Query Processing in Spatial Network Databases
SL07

- Spatial network databases
- Shortest Path
- Incremental Euclidean Restriction
- Incremental Network Expansion
- Multimodal Networks

Acknowledgements: I am indebted to Michael Böhlen for providing me the slides.
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)
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.
- 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 \( W : E \to \mathbb{R} \)
- Weight of path \( p = v_1 \to v_2 \to \ldots \to v_k \) 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

- **Single-source**
  Find a shortest path from a given source vertex \( s \) to all other vertices.

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

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

- **Unweighted shortest-paths**
  BFS.
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
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.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.

\[ u/5 \rightarrow v/9 \]
\[ u/5 \rightarrow v/6 \]
\[ u/5 \rightarrow v/7 \]
\[ u/5 \rightarrow v/6 \]
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 $v$.dist, which is re-organized whenever some dist decreases
Dijkstra’s Algorithm

```plaintext
foreach u ∈ G.V do
    u.dist := ∞;
    u.pred := NIL;

s.dist := 0;
init(Q, G.V) // priority queue Q;

while ¬isEmpty(Q) do
    u := extractMin(Q);
    foreach v ∈ u.adj do
        Relax(u, v, G);
        modifyKey(Q, v);
```

```
Dijkstra’s Algorithm

Example:

- Initialization
- Status after each `while` iteration
Dijkstra’s Algorithm

Init:

\[ Q = \langle (s/0), (u/\infty), (x/\infty), (v/\infty), (y/\infty) \rangle \]
Dijkstra’s Algorithm

1st while iteration $u = (s/0)$:

![Diagram](image)

$Q = \langle(x/5), (u/10), (v/\infty), (y/\infty)\rangle$
Dijkstra’s Algorithm

2\textsuperscript{nd} while iteration \( u = (x/5) \):

\[ Q = \langle (y/7), (u/8), (v/14) \rangle \]
Dijkstra’s Algorithm

3rd while iteration $u = (y/7)$:

$Q = \langle (u/8), (v/13) \rangle$
Dijkstra’s Algorithm

$4^{th}$ while iteration $u = (u/8)$:

$$Q = \langle (v/9) \rangle$$
Dijkstra’s Algorithm

5th while iteration $u = (v/9)$:

$Q = \langle \rangle$
Dijkstra’s Algorithm

Result:

\[ s / 0 \rightarrow u / 8 \rightarrow v / 9 \]
\[ x / 5 \rightarrow y / 7 \]
Dijkstra’s Algorithm

- The example calculated the shortest path from a given source vertex to all other vertices, i.e., it was single-source.

- When do we stop if we are looking for single-pair?

  **Stop-condition:** When `extractMin` returns the destination vertex.
Dijkstra’s algorithm expands all vertices with distance smaller than \( d(s, t) \).

- Is this large search space needed?
Bidirectional Search

- Simultaneously run a **forward search** from $s$ and a **backward search** from $t$.
- Search stops as soon as the search spaces intersect and contain a node $x$ on the shortest path from $s$ to $t$.

- Dijkstra and bidirectional search apply blind search!
A*-Star Search

- **Goal-directed method**: aims to “guide” the search towards the target
  - nodes “closer” to the target are expanded first.
- Priority of nodes $x$ in $Q$ is determined by $d(s, x) + d_E(x, t)$
  - Estimation of the cost of the SP
  - $d_E$ is the Euclidean distance (lower bound of network distance $d$).
- Bidirectional A*-search exists
Nearest Neighbors in SNDB

Definition

Given a source point \( q \) and an entity dataset \( S \), a \textbf{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 \( \{b, a\} \)
Incremental Euclidean Restriction (IER)/1

- 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 $d_{Emax} = d_N(q, p_{E1})$ (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 search region.
Incremental Euclidean Restriction (IER)/2

\[
\{p_1, \ldots, p_k\} = \text{Euclidean}_\text{NN}(q, k);
\]

\begin{itemize}
  \item \textbf{foreach} entity \(p_i\) \textbf{do}
  \begin{itemize}
    \item \(d_N(q, p_i) = \text{compute}_\text{ND}(q, p_i)\);
  \end{itemize}
\end{itemize}

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

\[
d_{E\text{max}} = d_N(q, p_k);
\]

\textbf{repeat}

\begin{itemize}
  \item \((p, d_E(q, p)) = \text{next}_\text{Euclidean}_\text{NN}(q)\);
  \item \textbf{if} \(d_N(q, p) < d_N(q, p_k)\) \textbf{then}
    \begin{itemize}
      \item Insert \(p\) in \(\{p_1, \ldots, p_k\}\);
      \item \(d_{E\text{max}} = d_N(q, p_k)\);
    \end{itemize}
\end{itemize}

\textbf{until} \(d_N(q, p) > d_{E\text{max}}\);
Incremental Euclidean Restriction (IER)/3

