

Temporal and Spatial Databases

Chapter 11: Query Processing in Spatial Network Databases

J. Gamper

- ▶ Nearest neighbor search
- ▶ Closest-pair search
- ▶ Computing isochrones

Literature

- ▶ D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao: Query processing in spatial network databases. In *Proc. of the VLDB*, 2003.
- ▶ V. Bauer, J. Gamper, R. Loperfido, S. Profanter, S. Putzer, and I. Timko: Computing isochrones in multi-modal, schedule-based transport networks. In *Proc. of the 16th ACM-GIS*, pages 1-2, 2008.

Spatial Network Databases (SNDB)

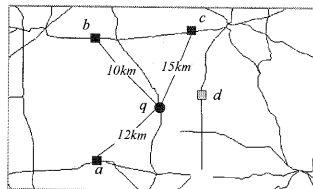
Definition (Spatial network databases)

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) ...

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 which 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.,

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

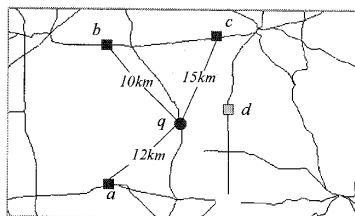
Nearest Neighbors in SNDB

Definition

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

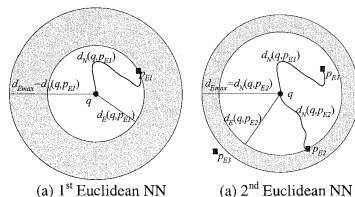
Example

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



Incremental Euclidean Restriction (IER)

- ▶ Applies the Euclidean distance to reduce the search space in combination with the Euclidean lower bound
- ▶ Basic idea for 1-NN 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 should be within Euclidean distance $d_{E_{max}} = d_N(q, p_{E1})$ (shaded area in the 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 $d_{E_{max}} = d_N(q, p_{E2})$.
 - ▶ Repeat the last step until no more Euclidean NN is found in the search region



Incremental Euclidean Restriction (IER) ...

Algorithm: IER(q, k)

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

foreach entity p_i **do**

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

end

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

 //p closer than the kth NN

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

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

end

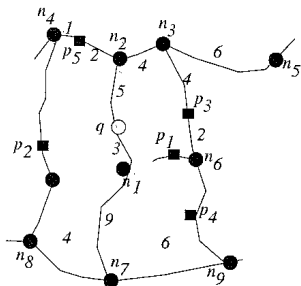
until $d_N(q, p) > d_{E_{max}};$

Incremental Euclidean Restriction (IER) ...

- ▶ 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 network distance

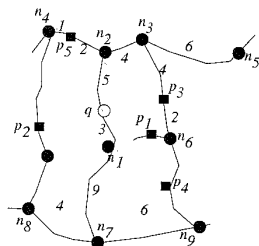


Incremental Network Expansion (INE)

- ▶ Performs network expansion starting from query point q and examines entities in the order they arrive
- ▶ Nodes not yet explored are stored in a queue Q which is sorted according to the network distance from q

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;
 $Q = \{(n_4, 7), (n_3, 9), (n_7, 12)\}$
- ▶ p_5 is discovered on (n_2, n_4)
- ▶ $d_N(q, p_5) = 6$ provides a bound to restrict the search space



Incremental Network Expansion (INE) ...

Algorithm: INE(q, k)

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

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

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

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

// p_m, \dots, p_k may be \emptyset if S_{cover} contains $< k$ entities

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

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

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

end

 De-queue next node n in Q ;

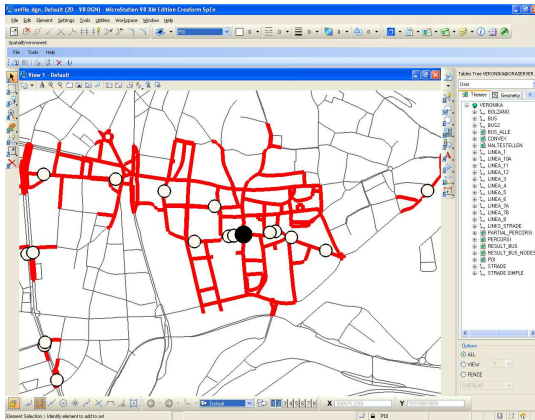
end

Isochrones

- ▶ **Isochrones:** Intuitively, the set of all points in a SNDB from which a point of interest (query point) can be reached within a given timespan.
- ▶ In multi-modal networks different transportation modes have to be considered, e.g., walking, taking a bus, using the car/bicycle, etc.
- ▶ Bus networks have schedules, thus the isochrone
 - ▶ depends also on the arrival time at the point of interest;
 - ▶ might be formed of several islands around bus stations.
- ▶ Can be used as an instrument for city planners
 - ▶ Analyse the coverage of the city with important services
 - ▶ How many citizen can reach a hospital in 20 minutes?
 - ▶ How many children can reach a specific school in 15 minutes?

Isochrones ...

