ITS: Network Modeling and shortest path
Outline

• Networks Representation
• Urban Network
• Intersection representation
• Shortest Path Problem (SPP)
  – Example:
    • Dijkstra
    • Pseudocode
    • Proof using linear programming
• Application of SPP
A network is:
- A set of nodes and links
- Nodes = vertices = points
- Links = arcs = edges

All the links are directed = Directed networks

Path is a sequence of links from one node to another.

We say a network is connected if at least there is one path from one node to another one.

Examples:
Urban Road Network

Physical streets
intersections zones

Conceptual nodes &
links & centroids
representation
Intersection Representations

• Simple node representation:
  – No direction differentiations
  – No conflicting movement

• Sub-network representation:
  – Explicit direction
  – Conflicting turns in intersections

• Conceptual representation is not unique and depends on:
  – Type of analysis
  – Data availability
  – Accuracy vs computation time trade-off
Shortest Path Problems (SPP)

• Basic: find a shortest path and the shortest distance between two nodes.
  – This called one to one shortest path problem

• Type of SPP:
  – One to one
  – One to all
  – All to one
  – Many to many
  – All to all

• Shortest also denotes minimum general cost
• They are so many algorithm (they are similar)
• Some of the algorithm uses negative cost only
Example: Dijkstra’s SPP
Dutch computer scientist from Netherlands

Received the 1972 A.M Turing Award, widely considered the most prestigious award in computer science

Know for his many essays on programming

http://www.cs.utexas.edu/~EWD/
Pseudo code for Dijkstra Algorithm

\[
\begin{align*}
\text{dist}[s] & \leftarrow 0 & \text{(distance to source vertex is zero)} \\
\text{for all } v \in V \setminus \{s\} & \text{ do } \text{dist}[v] \leftarrow \infty & \text{(set all other distances to infinity)} \\
S & \leftarrow \emptyset & \text{(S, the set of visited vertices is initially empty)} \\
Q & \leftarrow V & \text{(Q, the queue initially contains all vertices)} \\
\text{while } Q \neq \emptyset & \text{ do } u \leftarrow \text{mindistance}(Q, \text{dist}) & \text{(while the queue is not empty)} \\
& \quad S \leftarrow S \cup \{u\} & \text{(select the element of Q with the min. distance)} \\
& \quad \text{for all } v \in \text{neighbors}[u] & \text{(add u to list of visited vertices)} \\
& \quad \text{ if } \text{dist}[v] > \text{dist}[u] + w(u, v) \\
& \quad \quad \text{ then } \text{dist}[v] \leftarrow \text{dist}[u] + w(u, v) & \text{(if new shortest path found)} \\
& \quad \quad \text{(set new value of shortest path)} \\
& \quad \quad \text{(if desired, add traceback code)} \\
\end{align*}
\]

return dist
Dijkstra’s Algorithm - Example: 1 to 1

\[ C_{ij} \geq 0 \]
Dijkstra’s Algorithm - Example: 1 to 1
Dijkstra’s Algorithm - Example: 1 to 1
Dijkstra’s Algorithm - Example: 1 to 1
Dijkstra’s Algorithm - Example: 1 to 1
Dijkstra’s Algorithm - Example: 1 to 1
Dijkstra’s Algorithm - Example: 1 to 1
Example: Dijkstra’s SPP

- One to one
  - We solved 1 to 1 problem

- One to many
  - Implicitly we solved also 1 to any destination

- Many to many
  - We have just to backward pass

- All to all
  - Keep every vertex as source and run Dijkstra n time.
Example: Dijkstra’s SPP from linear programming view

• $X_{ij} = 1$ if arc $i - j$ is in the shortest path otherwise $X_{ij} = 0$

• Therefore we have to Minimize $\sum \sum C_{ij} X_{ij}$

• In our case
Example: Dijkstra’s SPP from linear programming view

- Apply to our case we will have the following equations: (11 variables)

\[
\begin{align*}
X_{12} + X_{13} &= 1 \\
-X_{12} + X_{24} + X_{25} &= 0 \\
-X_{13} + X_{34} + X_{36} &= 0 \\
-X_{24} - X_{34} + X_{45} + X_{46} + X_{47} &= 0 \\
-X_{25} - X_{45} + X_{57} &= 0 \\
-X_{36} - X_{46} + X_{67} &= 0 \\
-X_{47} - X_{57} - X_{67} &= -1
\end{align*}
\]

- \( \Rightarrow \) uni-modularity \( \Rightarrow X_{ij} = 0,1 \) \( \Rightarrow X_{ij} \geq 0 \)
Example: Dijkstra’s SPP from linear programming view

