Review: Client-Server Queue Model in Data Network

- Arrival rate \( \lambda \) (in voice network, denoted by A):
  - The number of packets coming from the client (incoming link) in a unit time

- Service rate \( \mu \) (in voice network, denoted by D):
  - The number of packets forwarded by the server (outgoing link) in unit time

- Utilization \( \rho = \frac{\lambda}{\mu} \) (in voice network, denoted by E=A/D)
  - If \( \rho > 1 \), the queue’s length is to be infinite, so is the queueing delay
  - If \( \rho = 1 \), the queue’s length is to be infinite, so is the queueing delay if the inter-arrival distributions \( \neq \) the inter-leaving distributions; otherwise, finite queue’s length, no queueing-delay (D/D/1 queue)
  - If \( \rho < 1 \), finite queue’s length, finite queueing delay!
    - Most scenarios belong to this case!
Review: The Drop Algorithm

1. Initially, mark all links as being deletable.
2. Find the most expensive deletable link. If there is a tie, take the link with the lowest utilization. We call this the candidate link for deletion.
3. If such link exists, delete the link and see if the remaining network is feasible (can carry the traffic).
   - If it is feasible, go back to step 2.
   - If not feasible, mark the link “not being deletable” and loop back to step 2.
4. If such link does not exist, terminate.

Graph Theory

- A graph consists of a set of vertices V and a set of edges E: G = (V, E)
- A loop is an edge where both endpoints are the same; vs. circle?
- Two edges are parallel if they have the same end points.
- A graph is simple if there is no loops or parallel edges.
- The degree of a node is the number of edges in the graph that have the node as an endpoint. The degrees of nodes in a graph are important graph property.
- A component of a graph is a maximal connected subgraph, vs. subgraph?
- A Tree is a connected, simple graph without cycles.
- Any tree with n nodes has n-1 edges.
- Trees are optimal network designs when links have very high capacity or enormously expensive, and there is no reliability constraints.
Minimum Cost Network / MST

Problem: Given the distances (km) among 4 cities, build the minimum road to connect them. Assume $1000/km to build the road.

\[
\text{Distance} = \begin{pmatrix}
0 & 1277 & 1168 & 1692 \\
1277 & 0 & 972 & 1052 \\
1168 & 972 & 0 & 562 \\
1692 & 1052 & 562 & 0
\end{pmatrix}
\]

Solution: Choose the shorter distance links first and avoid those links that connect nodes already included. This is Kruskal’s MST Algorithm.

Minimum Cost Network

- Select Charmes-Duval, Bregen-Charmes,
- Avoid Bregen-Duval since both them already included.
- Choose Anagon-Charmes.
- The results is a star topology.

<table>
<thead>
<tr>
<th>End1</th>
<th>End2</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charmes</td>
<td>Duval</td>
<td>562,000</td>
</tr>
<tr>
<td>Bregen</td>
<td>Charmes</td>
<td>972,000</td>
</tr>
<tr>
<td>Bregen</td>
<td>Duval</td>
<td>1,052,000</td>
</tr>
<tr>
<td>Anagon</td>
<td>Charmes</td>
<td>1,168,000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>End1</th>
<th>End2</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charmes</td>
<td>Duval</td>
<td>562,000</td>
</tr>
<tr>
<td>Bregen</td>
<td>Charmes</td>
<td>972,000</td>
</tr>
<tr>
<td>Anagon</td>
<td>Charmes</td>
<td>1,168,000</td>
</tr>
</tbody>
</table>
**Weighted Graphs**

- A tree is a star if only one node has degree greater than 1.
- A tree is a chain if no node has degree greater than 2.
- A graph $G$ is weighted if there is a real number associated with each edge. It is denoted as $(G, w)$.
- The weight of an edge $e$ is denoted as $w(e)$.
- The weighting function $w: E \rightarrow \mathbb{R}^+$. 
- If $G$ is connected, we like to select a connected subgraph with minimum weight
  - Straightforward to prove this subgraph must be a tree

---

**MST Problem and Kruskal’s MST Algorithm**

- Minimum Spanning Tree (MST) Problem
  - Given a weighed graph $G$ where each edge/link has a real weight, select a connected subgraph with the minimum total weight
- Kruskal’s MST Algorithm
  - Check if graph is connected. If not, abort.
  - Sort the edges in ascending order by weight.
  - Mark each node as separate component.
  - Loop on the edges until we accepted $|V|-1$ edges. Let $e$ be the candidate edge.
    - If the endpoints are in different components, merge the two components and accept the edge
    - and increment the # of edges accepted by 1.

