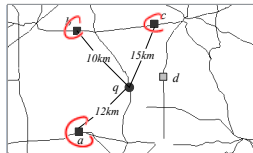
## Definition (Spatial network databases)

In a **spatial network database** objects can move only on pre-defined trajectories as specified by the underlying network (representing e.g., roads, railways, rivers, etc.)

- The distance between two objects is the shortest trajectory between the two rather than the Euclidean distance

## Example

- Query “Find the hotels within a 15km range” returns  $\{a, b, c\}$
- Query “Find the closest hotel” returns  $\{b\}$ 
  - Euclidean nearest neighbor is  $d$  (which is the farthest in the network)



# Spatial Network Databases (SNDB)/2

## Definition (Network distance)



- ▶ For each edge connecting  $n_i$  and  $n_j$ , the network distance,  $d_N(n_i, n_j)$ , is stored with the edge.
- ▶ For nodes  $n_i$  and  $n_j$  that are not directly connected, the network distance,  $d_N(n_i, n_j)$ , is the length of the shortest path from  $n_i$  to  $n_j$ .

**Euclidean Lower-Bound Property:** For any two nodes, the Euclidean distance,  $d_E(n_i, n_j)$ , lower bounds the network distance,  $d_N(n_i, n_j)$ , i.e.,

$$d_E(n_i, n_j) \leq d_N(n_i, n_j)$$



# Shortest Path

- ▶ Much of the work on spatial network databases is on shortest path, nearest neighbor and range search. *locally best decision*
- ▶ Dijkstra's incremental network expansion is a greedy algorithm and the most basic solution to the shortest path problem.
- ▶ Dijkstra's algorithm influenced much of the later work in spatial network databases.
- ▶ Starting point: directed weighted graph
- ▶  $G = (V, E)$  with weight function  $W : E \rightarrow \mathbb{R}$
- ▶ Weight of path  $p = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$  is

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

- ▶ Shortest path = a path of minimum weight (cost)

# Shortest Path Problems

Dijkstra

## ► Single-source

from A to all other nodes

Find a shortest path from a given source vertex  $s$  to all other vertices.

## ► Single-pair

from A to B

Given two vertices, find a shortest path between them. Solution to single-source problem solves this problem efficiently, too.

## ► All-pairs

from all nodes to all other nodes

Find shortest-paths for every pair of vertices. Dynamic programming algorithm.

## ► Unweighted shortest-paths

BFS.

same complexity

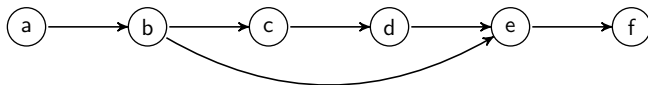
Floyd  
Warshall  
DP

# Optimal Substructure

► **Theorem:**

Subpaths of shortest paths are shortest paths

- Proof: if some subpath were not the shortest path, one could substitute the shorter subpath and create a shorter total path

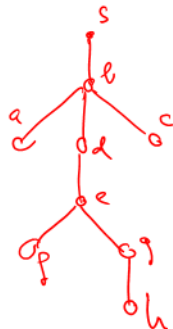


SP: a b c d e f  
must be a SP from b to e

# Shortest Path Tree

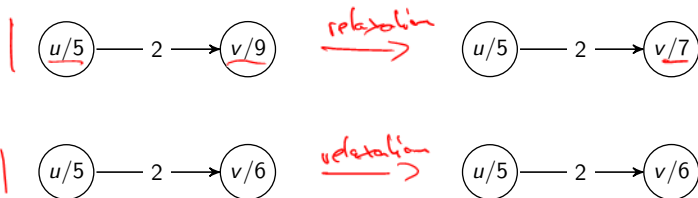
SP from start vertex  
to all other vertices

- ▶ The result of the algorithms is a shortest path tree.  
For each vertex  $v$  it
  - ▶ records a shortest path from the start vertex  $s$  to  $v$
  - ▶  $v.pred$  is the predecessor of  $v$  in this shortest path
  - ▶  $v.dist$  is the shortest path length from  $s$  to  $v$



# Relaxation

- ▶ For each vertex  $v$  in the graph, we maintain  $v.\text{dist}$ , the estimate of the shortest path from  $s$ . It is initialized to  $\infty$  at the start.
- ▶ Relaxing an edge  $(u,v)$  means testing whether we can improve the shortest path to  $v$  found so far by going through  $u$ .