- Apply to our case we will have the following equations: (11 variables)

\[
\begin{align*}
X_{12} + X_{12} &= 1 \\
-X_{12} + X_{24} + X_{25} &= 0 \\
-X_{13} + X_{34} + X_{36} &= 0 \\
-X_{24} - X_{34} + X_{45} + X_{46} + X_{47} &= 0 \\
-X_{25} - X_{45} + X_{57} &= 0 \\
-X_{36} - X_{46} + X_{67} &= 0 \\
-X_{47} - X_{57} - X_{67} &= -1 \\
\end{align*}
\]

\[
\Rightarrow X_{ij} = 0,1 \quad \Rightarrow X_{ij} \geq 0
\]

Introduce dual variables

Associate

\[\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6, \omega_7\]

We can treat it as a linear programming problem.
Example: Dijkstra’s SPP from linear programming view

- Dual (11 constraint)

  - Maximize \[ \omega_1 - \omega_7 \]

  \[
  \begin{align*}
  \omega_1 - \omega_2 & \leq 15 \\
  \omega_1 - \omega_3 & \leq 20 \\
  \omega_2 - \omega_4 & \leq 10 \\
  \omega_2 - \omega_5 & \leq 25 \\
  \omega_3 - \omega_4 & \leq 15 \\
  \omega_3 - \omega_6 & \leq 20 \\
  \omega_4 - \omega_5 & \leq 20 \\
  \omega_4 - \omega_6 & \leq 15 \\
  \omega_4 - \omega_7 & \leq 30 \\
  \omega_5 - \omega_7 & \leq 10 \\
  \omega_6 - \omega_7 & \leq 20 \\
  \end{align*}
  \]

  \[
  \begin{align*}
  \omega_7 & = 0 \\
  \omega_6 & = 20 \\
  \omega_5 & = 10 \\
  \omega_4 & = 30 \\
  \omega_3 & = 40 \\
  \omega_2 & = 35 \\
  \omega_1 & = 50 \\
  \end{align*}
  \]

  \[ W = \omega_1 - \omega_7 = 50 \]
Example: Dijkstra’s SPP from linear programming view

- Dual (11 constraint)
  - finding the basic variables

\[
\begin{align*}
\omega_1 - \omega_2 &= 50 - 35 = 15 \\
\omega_1 - \omega_3 &= 50 - 40 = 10 \\
\omega_2 - \omega_4 &= 35 - 30 = 5 \\
\omega_2 - \omega_5 &= 35 - 10 = 25 \\
\omega_3 - \omega_4 &= 40 - 30 = 10 \\
\omega_3 - \omega_6 &= 40 - 20 = 20 \\
\omega_4 - \omega_5 &= 30 - 10 = 20 \\
\omega_4 - \omega_6 &= 30 - 20 = 10 \\
\omega_4 - \omega_7 &= 30 - 0 = 30 \\
\omega_5 - \omega_7 &= 10 - 0 = 10 \\
\omega_6 - \omega_7 &= 20 - 0 = 20 \\
\end{align*}
\]
Example: Dijkstra’s SPP from linear programming view

• Using the previous slide Basic variable are:

<table>
<thead>
<tr>
<th>Equation/Inequality</th>
<th>( X_{12} + X_{13} = 1 )</th>
<th>( -X_{12} + X_{24} + X_{25} = 0 )</th>
<th>( -X_{13} + X_{34} + X_{36} = 0 )</th>
<th>( -X_{24} - X_{34} + X_{45} + X_{46} + X_{47} = 0 )</th>
<th>( X_{12} = 1 )</th>
<th>( X_{25} = 1 )</th>
<th>( X_{47} = X_{67} = X_{13} = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \omega_1 - \omega_2 \leq 15 )</td>
<td>Equation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \omega_1 - \omega_3 \leq 20 )</td>
<td>Inequality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \omega_2 - \omega_4 \leq 10 )</td>
<td>Inequality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \omega_2 - \omega_5 \leq 25 )</td>
<td>Equation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \omega_3 - \omega_4 \leq 15 )</td>
<td>Inequality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \omega_3 - \omega_6 \leq 20 )</td>
<td>Equation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \omega_4 - \omega_5 \leq 20 )</td>
<td>Equation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \omega_4 - \omega_6 \leq 15 )</td>
<td>Inequality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \omega_4 - \omega_7 \leq 30 )</td>
<td>Equation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \omega_5 - \omega_7 \leq 10 )</td>
<td>Equation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \omega_6 - \omega_7 \leq 20 )</td>
<td>Equation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ Z = 50 \]
IMPLEMENTATIONS AND RUNNING TIMES

The simplest implementation is to store vertices in an array or linked list. This will produce a running time of

\[ O(|V|^2 + |E|) \]

For sparse graphs, or graphs with very few edges and many nodes, it can be implemented more efficiently storing the graph in an adjacency list using a binary heap or priority queue. This will produce a running time of

\[ O((|E|+|V|) \log |V|) \]
Applications of Dijkstra's Algorithm

- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
- Routing Systems