

CSE6488: Mobile Computing Systems

LBS Query Processing Methods in Road Network DB

Sungwon Jung

Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea
Tel: +82-2-705-8930
Email : jungsung@sogang.ac.kr

1

Road Network Query Processing

- If a user at location q poses the range query "find the hotels within a 15km range", the result will contain a , b and c (the numbers in the figure correspond to network distance).
- Similarly, a nearest neighbor query will return hotel b .
 - the Euclidean nearest neighbor is d , which is actually farthest hotel in the network.

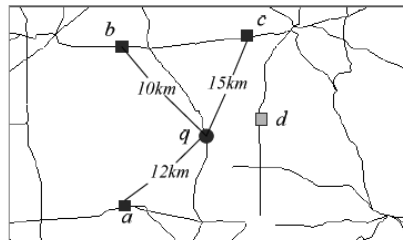
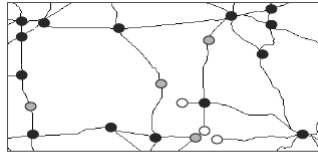


Figure 1.1: Road network query example

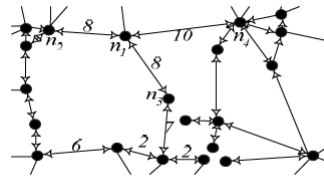
2

Graph Modeling of the Road Network

- The graph nodes generated by this process are:
 - the network junctions (e.g., the black points)
 - the starting/ending point of a road segment (white)
 - depending on the application, additional points (gray) such as the ones where the curvature or speed limit changes



(a) A road network



(b) The modeling graph

3

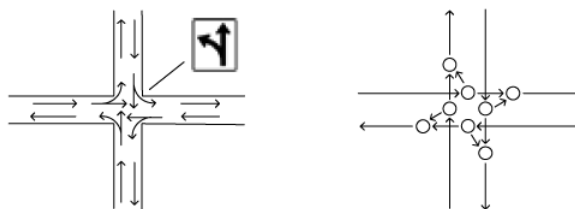
Graph Modeling of the Road Network

- Each edge connecting nodes n_i, n_j stores the *network distance* $\mathbf{d}_N(\mathbf{n}_i, \mathbf{n}_j)$.
- For nodes that are not directly connected, $\mathbf{d}_N(\mathbf{n}_i, \mathbf{n}_j)$ equals the length of the shortest path from n_i to n_j
- If unidirectional traffic is allowed (e.g., one-way road segments), $\mathbf{d}_N(\mathbf{n}_i, \mathbf{n}_j)$ is asymmetric
 - $\mathbf{d}_N(\mathbf{n}_i, \mathbf{n}_j) \neq \mathbf{d}_N(\mathbf{n}_j, \mathbf{n}_i)$.
- *Euclidean lower-bound property*
 - $\mathbf{d}_E(\mathbf{n}_i, \mathbf{n}_j) \leq \mathbf{d}_N(\mathbf{n}_j, \mathbf{n}_i)$

4

Graph Modeling of the Road Network

- Constraints, such as special traffic controls, can be modeled by including extra nodes to the graph



(a) A road junction

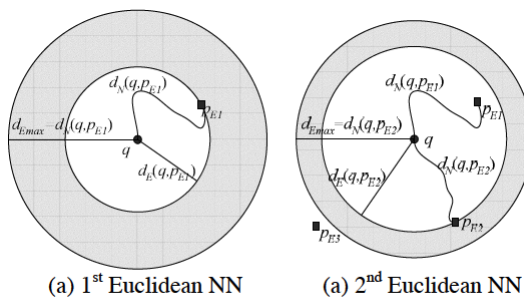
(b) The modeling graph

Figure 3.2: Example of pragmatic constraint

5

Nearest Neighbor Queries in Spatial Network DB

- IER(Incremental Euclidean Restrictions) Algorithm



(a) 1st Euclidean NN

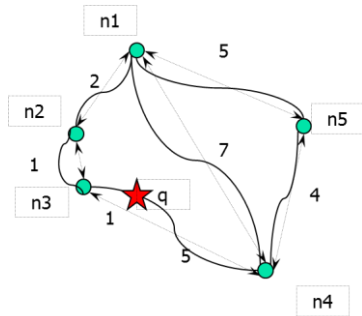
(a) 2nd Euclidean NN

Figure 4.1: Finding the NN p_{E2}

6

Nearest Neighbor Queries in Spatial Network DB

- IER(Incremental Euclidean Restrictions) Algorithm
 - Find the first 1 nearest neighbors of location q



- Find the Euclidean nearest neighbor $n3$
- Compute the network distance:
 $d_N(q, n3) = \text{Compute_ND}(q, n3)$
- Set $d_{E_{\max}} = d_N(q, n3)$
- Repeat the process of retrieving other nodes. To node nk ,
 - if $d_N(q, nk) < d_N(q, n3)$, then set
 $d_{E_{\max}} = d_N(q, nk)$
 - Otherwise, return the node which has set $d_{E_{\max}}$ and stop

7

Nearest Neighbor Queries in Spatial Network DB

- IER Algorithm:
 - Find the k nearest neighbors of location q

Algorithm IER (q, k)

/* q is the query point */

1. $\{p_1, \dots, p_k\} = \text{Euclidean_NN}(q, k)$;
2. for each entity p_i
3. $d_N(q, p_i) = \text{compute_ND}(q, p_i)$
4. sort $\{p_1, \dots, p_k\}$ in ascending order of $d_N(q, p_i)$
5. $d_{E_{\max}} = d_N(q, p_k)$
6. repeat
7. $(p, d_E(q, p)) = \text{next_Euclidean_NN}(q)$;
8. if $(d_N(q, p) < d_N(q, p_k))$ // p closer than the k^{th} NN
9. insert p in $\{p_1, \dots, p_k\}$ // remove ex- k^{th} NN
10. $d_{E_{\max}} = d_N(q, p_k)$
11. until $d_E(q, p) > d_{E_{\max}}$