# Dijkstra's Algorithm

- ▶ Basic idea of Dijkstra's algorithm:
  - ▶ maintains a set  $S$  of solved vertices
  - ▶ at each step select closest vertex  $u$ , add it to  $S$ , and relax all edges from  $u$
- ▶ Greedy algorithm that gives optimal solution
- ▶ Similar to breadth-first search (if all weights = 1 then one can simply use BFS)
- ▶ Use a priority queue  $Q$  with keys  $v.\text{dist}$ , which is re-organized whenever some dist decreases

# Dijkstra's Algorithm

$G = (V, E)$   
vertices edges

**foreach**  $u \in G.V$  **do**

$u.dist := \infty$ ;  
     $u.pred := NIL$ ;

$s.dist := 0$ ;

**init**( $Q, G.V$ ) // priority queue  $Q$ ; —  $O(|V|)$

**while**  $\neg isEmpty(Q)$  **do** —  $O(|V|)$

$u := extractMin(Q)$ ; —  $O(|V| \cdot \log |V|)$

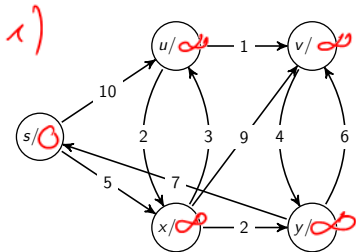
**foreach**  $v \in u.adj$  **do**

$Relax(u, v, G)$ ;

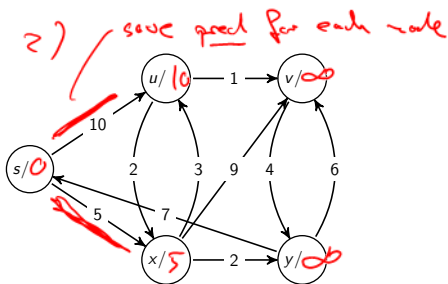
$modifyKey(Q, v)$ ; —  $O(|E| \cdot \log |V|)$

$O((|V| + |E|) \cdot \log |V|)$

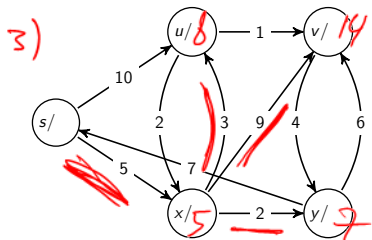
all edges



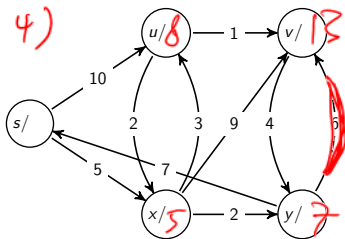
$$Q = \{s(0, -), u(\infty, -), u(\infty, -), u(\infty, -), u(\infty, -)\}$$



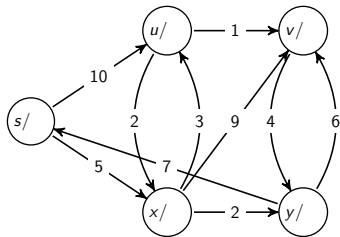
$$Q = \{\underline{x(5, s)}, u(10, s), v(\infty, -), y(\infty, -)\}$$



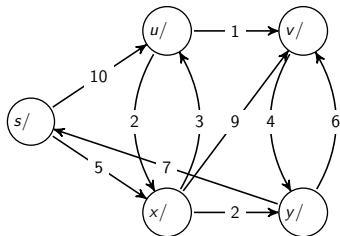
$$Q = \{y(7, x), u(8, x), v(14, x)\}$$



$$Q = \{u(8, x), v(13, y)\}$$



$Q = \{ \cancel{v}(9, u) \}$



$Q = \{ \}$

SP tree



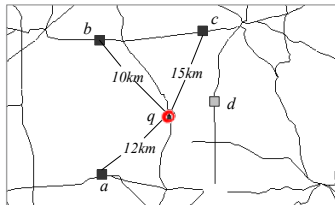
# Nearest Neighbors in SNDB

## Definition

Given a source point  $q$  and an entity dataset  $S$ , a  $k$ -nearest neighbor (kNN) query retrieves the  $k$  ( $\geq 1$ ) objects of  $S$  closest to  $q$  according to the network distance.