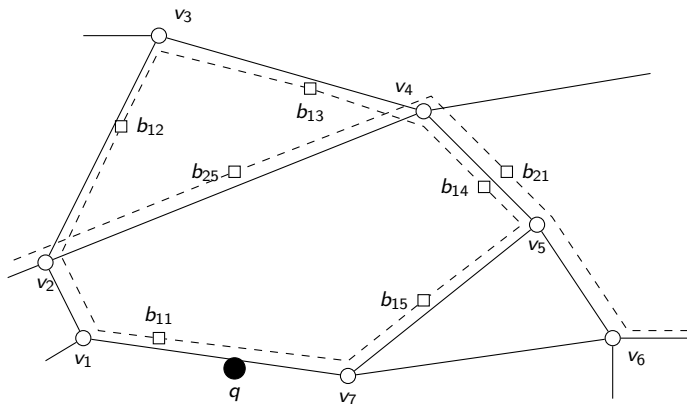
Example



Isochrones ...

Example

Multi-modal network consisting of a street/pedestrian network (solid lines) and two bus lines (dashed lines).



Pedestrian Network

Definition (Pedestrian Network)

A **pedestrian network** is a directed graph $N = (V, E, \lambda)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, and $\lambda : E \mapsto \mathbb{R}^+$ is a function that maps an edge $(v_i, v_j) \in E$ to a positive real number that represents the edge's length.

Example

In our example we have $N = (V, E, \lambda)$, where

- ▶ $V = \{v_1, v_2, \dots, v_7\}$
- ▶ $E = \{(v_1, v_2), (v_2, v_3), \dots\}$
- ▶ $\lambda((v_1, v_2)) = 70, \dots$

We use also $e^{v, v'}$ to represent edge $(v, v') \in E$

Pedestrian Network ...

Definition (Network Location)

A **network location** in a pedestrian network $N = (V, E, \lambda)$ is any point that lies on an edge, and we represent it as $l = (e^{v,v'}, o)$, where $e^{v,v'} \in E$ is an edge and $o \in \mathbb{R}$, $0 \leq o \leq \lambda(e^{v,v'})$, is an offset which represents the relative position of l from v on the edge $e^{v,v'}$.

Example

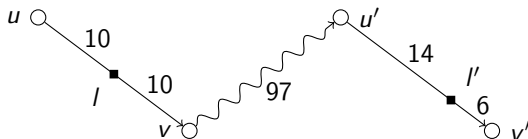
- ▶ $l = (e^{v,v'}, 10)$ is the point on $e^{v,v'}$ with a relative position of 10 from v ;
- ▶ $l = (e^{v,v'}, 0)$ is the point which lies on node v ;
- ▶ $l = (e^{v,v'}, \lambda(e^{v,v'}))$ is the point which lies on node v' ;

Pedestrian Network ...

Definition (Path in Pedestrian Network)

Let $N = (V, E, \lambda)$ be a pedestrian network and $l = (e^{u,v}, o)$ and $l' = (e^{u',v'}, o')$ be two network locations. A **path** from l to l' is defined as a sequence of locations and vertices $p^{l,l'} = (l, v_0, \dots, v_k, l')$, where $v_0 = v$, $v_k = u'$, and $\forall i, 1 \leq i \leq k : e^{v_{i-1}, v_i} \in E$.

Example



- ▶ $l = (e^{u,v}, 10), l' = (e^{u',v'}, 14)$
- ▶ $p^{l,l'} = ((e^{u,v}, 10), v, \dots, u', (e^{u',v'}, o'))$

Pedestrian Network ...

Definition (Length of Path in Pedestrian Network)

Let $N = (V, E, \lambda)$ be a pedestrian network and $p^{l,l'} = (l, v_0, \dots, l_k, l')$ be a path from location $l = (e^{u,v}, o)$ to location $l' = (e^{u',v'}, o')$. The **length** of the path $p^{l,l'}$ is defined as

$$\lambda(p^{l,l'}) = \lambda(e^{u,v}) - o + \sum_{i=1}^k \lambda(e^{v_{i-1},v_i}) + o'$$

Example

Length of the path from l to l' : $p^{l,l'} = (20 - 10) + 79 + 14 = 103$

Pedestrian Network ...

Definition (Cost of Path in Pedestrian Network)

Let $N = (V, E, \lambda)$ be a pedestrian network, $s > 0$ be an average walking speed, and $p^{l,l'} = (l, v_0, \dots, l_k, l')$ be a path from location $l = (e^{u,v}, o)$ to location $l' = (e^{u',v'}, o')$. The cost of the path $p^{l,l'}$ is defined as

$$\gamma(p^{l,l'}, s) = \frac{\lambda(p^{l,l'})}{s}$$

Schedule-Based Network

Definition (Schedule-based Network)

A schedule-based network is a directed graph $N_s = (V, E, TID, S)$, where

- ▶ V is a set of vertices;
- ▶ $E \subseteq V \times V$ is a set of edges such that each vertex has exactly one ingoing and one outgoing edge, i.e.,

$$\forall v \in V \exists u, w \in V (e^{u,v} \in E \wedge e^{v,w} \in E)$$

$$\forall e^{u,v}, e^{u',v'} \in E (u \neq u' \wedge v \neq v')$$

- ▶ TID is a finite set of trip identifiers;
- ▶ $S = (\sigma_a, \sigma_d)$ is a schedule specified by two functions $\sigma_a : TID \times V \mapsto T$ and $\sigma_d : TID \times V \mapsto T$, where $\sigma_a(tid, v) \leq \sigma_d(tid, v)$ for all $tid \in TID$ and $v \in V$; the two functions represent arrival and departure times, respectively.