*Essential point of Kruskal’s algorithm is that it only adds links that Connect 2 previously unconnected pieces of the graph*
Prim’s MST Algorithm

- Start with a node \( v \) (random), no edge, \( T_0 = (v, \emptyset) \)
  - Add least expensive edge by examining all the nodes that are adjacent to \( v \) and finding the one that is connected to \( v \) by the edge of lowest weight

```
1: Tree *Prim(Graph *G, Node *root) {
2: 
3:   for_each(node, G->nodes) { // initially, each node is out of the tree
4:       node->label = INF; // infinite distance from the trees
5:       node->intree = FALSE;
6:   }
7:   root->label = 0; // root node will be selected in the first loop of Step 11
8:   root->pred = root;
9:   NodesChosen = 0;
10: 
11:   while (Nodeschosen < NumberNodes(G)) {
12:        node=FindMinLabel(G); // Only nodes not in the tree are checked
13:        node->intree = TRUE; // the node with the smallest label is added
14:        ++NodeChosen;
15:        for_each(node2, node->adjacent_nodes) { // scan all adjacent nodes
16:            if (node2->intree == FALSE && // and update their labels
17:                node2->label > weight(edge(node, node2)) { // if closer to the node
18:                node2->pred=node; // Min neighbor dist(node, neighbor)
19:                node2->label=weight(edge(node, node2));
20:            } /* endif */
21:        } /* endfor */
22:    } /* endwhile */
23: 
24:   tree=create_Tree();
25:   BuildTreeFromPreds(G, tree);
26:   return(tree);
27:} /* end Prim */
```
Example of Prim’s MST Algorithm

1. A is added into the tree
   - D.label = 1692, D.pred = A
   - C.label = 1168, C.pred = A
   - B.label = 1277, B.pred = A

2. C is added into the tree (C.pred = A)
   - D.label = 562, D.pred = C
   - B.label = 972, B.pred = C
   - D's info is updated by S.16-19
   - B's info is updated by S.16-19

3. D is added into the tree (D.pred = C)
   - B.label = 972, B.pred = C
   - 972 < 1052, B's info is NOT updated by S.16-19

4. B is added into the tree (B.pred = C)

Optimality of the Two MST Algorithms

Theorem 3.2

*If G is a weighted graph, then both Kruskal’s and Prim’s algorithms produce a minimum spanning tree*
Use Delite to Calculate MSTs

* Start Delite. Select Start | Programs | Delite.
* Select File | Read Input File | Prim.INP.
* The nodes are displayed.
* Select Design | Prim for displaying the result of the Prim MST Algorithm.
* The legend window shows the link_cost of the MST.
* Select Display | Select Map File | USAVH.met for overlay Map file.
* The result display is shown next.
* Select Design | Set Input Parameters | Trace | Yes to generate .trc trace file.
* Use Notepad/wordpad to view the Prim.trc or Prim.INP file.

Use Delite to Calculate MSTs (Cont.)
Tree Designs

- MST is good when
  - Links are highly reliable
  - Networks can tolerate low reliability
  - The number of sites are small

- Either Prim’s or Kruskal’s algorithm gives optimal solution

- MST’s are not good networks to use when the number of nodes is large
  - The reliability of a tree network decreases (probability of failure increases) exponentially as the number of sites increase

Squreworld Counter-example

- 1000 miles x 1000 miles.
- One type of transmission lines $\rightarrow$ 1Mbps
- Cost for two sites with locations $(x_1, y_1)$ and $(x_2, y_2)$ with Euclidian distance $d$ is $(\$1000 + \$10 \times d) / month$
  - $d = ((x_2 - x_1)^2 + (y_2 - y_1)^2)^{\frac{1}{2}}$
- Consider problems with 5, 10, 20, 50, and 100 sites, traffic are normalized with 1 kbps from one site to another. The traffic volumes grows quadratically:

<table>
<thead>
<tr>
<th>N</th>
<th>$2 \times \left(\frac{n}{2}\right)$</th>
<th>Total Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>20</td>
<td>20 kbps</td>
</tr>
<tr>
<td>10</td>
<td>90</td>
<td>90 kbps</td>
</tr>
<tr>
<td>20</td>
<td>380</td>
<td>380 kbps</td>
</tr>
<tr>
<td>50</td>
<td>2450</td>
<td>2450 kbps</td>
</tr>
<tr>
<td>100</td>
<td>8900</td>
<td>8900 kbps</td>
</tr>
</tbody>
</table>
**MST for 5 Node Network**

![MST Diagram]

**Generate Files for Squreworld Counter-example**

- Compile the `c:\Program Files\Delite\Source\gen.c`
- Run `gen <number of nodes> <filename.gen>` to produce the 5, 20, 50, 100 nodes files.
- Run `delite`, select File | Generate Input
- Select the corresponding `.gen` file.
- The monitor will show the files are successfully generated. They include `.REQ` (traffic requirement), `.CST` (cost file.).
- Edit the `.REQ` file, replace the link bandwidth with 1000.
MST for 10 Node Network

Legginess? in Network

° Traffic in the above 10 node MST takes a circuitous route between source and destination.

° To quantify the legginess in the network, we define:

**Definition 3.17**

The number of hops (hop count) between node n1 and n2 is the number of edges in the path chosen by the routing algorithm for the traffic flowing from n1 to n2. If only one path is chosen or if all paths chosen have the same number of edges, then we denote the number by hops(n1, n2)

* there is no requirement that the traffic from n2 to n1 follow the reverse of the path from n1 to n2.
Average # of Hops

- Although the two routes between a pair of nodes can be asymmetric, there is only one path between a pair of nodes in a tree.

**Definition 3.18**

The average number of hops in a network, \( \text{hops} \), is

\[
\text{hops} = \frac{\sum_{n_1,n_2} \text{Traffic}(n_1,n_2) \times \text{hops}(n_1,n_2)}{\sum_{n_1,n_2} \text{Traffic}(n_1,n_2)}
\]

The average number of hops is quite important in evaluating MST designs.

- The sum of the traffic on all links = Total traffic x hops

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-&gt;B:</td>
<td>1000bps</td>
<td>1000bps</td>
<td>1000bps</td>
</tr>
<tr>
<td>A-&gt;C:</td>
<td>1000bps</td>
<td>1000bps</td>
<td>1000bps</td>
</tr>
<tr>
<td>B-&gt;C:</td>
<td>1000bps</td>
<td>1000bps</td>
<td>1000bps</td>
</tr>
</tbody>
</table>

Sum of traffic = 4000 bps

Total traffic: 3000bps

MST for 20 Node Network

[Diagram of MST for 20 Node Network]
MST for 50 Node Network

MST for 100 Node Network
Traffic Volume and Costs

<table>
<thead>
<tr>
<th>n</th>
<th>2xc(n,2)</th>
<th>Total traffic</th>
<th>avg(hops)</th>
<th>sum(flows)</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>20</td>
<td>20,000</td>
<td>1.8</td>
<td>36,000</td>
<td>$13,158</td>
</tr>
<tr>
<td>10</td>
<td>90</td>
<td>90,000</td>
<td>3.1778</td>
<td>286,002</td>
<td>$28,156</td>
</tr>
<tr>
<td>20</td>
<td>380</td>
<td>380,000</td>
<td>4.4158</td>
<td>1,678,004</td>
<td>$48,686</td>
</tr>
<tr>
<td>50</td>
<td>2450</td>
<td>2,450,000</td>
<td>8.5159</td>
<td>20,863,955</td>
<td>$109,925</td>
</tr>
<tr>
<td>100</td>
<td>9900</td>
<td>9,900,000</td>
<td>13.9479</td>
<td>138,084,210</td>
<td>$323,516</td>
</tr>
</tbody>
</table>

- The traffic of 100 node MST grows 5 orders of magnitude from 20kbps of 5 node MST.
- 1.5 order of magnitude comes only from average hop count.
- Problem: MSTs tend to have very long and circuitous paths.