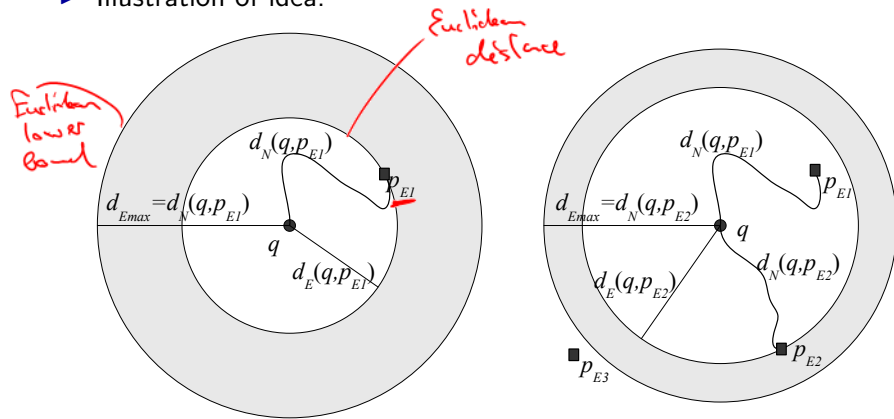
## Example

- ▶  $S = \{a, b, c, d\}$
  - ▶ 1-NN is  $\{b\}$
  - ▶ 2-NN is  $\{b, a\}$
- ordered set*



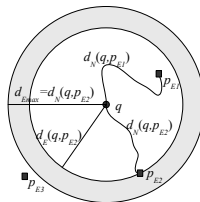
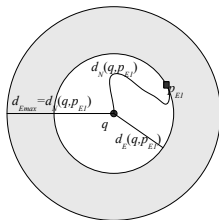
# Incremental Euclidean Restriction (IER)/1

- ▶ Applies the Euclidean distance to reduce the search space in combination with the Euclidean lower bound
- ▶ Illustration of idea:



# Incremental Euclidean Restriction (IER)/2

- ▶ Basic idea for 1NN search
  - ▶ Retrieve the Euclidean NN  $p_{E1}$  using an incremental kNN algorithm on the R-tree built on  $S$ .
  - ▶ Compute the network distance  $d_N(q, p_{E1})$ .
  - ▶ Due to the lower-bound property, objects closer to  $q$  must be within Euclidean distance  $d_{Emax} = d_N(q, p_{E1})$  (shaded area in figure below).
  - ▶ The second Euclidean NN,  $p_{E2}$ , is retrieved with  $d_N(q, p_{E2}) < d_N(q, p_{E1})$ ; thus  $p_{E2}$  becomes the new NN and  $d_{Emax} = d_N(q, p_{E2})$ .
  - ▶ Repeat last step until no more Euclidean NN is found in the search region.



# Incremental Euclidean Restriction (IER)/3

- ▶ Algorithm for kNN search
- ▶  $\{\dots\}$  denotes an ordered set
- ▶  $\{p_1, \dots, p_k\}$  is ordered by  $d_N(p_i, q)$

*initial solution set*

$$\{p_1, \dots, p_k\} = \text{Euclidean\_NN}(q, k);$$

# Incremental Euclidean Restriction (IER)/3

- ▶ Algorithm for kNN search
- ▶  $\{\dots\}$  denotes an ordered set
- ▶  $\{p_1, \dots, p_k\}$  is ordered by  $d_N(p_i, q)$

$\{p_1, \dots, p_k\} = \text{Euclidean\_NN}(q, k);$

**foreach** entity  $p_i$  **do**

$d_N(q, p_i) = \text{compute\_ND}(q, p_i);$

# Incremental Euclidean Restriction (IER)/3

- ▶ Algorithm for kNN search
- ▶  $\{\dots\}$  denotes an ordered set
- ▶  $\{p_1, \dots, p_k\}$  is ordered by  $d_N(p_i, q)$

$\{p_1, \dots, p_k\} = \text{Euclidean\_NN}(q, k);$

**foreach** entity  $p_i$  **do**

$d_N(q, p_i) = \text{compute\_ND}(q, p_i);$

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i);$

# Incremental Euclidean Restriction (IER)/3

- ▶ Algorithm for kNN search
- ▶  $\{\dots\}$  denotes an ordered set
- ▶  $\{p_1, \dots, p_k\}$  is ordered by  $d_N(p_i, q)$

$\{p_1, \dots, p_k\} = \text{Euclidean\_NN}(q, k);$

**foreach** entity  $p_i$  **do**

$d_N(q, p_i) = \text{compute\_ND}(q, p_i);$

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i);$

$d_{E_{\max}} = d_N(q, p_k);$

# Incremental Euclidean Restriction (IER)/3

- ▶ Algorithm for kNN search
- ▶  $\{\dots\}$  denotes an ordered set
- ▶  $\{p_1, \dots, p_k\}$  is ordered by  $d_N(p_i, q)$