Schedule-Based Network ...

Example (Schedule)

The following table represents a schedule:

- ▶ $TID = \{\dots, 5, 6, \dots\}$
- ▶ $V = \{\dots, s_1, s_2, s_3, \dots\}$
- ▶ Columns “Arrival time” and “Departure time” represent σ_a and σ_d
 - ▶ e.g., $\sigma_a(5, s_2) = 09:41:00$, $\sigma_d(5, s_2) = 09:41:30$

TripID	Station	Arrival time	Departure time
⋮	⋮	⋮	⋮
5	s_1	—	09:40:00
5	s_2	09:41:00	09:41:30
5	s_3	09:42:00	09:42:00
5	s_4	09:42:30	09:43:00
⋮	⋮	⋮	⋮
6	s_1	—	10:00:00
6	s_2	10:01:00	10:01:00
6	s_3	10:01:30	10:02:00
6	s_4	10:02:30	10:03:00
⋮	⋮	⋮	⋮

Path in Schedule-Based Network

Definition (Path in Schedule-Based Network)

Let $N = (V, E, TID, \sigma)$ be a schedule-based network. A *path* from vertex v_0 to vertex v_k is defined as a sequence of vertices $p^{v_0, v_k} = (v_0, \dots, v_k)$, where $e^{v_{i-1}, v_i} \in E$ for $i = 1, \dots, k$.

Multi-Modal Network

Definition (Multi-modal Network)

A multi-modal network is defined as $M = (\mathcal{N}, \mu)$, where $\mathcal{N} = \{N_1, \dots, N_k\}$ is a set of (pedestrian/bus) networks and μ is a mapping that maps a network location into the corresponding locations in all other networks in \mathcal{N} .

Path in Multi-Modal Network

Definition (Path in Multi-Modal Network)

Let $M = (\mathcal{N}, \mu)$ be a multi-modal network with $\mathcal{N} = \{N_1, \dots, N_m\}$, and let $P(N_i)$ denote the set of all paths in N_i . A *path* from location l in network N to a location l' in network N_j is defined as a sequence of network locations $p^{l,l'} = (l_0, \dots, l_k)$ such that for $i = 1, \dots, k$ the following holds:

$$\begin{aligned} & \exists N \in \mathcal{N} (l_{i-1} \in N \wedge l_i \in N \wedge \exists p^{l_{i-1}, l_i} \in P(N)) \vee \\ & \exists N, N' \in \mathcal{N} (l_{i-1} \in N \wedge l_i \in N' \wedge N \neq N' \wedge (l_{i-1}, l_i) \in \mu) \end{aligned}$$

Definition of Isochrone

Definition (Isochrone)

Let $M = (\mathcal{N}, \mu)$ be a multi-modal network, $P(M)$ denote the set of all paths in M , let l_{poi} be a network location that represents a point of interest, t_{poi} be the arrival time at the point of interest, and $d > 0$ be a time duration. An *isochrone*, ISO , is defined as the set of all network locations from which the point of interest can be reached by t_{poi} and starting not earlier than $t_{poi} - d$, i.e.,

$$\forall l (l \in ISO \iff \exists p^{l, l_{poi}} \in P(M) (\gamma(p^{l, l_{poi}}) \leq d))$$

Algorithm

Algorithm: ISOCHRONE($N, l_{poi}, t_{poi}, d, s_w$)

Add $(l_{poi}, t_{poi}, N_{streets})$ to empty queue Q ;

while $Q \neq \emptyset$ **do**

$(n, t, N) \leftarrow$ next node from Q ;

foreach (*incoming*) link $(n', n) \in N$ **do**

 Compute latest starting time, t' , for n' ;

if $t' > t_{poi} - d$ **then**

 Add (n', t', N) to Q ;

if $N = N_{streets}$ **then**

 Add (n', n) to result set;

 Check for bus stops on (n', n) and add them to Q with appropriate starting time;

end

else if N is a bus network **then**

 Check for stops of other buses at n' and add them to Q with appropriate starting time;

end

end

end

end

Future Work

- ▶ Computation of isochrone as area rather than as network
 - ▶ Fixed/varying buffer size around network representation
 - ▶ Exploiting the houses plus convex/concave hull computation
- ▶ Join isochrone with inhabitants DB to get the population leaving in an isochrone
- ▶ Efficient algorithms for incremental computation of isochrones