ITS: Traffic simulation

Network Modeling
Outline

• Networks Representation
• Urban Network
• Intersection representation
• Shortest Path Problem (SPP)
  – Example:
    • Dijkstra
    • Pseudocode
    • Proof using linear programming
• Application of SPP
Network Representation

• 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 Dijkstra Algorithm

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>20</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>20</td>
<td>25</td>
<td>40</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>25</td>
<td>40</td>
<td>40</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>40</td>
<td>40</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- Nodes: 1, 2, 3, 4, 5, 6, 7
- Edges and distances:
  - 1 to 2: 15
  - 1 to 3: 20
  - 2 to 4: 10
  - 2 to 5: 25
  - 3 to 4: 15
  - 3 to 6: 20
  - 4 to 5: 30
  - 4 to 7: 20
  - 5 to 6: 20
  - 5 to 7: 40
  - 6 to 7: 40
  - 6 to 5: 20
  - 7 to 4: 15

The diagram illustrates the network with nodes and distances between them, and the table shows the distances between nodes as per Dijkstra’s algorithm.
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)

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

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

Associate dual variables

Introduce dual variables

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

• Dual (11 constraint)
  
  - Maximize \( \omega_1 - \omega_7 \) \( \omega_j \) in unrestricted in sign \( \omega_i - \omega_j \leq C_{ij} \)

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

For \( \omega_7 = 0 \)

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

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

- **Dual** (11 constraint)
  - finding the basic variables

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

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

• Using the previous slide Basic variable are:

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

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

\[
\begin{align*}
X_{12} &= 1 \\
X_{25} &= 1 \\
X_{57} &= 1 \\
X_{47} &= X_{67} = X_{13} = 0 \\
Z &= 50
\end{align*}
\]
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

\[ O((|E|+|V|) \log |V|) \]
Dijkstra in Action

Source: https://karussell.wordpress.com/2012/12/03/make-your-dijkstra-faster/
Dijkstra in Action

Source: https://karussell.wordpress.com/2012/12/03/make-your-dijkstra-faster/
Applications of Dijkstra's Algorithm

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