$\{p_1, \dots, p_k\} = \text{Euclidean\_NN}(q, k);$

**foreach** entity  $p_i$  **do**

$d_N(q, p_i) = \text{compute\_ND}(q, p_i);$

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i);$

$d_{E_{\max}} = d_N(q, p_k);$

**repeat**

|

# Incremental Euclidean Restriction (IER)/3

- ▶ Algorithm for kNN search
- ▶  $\{\dots\}$  denotes an ordered set
- ▶  $\{p_1, \dots, p_k\}$  is ordered by  $d_N(p_i, q)$

$\{p_1, \dots, p_k\} = \text{Euclidean\_NN}(q, k);$

**foreach** entity  $p_i$  **do**

$d_N(q, p_i) = \text{compute\_ND}(q, p_i);$

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i);$

$d_{E_{\max}} = d_N(q, p_k);$

**repeat**

$(p, d_E(q, p)) = \text{next\_Euclidean\_NN}(q);$

**if** ( $p == \text{NULL}$ ) **then** break;

# Incremental Euclidean Restriction (IER)/3

- ▶ Algorithm for kNN search
- ▶  $\{\dots\}$  denotes an ordered set
- ▶  $\{p_1, \dots, p_k\}$  is ordered by  $d_N(p_i, q)$

$\{p_1, \dots, p_k\} = \text{Euclidean\_NN}(q, k);$

**foreach** entity  $p_i$  **do**

$d_N(q, p_i) = \text{compute\_ND}(q, p_i);$

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i);$

$d_{E_{\max}} = d_N(q, p_k);$

**repeat**

$(p, d_E(q, p)) = \text{next\_Euclidean\_NN}(q);$

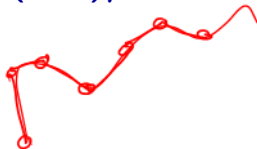
**if**  $(p == \text{NULL})$  **then** break;

**if**  $d_N(q, p) < d_N(q, p_k)$  **then**

our worst  
solution so  
far

# Incremental Euclidean Restriction (IER)/3

- ▶ Algorithm for kNN search
- ▶  $\{\dots\}$  denotes an ordered set
- ▶  $\{p_1, \dots, p_k\}$  is ordered by  $d_N(p_i, q)$



```
 $\{p_1, \dots, p_k\} = \text{Euclidean\_NN}(q, k);$ 
```

```
foreach entity  $p_i$  do
```

```
     $d_N(q, p_i) = \text{compute\_ND}(q, p_i);$ 
```

```
Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i);$ 
```

```
 $d_{E_{\max}} = d_N(q, p_k);$ 
```

```
repeat
```

```
     $(p, d_E(q, p)) = \text{next\_Euclidean\_NN}(q);$ 
```

```
    if ( $p == \text{NULL}$ ) then break;
```

```
    if  $d_N(q, p) < d_N(q, p_k)$  then
```

```
        Insert  $p$  in  $\{p_1, \dots, p_k\};$ 
```

```
         $d_{E_{\max}} = d_N(q, p_k);$ 
```

```
until  $d_E(q, p) > d_{E_{\max}};$ 
```

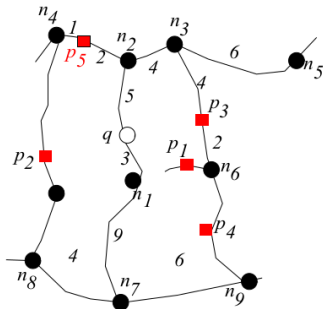
worst drops out

# Incremental Euclidean Restriction (IER)/4

- IER performs well if Euclidean distance is similar to network distance; otherwise, many Euclidean NNs are inspected before the network NN.

## Example

- The nearest entity to query point  $q$  is the entity  $p_5$
- The subscripts in  $p_1, \dots, p_5$  are in ascending order of  $d(q, p_i)$
- $p_5$  will be examined after  $p_1, \dots, p_4$ , since it has the largest Euclidean distance



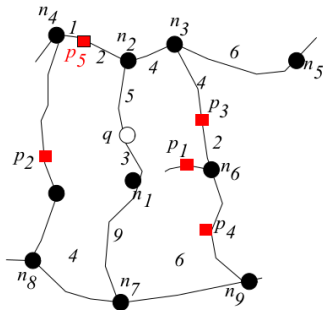
# Incremental Network Expansion (INE)/1

↳ similar to Dijkstra's SP algo

- ▶ Incremental network expansion (INE) performs network expansion starting from query point  $q$  and examines entities in the order they arrive when expanding according to the network.
- ▶ Nodes that have not yet been explored (expanded) are stored in a queue  $Q$  which is sorted according to the network distance from  $q$ .