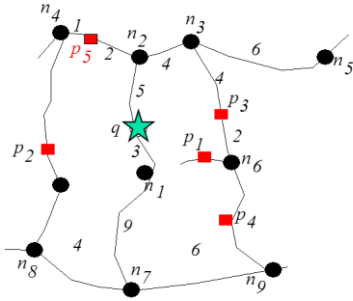
End IER

Figure 4.2: Incremental Euclidean Restriction

8

Nearest Neighbor Queries in Spatial Network DB

- **INE**(Incremental Network Expansion) Algorithm
 - According to IER algorithm, p_5 will be retrieved as the last one, because it has the largest Euclidean distance to q



Algorithm INE (q, k)

1. $n_{i_1} = \text{find_segment}(q)$
2. $S_{\text{cover}} = \text{find_entities}(n_{i_1})$; // S_{cover} is the set of entities covered by n_{i_1}
3. $\{p_1, \dots, p_k\}$ = the k (network) nearest entities in S_{cover} sorted in ascending order of their network distance (p_m, p_{m+1}, \dots, p_k may be \emptyset if S_{cover} contains $< k$ points)
4. $d_{N_{\text{max}}} = d_N(q, p_k)$ // if $p_k = \emptyset$, $d_{N_{\text{max}}} = \infty$
5. $Q = \langle (n_i, d_N(q, n_i)), (n_j, d_N(q, n_j)) \rangle$ // sorted on d_N
6. de-queue the node n in Q with the smallest $d_N(q, n)$
7. while $(d_N(q, n) < d_{N_{\text{max}}})$
8. for each non-visited adjacent node n_x of n
9. $S_{\text{cover}} = \text{find_entities}(n_x, n)$;
10. update $\{p_1, \dots, p_k\}$ from $\{p_1, \dots, p_k\} \cup S_{\text{cover}}$
11. $d_{N_{\text{max}}} = d_N(q, p_k)$
12. en-queue $(n_x, d_N(q, n_x))$
13. de-queue the next node n in Q

End INE

Figure 4.4: Incremental Network Expansion

Range Queries in SNDB

- **RER**(Range Euclidean Restriction) method
 - Given a source point q , a value e and a spatial dataset S , a range query retrieves all objects of S that are within network distance e from q
 - $d_N(q, p) \leq e \rightarrow d_E(q, p) \leq e$

Algorithm RER(q, e)

/* q : query point, e : the network distance threshold */

1. $result = \emptyset$
2. $S' = \text{Euclidean-range}(q, e)$
3. $n_{i_1} = \text{find_segment}(q)$
4. $Q = \langle (n_i, d_N(q, n_i)), (n_j, d_N(q, n_j)) \rangle$
5. de-queue the node n in Q with the smallest $d_N(q, n)$
6. while $(d_N(q, n) \leq e \text{ and } S' \neq \emptyset)$
7. for each non-visited adjacent node n_x of n
8. for each point s of S'
9. if $\text{check_entity}(n_x, n, s)$
10. $result = result \cup \{s\}$; $S' = S' - \{s\}$
11. en-queue $(n_x, d_N(q, n_x))$
12. de-queue the next node n in Q
13. end while

End RER

Figure 5.1: Range Euclidean Restriction

10

Range Queries in SNDB

- RNE(Range Network Expansion) method
 - RNE first computes the set QS of qualifying segments within network range e from q
 - It then retrieves the data entities falling on these segments

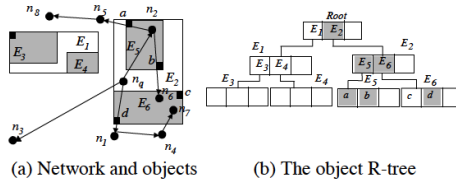


Figure 5.2: Example of RNE

- Alternative Method:
 - The MBR of all segments in QS is applied as a range query to the object R-tree

Algorithm RNE($node_id$, QS , $result$)

1. if ($node_id$ is an intermediate node)
 2. compute QS_i for each entry E_i in $node_id$ // join
 3. for each entry E_i in $node_id$
 4. if ($QS_i \neq \emptyset$)
 5. RNE($E_i, node_id$, QS_i , $result$)
 6. else // $node$ is a leaf node
 7. $result_{node_id} = \text{plane-sweep}(node_id.entries, QS_i)$
 8. sort $result_{node_id}$ to remove duplicates
 9. $result = result \cup result_{node_id}$
- End RNE**

Closet-Pairs Euclidean Restriction

Algorithm CPER (S, T, k)

/* S and T are two entity data sets; k is the number of closet pairs to be retrieved*/

1. $\{(s_1, t_1), \dots, (s_k, t_k)\} = \text{Euclidean_CP}(S, T, k)$
// find the k Euclidean closet pairs
2. for $i=1$ to k
3. $d_N(s_i, t_i) = \text{compute_ND}(s_i, t_i)$
4. sort (s_i, t_i) in ascending order of their $d_N(s_i, t_i)$
5. $d_{Emax} = d_N(s_k, t_k)$
6. repeat
7. $(s', t') = \text{next_Euclidean_CP}(S, T)$
8. $d_N(s', t') = \text{compute_ND}(s', t')$
9. if ($d_N(s', t') < d_{Emax}$)
// (s', t') is closer in the network than (s_k, t_k)
10. insert (s', t') in $\{(s_1, t_1), \dots, (s_k, t_k)\}$
11. $d_{Emax} = d_N(s_k, t_k)$
12. until $d_E(s', t') > d_{Emax}$

End CPER