Shortest-Path Trees

**Definition 3.19:**

Given a weighted graph \((G, w)\) and nodes \(n_1\) and \(n_2\), the shortest path from \(n_1\) to \(n_2\) is a path such that \(\Sigma_{e\in P} w(e)\) is a minimum.

* Segments of a shortest path are the shortest paths of their end points.
  (recall the optimality principle)

**Definition 3.20:**

Given a weighted graph \((G, w)\) and a node \(n_1\), a shortest path tree (SPT) rooted at \(n_1\) is a tree such that for any other node \(n_2 \in G\), the path from \(n_1\) to \(n_2\) in the tree \(T\) is a shortest path between the nodes.
Dijkstra’s Algorithm

- Given a connected graph, Prim’s algorithm constructs an MST
  - Given any starting node, the same MST is built if unique

- Dijkstra’s algorithm builds a SPT rooted at a distinguished node
  - 1. Mark every node as unscanned and give each node a label of INF
  - 2. Set the label of the root to 0 and the predecessor of the root to itself. The root will be the only node that is its own predecessor.
  - 3. Loop until you have scanned all the nodes
    - Find the node n with the smallest label. Since the label represents the distance to the root we call it d_min.
    - Mark the node as scanned.
    - Scan all the adjacent nodes m and see if the distance to the root through n is shorter than the distance stored in the label of m. If it is, update the label and update pred [m] = n.
    - Min neighbors (dist(root, neighbor) + dist(neighbor, node))
  - 4. When the loop finishes, we have a tree stored in pred format rooted at the root

---

Example of Dijkstra’s SPT Algorithm

- Initially, A is root (label = 0), other label = INF

- 1. A (root) is added into the tree
  - C.label = 3168, C.pred = A
  - D.label = 1692, D.pred = A
  - B.label = 1277, B.pred = A

- 2. B is added into the tree (B.pred = A)
  - C.label = 2249, C.pred = B
    (C’s info is updated, C->B->A < C->A)
  - D.label = 1692, D.pred = A
    (D’s is not updated, D->A < D->B->A)

- 3. D is added into the tree (D.pred = A)
  - C.label = 2249, C.pred = B
    (C’s is not updated, 2249 < 562 + 1692)

- 4. C is added into the tree (C.pred = B)

SPT is NOT a star, only if the triangle inequality does not hold!
SPT for 20 Node Network

**Metrics:**

- **MST** vs. **SPT**
- Average packet delay (response time) and Utilization

- Let \( \bar{T} = \frac{1}{\mu} \) be average transmission time for a packet over a link (\( \mu \) is the processing/forwarding rate of the link)

- According to M/M/1 queueing theory,
  - Average response time: \( \bar{T} = \frac{1}{1 - \rho} \) (\( \rho \) is the link utilization)

- In a tree with \( \text{hop} \), total average response time is \( D = T \cdot \text{hop} \)

- In a 20 node network
  - Average packet delay for MST = \( \frac{\bar{T} \cdot 5.0316}{1-0.099} = \frac{901}{0.901} \)
  - Average packet delay for SPT = \( \frac{\bar{T} \cdot 1.9000}{1-0.019} = \frac{981.0}{0.981} \)

- **SPT:** higher cost, lower delay, lower reliability (N14 failure!)
  - Not particularly good networks
Prim-Dijkstra Tree

- Both Prim and Dijkstra algorithms start with an initial label, looping over nodes to find one with the smallest label, bringing it into the tree, and finally relabeling all the neighbors.
  - MST: minimize the cost by choosing short links, circuitous tree
    - Select label: Min neighbors \( \text{dist}(\text{node}, \text{neighbor}) \)
  - PST: minimize the path of nodes to the root, expensive tree
    - Select label: Min neighbors \( \text{dist}((\text{root}, \text{neighbor}) + \text{dist}(\text{neighbor}, \text{node})) \)