# Incremental Network Expansion (INE)/2

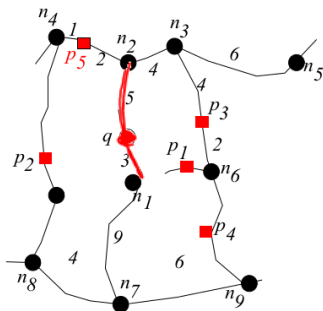
## Example



# Incremental Network Expansion (INE)/2

## Example

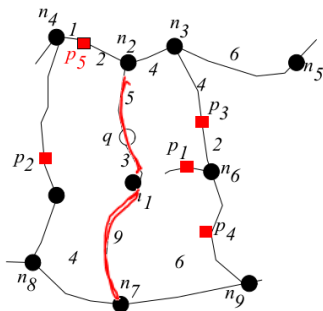
- Locate the edge  $(n_1, n_2)$  that covers  $q$  and add  $n_1$  and  $n_2$  to the queue;  
 $Q = \{(n_1, 3), (n_2, 5)\}$ .



# Incremental Network Expansion (INE)/2

## Example

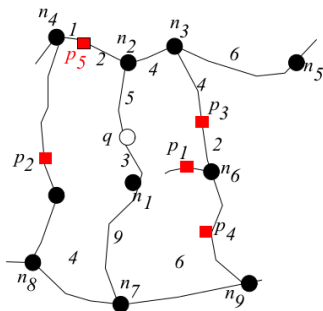
- ▶ Locate the edge  $(n_1, n_2)$  that covers  $q$  and add  $n_1$  and  $n_2$  to the queue;  
 $Q = \{(n_1, 3), (n_2, 5)\}$ .
- ▶ No entity is on  $(n_1, n_2)$ , and the closest node  $n_1$  is expanded;  
 $Q = \{(n_2, 5), (n_7, 12)\}$



# Incremental Network Expansion (INE)/2

## Example

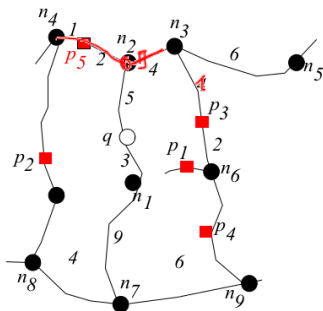
- ▶ Locate the edge  $(n_1, n_2)$  that covers  $q$  and add  $n_1$  and  $n_2$  to the queue;  
 $Q = \{(n_1, 3), (n_2, 5)\}$ .
- ▶ No entity is on  $(n_1, n_2)$ , and the closest node  $n_1$  is expanded;  
 $Q = \{(n_2, 5), (n_7, 12)\}$
- ▶ No entity is on  $(n_1, n_7)$  and  $n_2$  is expanded;  
 $Q = \{(n_4, 8), (n_3, 9), (n_7, 12)\}$



# Incremental Network Expansion (INE)/2

## Example

- ▶ Locate the edge  $(n_1, n_2)$  that covers  $q$  and add  $n_1$  and  $n_2$  to the queue;  
 $Q = \{(n_1, 3), (n_2, 5)\}$ .
  - ▶ No entity is on  $(n_1, n_2)$ , and the closest node  $n_1$  is expanded;  
 $Q = \{(\underline{n_2}, 5), (n_7, 12)\}$
  - ▶ No entity is on  $(n_1, n_7)$  and  $n_2$  is expanded;  
 $Q = \{(\underline{n_4}, 8), (\underline{n_3}, 9), (n_7, 12)\}$
- $p_5$  is discovered on  $(n_2, n_4)$

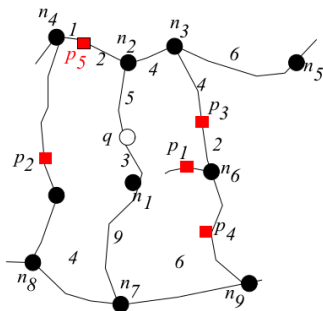


can we stop?