- 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 $p_1, \ldots, p_5$ are in ascending order of $d(q, p_i)$
- $p_5$ will be examined after $p_1, \ldots, p_4$, since it has the largest Euclidean distance
Incremental Network Expansion (INE)/1

- 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;
  \[ Q = \{(n_4, 8), (n_3, 9), (n_7, 12)\} \]

- \( p_5 \) is discovered on \((n_2, n_4)\)

- \( d_N(q, p_5) = 7 \) provides a bound to restrict the search space.
Algorithm: \textsc{INE}(q, k)

\begin{align*}
n_i n_j &= \text{find\_segment}(q); \\
S_{\text{cover}} &= \text{find\_entities}(n_i n_j); \\
\{p_1, \ldots, p_k\} &= \text{the } k \text{ network nearest entities in } S_{\text{cover}}; \\
\text{Sort } \{p_1, \ldots, p_k\} \text{ in ascending order of } d_N(q, p_i); \\
\textbf{if } p_k \neq \emptyset \textbf{ then } d_{N_{\text{max}}} = d_N(q, p_k) \textbf{ else } d_{N_{\text{max}}} = \infty \\
Q &= \langle(n_i, d_N(q, n_i)), (n_j, d_N(q, n_j))\rangle; \\
\text{De-queue the node } n \text{ in } Q \text{ with the smallest } d_N(q, n); \\
\textbf{while } d_N(q, n) < d_{N_{\text{max}}} \textbf{ do} \\
\quad \textbf{foreach non-visited adjacent node } n_x \text{ of } n \textbf{ do} \\
\quad \quad S_{\text{cover}} &= \text{find\_entities}(n_x n); \\
\quad \quad \text{Update } \{p_1, \ldots, p_k\} \text{ from } \{p_1, \ldots, p_k\} \cup S_{\text{cover}}; \\
\quad \quad d_{N_{\text{max}}} &= d_N(q, p_k); \\
\quad \quad \text{En-queue } (n_x, d_N(q, n_x)); \\
\quad \text{De-queue next node } n \text{ in } Q;
\end{align*}
Multimodal Network

- Networks get significantly more complicated if **schedules** must be considered as well.
- A multimodal network has different transportation modes between nodes.
  - **continuous space and time mode**, $\mu(.) = 'csct'$, e.g., pedestrian network;
  - **discrete space and time mode**, $\mu(.) = 'dsdt'$, e.g., the public transport system such as trains and buses
  - **discrete space continuous time mode**, $\mu(.) = 'dsct'$, e.g., moving walkways or stairs
  - **continuous space discrete time mode**, $\mu(.) = 'csdt'$, e.g., regions or streets that can be passed by pedestrians or cars only in specific time slots.
Multimodal Network

is a seven-tuple \( N = (G, R, S, \rho, \mu, \lambda, \tau) \), where

- \( G = (V, E) \) a directed multigraph with set \( V \) of vertices and multiset \( E \) of ordered pairs of vertices termed edges
- \( R \) is a set of transport systems (Walking, Bus, Train,...)
- partial functions that assigns to each edge
  - \( \rho : E \mapsto R \) the transportation system
  - \( \mu : E \mapsto \{ 'csct', 'csdt', 'dsct', 'dsdt' \} \) the transportation mode (continuous or discrete in space and time)
  - \( \lambda : E \mapsto \mathbb{R}^+ \) the length,
  - \( \tau : E \times T \mapsto \mathbb{R}^+ \) the transfer time
- \( S = (R, TID, \tau_a, \tau_d) \) is a schedule specified by
  - finite set of trip identifiers, \( TID \)
  - \( \tau_a : R \times TID \times V \mapsto T \)
  - \( \tau_d : R \times TID \times V \mapsto T \)
Multimodal Network

(a) Network

(b) Schedule

<table>
<thead>
<tr>
<th>R</th>
<th>TID</th>
<th>Stop</th>
<th>Arrival</th>
<th>Departure</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B$</td>
<td>1</td>
<td>$v_7$</td>
<td>05:31:30</td>
<td>05:32:00</td>
</tr>
<tr>
<td>$B$</td>
<td>1</td>
<td>$v_6$</td>
<td>05:33:00</td>
<td>05:33:00</td>
</tr>
<tr>
<td>$B$</td>
<td>2</td>
<td>$v_7$</td>
<td>06:01:30</td>
<td>06:02:00</td>
</tr>
<tr>
<td>$B$</td>
<td>2</td>
<td>$v_6$</td>
<td>06:03:00</td>
<td>06:03:00</td>
</tr>
<tr>
<td>$B$</td>
<td>2</td>
<td>$v_3$</td>
<td>06:05:00</td>
<td>06:05:30</td>
</tr>
</tbody>
</table>

Figure: Multimodal Network.
Multimodal Network

- A multimodal network can be used to compute isochrones.
- Isochrones: Intuitively, the set of all geometry objects (point, lines, areas) in a SNDB from which a point of interest (query point) can be reached within a given timespan.
**Isochrones**

- **Isochrones**: Intuitively, the set of all geometry objects (point, lines, areas) in a SNDB from which a point of interest (query point) can be reached within a given timespan.
- In multi-modal networks different transporation 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?
Isochrone Example

Example

“From where can I reach FUB in 10 minutes?”

- Credits:
  - Project: http://www.isochrones.inf.unibz.it
  - Demo: http://dbis-isochrone.uibk.ac.at:8080/isochrone/
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 complicates network algorithms significantly