# Incremental Network Expansion (INE)/2

## Example

- ▶ Locate the edge  $(n_1, n_2)$  that covers  $q$  and add  $n_1$  and  $n_2$  to the queue;  
 $Q = \{(n_1, 3), (n_2, 5)\}$ .
- ▶ No entity is on  $(n_1, n_2)$ , and the closest node  $n_1$  is expanded;  
 $Q = \{(n_2, 5), (n_7, 12)\}$
- ▶ No entity is on  $(n_1, n_7)$  and  $n_2$  is expanded;  
 $Q = \{(n_4, 8), (n_3, 9), (n_7, 12)\}$
- ▶  $p_5$  is discovered on  $(n_2, n_4)$
- ▶  $d_N(q, p_5) = 7$  provides a bound to restrict the search space



go on until no  
longer potential  
candidates

# Incremental Network Expansion (INE)/3

**Algorithm:**  $\text{INE}(q, k)$

$n_i n_j = \text{find\_segment}(q);$

# Incremental Network Expansion (INE)/3

**Algorithm:**  $\text{INE}(q, k)$

$n_i n_j = \text{find\_segment}(q);$

$S_{\text{cover}} = \text{find\_entities}(n_i n_j);$

all objects on segment  $u_i u_j$

# Incremental Network Expansion (INE)/3

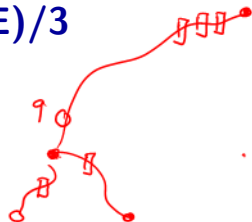
**Algorithm:**  $\text{INE}(q, k)$

$n_i n_j = \text{find\_segment}(q);$

$S_{\text{cover}} = \text{find\_entities}(n_i n_j);$

$\{p_1, \dots, p_k\} = \text{the } k \text{ network nearest entities in } S_{\text{cover}};$

objects  
(possible  
solutions



# Incremental Network Expansion (INE)/3

**Algorithm:**  $\text{INE}(q, k)$

$n_i n_j = \text{find\_segment}(q);$

$S_{\text{cover}} = \text{find\_entities}(n_i n_j);$

$\{p_1, \dots, p_k\} = \text{the } k \text{ network nearest entities in } S_{\text{cover}};$

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i);$

# Incremental Network Expansion (INE)/3

**Algorithm:**  $\text{INE}(q, k)$

$n_i n_j = \text{find\_segment}(q)$ ;

$S_{\text{cover}} = \text{find\_entities}(n_i n_j)$ ;

$\{p_1, \dots, p_k\} = \text{the } k \text{ network nearest entities in } S_{\text{cover}}$ ;

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i)$ ;

**if**  $p_k \neq \emptyset$  **then**  $d_{N_{\max}} = d_N(q, p_k)$  ;

**else**  $d_{N_{\max}} = \infty$  ;

# Incremental Network Expansion (INE)/3

**Algorithm:**  $\text{INE}(q, k)$

$n_i n_j = \text{find\_segment}(q)$ ;

$S_{\text{cover}} = \text{find\_entities}(n_i n_j)$ ;

$\{p_1, \dots, p_k\} = \text{the } k \text{ network nearest entities in } S_{\text{cover}}$ ;

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i)$ ;

**if**  $p_k \neq \emptyset$  **then**  $d_{N_{\text{max}}} = d_N(q, p_k)$  ;

**else**  $d_{N_{\text{max}}} = \infty$  ;

$Q = \langle (n_i, d_N(q, n_i)), (n_j, d_N(q, n_j)) \rangle$ ;

# Incremental Network Expansion (INE)/3

**Algorithm:**  $\text{INE}(q, k)$

$n_i n_j = \text{find\_segment}(q);$

$S_{\text{cover}} = \text{find\_entities}(n_i n_j);$

$\{p_1, \dots, p_k\} = \text{the } k \text{ network nearest entities in } S_{\text{cover}};$

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i);$

**if**  $p_k \neq \emptyset$  **then**  $d_{N_{\max}} = d_N(q, p_k);$

**else**  $d_{N_{\max}} = \infty;$

$Q = \langle (n_i, d_N(q, n_i)), (n_j, d_N(q, n_j)) \rangle;$

De-queue the node  $n$  in  $Q$  with the smallest  $d_N(q, n);$

# Incremental Network Expansion (INE)/3

**Algorithm:**  $\text{INE}(q, k)$

$n_i n_j = \text{find\_segment}(q);$

$S_{\text{cover}} = \text{find\_entities}(n_i n_j);$

$\{p_1, \dots, p_k\} = \text{the } k \text{ network nearest entities in } S_{\text{cover}};$

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i);$

**if**  $p_k \neq \emptyset$  **then**  $d_{N_{\max}} = d_N(q, p_k);$

**else**  $d_{N_{\max}} = \infty;$

$Q = \langle (n_i, d_N(q, n_i)), (n_j, d_N(q, n_j)) \rangle;$

De-queue the node  $n$  in  $Q$  with the smallest  $d_N(q, n);$

**while**  $d_N(q, n) < d_{N_{\max}}$  **do**

# Incremental Network Expansion (INE)/3

**Algorithm:**  $\text{INE}(q, k)$

$n_i n_j = \text{find\_segment}(q)$ ;

$S_{\text{cover}} = \text{find\_entities}(n_i n_j)$ ;

$\{p_1, \dots, p_k\} = \text{the } k \text{ network nearest entities in } S_{\text{cover}}$ ;

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i)$ ;

**if**  $p_k \neq \emptyset$  **then**  $d_{N_{\max}} = d_N(q, p_k)$  ;

**else**  $d_{N_{\max}} = \infty$  ;

$Q = \langle (n_i, d_N(q, n_i)), (n_j, d_N(q, n_j)) \rangle$ ;

De-queue the node  $n$  in  $Q$  with the smallest  $d_N(q, n)$ ;

**while**  $d_N(q, n) < d_{N_{\max}}$  **do**

**foreach** *non-visited adjacent node*  $n_x$  *of*  $n$  **do**

# Incremental Network Expansion (INE)/3

**Algorithm:** INE( $q, k$ )

$n_i n_j = \text{find\_segment}(q)$ ;

$S_{\text{cover}} = \text{find\_entities}(n_i n_j)$ ;

$\{p_1, \dots, p_k\} = \text{the } k \text{ network nearest entities in } S_{\text{cover}}$ ;

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i)$ ;

**if**  $p_k \neq \emptyset$  **then**  $d_{N_{\max}} = d_N(q, p_k)$  ;

**else**  $d_{N_{\max}} = \infty$  ;

$Q = \langle (n_i, d_N(q, n_i)), (n_j, d_N(q, n_j)) \rangle$ ;

De-queue the node  $n$  in  $Q$  with the smallest  $d_N(q, n)$ ;

**while**  $d_N(q, n) < d_{N_{\max}}$  **do**

**foreach** non-visited adjacent node  $n_x$  of  $n$  **do**

$S_{\text{cover}} = \text{find\_entities}(n_x n)$ ;

# Incremental Network Expansion (INE)/3

**Algorithm:** INE( $q, k$ )

$n_i n_j = \text{find\_segment}(q)$ ;

$S_{\text{cover}} = \text{find\_entities}(n_i n_j)$ ;

$\{p_1, \dots, p_k\} = \text{the } k \text{ network nearest entities in } S_{\text{cover}}$ ;

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i)$ ;

**if**  $p_k \neq \emptyset$  **then**  $d_{N_{\text{max}}} = d_N(q, p_k)$  ;

**else**  $d_{N_{\text{max}}} = \infty$  ;

$Q = \langle (n_i, d_N(q, n_i)), (n_j, d_N(q, n_j)) \rangle$ ;

De-queue the node  $n$  in  $Q$  with the smallest  $d_N(q, n)$ ;

**while**  $d_N(q, n) < d_{N_{\text{max}}}$  **do**

**foreach** non-visited adjacent node  $n_x$  of  $n$  **do**

$S_{\text{cover}} = \text{find\_entities}(n_x n)$ ;

        Update  $\{p_1, \dots, p_k\}$  from  $\{p_1, \dots, p_k\} \cup S_{\text{cover}}$ ;

insert and  
remove from  
PQ

# Incremental Network Expansion (INE)/3

**Algorithm:**  $\text{INE}(q, k)$

$n_i n_j = \text{find\_segment}(q)$ ;

$S_{\text{cover}} = \text{find\_entities}(n_i n_j)$ ;

$\{p_1, \dots, p_k\} = \text{the } k \text{ network nearest entities in } S_{\text{cover}}$ ;

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i)$ ;

**if**  $p_k \neq \emptyset$  **then**  $d_{N_{\max}} = d_N(q, p_k)$  ;

**else**  $d_{N_{\max}} = \infty$  ;

$Q = \langle (n_i, d_N(q, n_i)), (n_j, d_N(q, n_j)) \rangle$ ;

De-queue the node  $n$  in  $Q$  with the smallest  $d_N(q, n)$ ;

**while**  $d_N(q, n) < d_{N_{\max}}$  **do**

**foreach** *non-visited adjacent node  $n_x$  of  $n$*  **do**

$S_{\text{cover}} = \text{find\_entities}(n_x n)$ ;

        Update  $\{p_1, \dots, p_k\}$  from  $\{p_1, \dots, p_k\} \cup S_{\text{cover}}$ ;

$d_{N_{\max}} = d_N(q, p_k)$ ;

# Incremental Network Expansion (INE)/3

**Algorithm:**  $\text{INE}(q, k)$

$n_i n_j = \text{find\_segment}(q)$ ;

$S_{\text{cover}} = \text{find\_entities}(n_i n_j)$ ;

$\{p_1, \dots, p_k\} = \text{the } k \text{ network nearest entities in } S_{\text{cover}}$ ;

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i)$ ;

**if**  $p_k \neq \emptyset$  **then**  $d_{N_{\max}} = d_N(q, p_k)$  ;

**else**  $d_{N_{\max}} = \infty$  ;

$Q = \langle (n_i, d_N(q, n_i)), (n_j, d_N(q, n_j)) \rangle$ ;

De-queue the node  $n$  in  $Q$  with the smallest  $d_N(q, n)$ ;

**while**  $d_N(q, n) < d_{N_{\max}}$  **do**

**foreach** *non-visited adjacent node  $n_x$  of  $n$*  **do**

$S_{\text{cover}} = \text{find\_entities}(n_x n)$ ;

        Update  $\{p_1, \dots, p_k\}$  from  $\{p_1, \dots, p_k\} \cup S_{\text{cover}}$ ;

$d_{N_{\max}} = d_N(q, p_k)$ ;

        En-queue  $(n_x, d_N(q, n_x))$ ;

# Incremental Network Expansion (INE)/3

**Algorithm:** INE( $q, k$ )

$n_i n_j = \text{find\_segment}(q)$ ;

$S_{\text{cover}} = \text{find\_entities}(n_i n_j)$ ;

$\{p_1, \dots, p_k\} = \text{the } k \text{ network nearest entities in } S_{\text{cover}}$ ;

Sort  $\{p_1, \dots, p_k\}$  in ascending order of  $d_N(q, p_i)$ ;

**if**  $p_k \neq \emptyset$  **then**  $d_{N_{\max}} = d_N(q, p_k)$  ;

**else**  $d_{N_{\max}} = \infty$  ;

$Q = \langle (n_i, d_N(q, n_i)), (n_j, d_N(q, n_j)) \rangle$ ;

De-queue the node  $n$  in  $Q$  with the smallest  $d_N(q, n)$ ;

**while**  $d_N(q, n) < d_{N_{\max}}$  **do**

**foreach** *non-visited adjacent node  $n_x$  of  $n$*  **do**

$S_{\text{cover}} = \text{find\_entities}(n_x n)$ ;

        Update  $\{p_1, \dots, p_k\}$  from  $\{p_1, \dots, p_k\} \cup S_{\text{cover}}$ ;

$d_{N_{\max}} = d_N(q, p_k)$ ;

        En-queue  $(n_x, d_N(q, n_x))$ ;

    De-queue next node  $n$  in  $Q$ ;

# Summary

- ▶ Network databases consider the street network, which constrains the positions and movements of objects.

- ▶ Incremental Euclidean Restriction (IER) — joins, closest pairs
- ▶ Incremental Network Expansion (INE) — range search, NN
- ▶ The addition of schedules (start and end times, multimodal networks) complicates network algorithms significantly.
  - no goal pruning
  - car  
walk  
train  
from buses

# Summary

- ▶ Network databases consider the street network, which constrains the positions and movements of objects.
- ▶ The Euclidean distance lower bounds the network distance.
- ▶ Incremental Euclidean Restriction (IER)
- ▶ Incremental Network Expansion (INE)
- ▶ The addition of schedules (start and end times, multimodal networks) complicates network algorithms significantly.

# Summary

- ▶ Network databases consider the street network, which constrains the positions and movements of objects.
- ▶ The Euclidean distance lower bounds the network distance.
- ▶ Most network expansion algorithms are based on Dijkstra's shortest path algorithm.
- ▶ Incremental Euclidean Restriction (IER)
- ▶ Incremental Network Expansion (INE)
- ▶ The addition of schedules (start and end times, multimodal networks) complicates network algorithms significantly.