- Objective: try to find (tradeoff) trees that fall between MST and SPT
  - Prim-Dijkstra Tree selects the following label.
    \[
    \min_{\text{neighbors}} \alpha \times \text{dist}(\text{root}, \text{neighbor}) + \text{dist}(\text{neighbor}, \text{node})
    \]
    Where \( 0 \leq \alpha \leq 1 \) is used to parameterize the algorithm.
    - \( \alpha = 0 \), we build MST
    - \( \alpha = 1 \), we build an SPT from root

Choosing Prim-Dijkstra Trees

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>avg(hops)</th>
<th>Link Delay</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (MST)</td>
<td>13.9479</td>
<td>0.3066</td>
<td>$325,516</td>
</tr>
<tr>
<td>0.1</td>
<td>10.5717</td>
<td>0.1451</td>
<td>$280,162</td>
</tr>
<tr>
<td>0.2</td>
<td>7.8640</td>
<td>0.1067</td>
<td>$247,217</td>
</tr>
<tr>
<td>0.3</td>
<td>6.7762</td>
<td>0.0913</td>
<td>$243,193</td>
</tr>
<tr>
<td>0.4</td>
<td>5.9679</td>
<td>0.0746</td>
<td>$240,099</td>
</tr>
<tr>
<td>0.5</td>
<td>4.6303</td>
<td>0.0598</td>
<td>$253,579</td>
</tr>
<tr>
<td>0.6</td>
<td>3.7063</td>
<td>0.0467</td>
<td>$273,742</td>
</tr>
<tr>
<td>0.7</td>
<td>3.0186</td>
<td>0.0380</td>
<td>$295,012</td>
</tr>
<tr>
<td>0.8</td>
<td>2.2879</td>
<td>0.0277</td>
<td>$378,794</td>
</tr>
<tr>
<td>0.9 (star)</td>
<td>1.9800</td>
<td>0.0233</td>
<td>$453,861</td>
</tr>
</tbody>
</table>

\( \alpha = 0.3 \) and \( \alpha = 0.4 \) give attractive trees.

Problem: to decide which merit consideration and which should be discarded.
Dominance among Designs

- Impose a partial ordering when picking the designs.

**Definition 3.21:**

Given a set $S$ and an operator $\succ$ that maps $S \times S \rightarrow \{\text{TRUE, FALSE}\}$, then we call $S$ a partial ordered set, or poset, if

- For any $s \in S$, $s \succ s$ is FALSE.
- For any $s_1, s_2 \in S$, $s_1 \neq s_2$, if $s_1 \succ s_2$ is True, then $s_2 \succ s_1$ is FALSE.
- If $s_1 \succ s_2$ and $s_2 \succ s_3$ are TRUE, then $s_1 \succ s_3$ is TRUE.

Note that there may exist 2 elements $s_1, s_2 \in S$, s.t. neither $s_1 \succ s_2$ nor $s_2 \succ s_1$ is TRUE. They are called incomparable.

Apply POSET to Pick Designs

**Definition 3.22:**

Suppose $D_1$ has cost $C_1$ and performance $P_1$. Suppose $D_2$ has cost $C_2$ and performance $P_2$.

We will says $D_1$ dominates $D_2$, or $D_1 \succ D_2$, if $C_1 < C_2$ and $P_1 > P_2$.

Use the above definition, we compute the domination partial order in Table 3.12.
**Domination Partial Ordering**

<table>
<thead>
<tr>
<th>Design</th>
<th>Dominates</th>
<th>Link Delay</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>N₀</td>
<td>φ</td>
<td>0.3066</td>
<td>$325,516</td>
</tr>
<tr>
<td>N₁</td>
<td>N₀</td>
<td>0.1451</td>
<td>$280,162</td>
</tr>
<tr>
<td>N₂</td>
<td>N₀, N₁</td>
<td>0.1067</td>
<td>$247,217</td>
</tr>
<tr>
<td>N₃</td>
<td>N₀, N₁, N₂</td>
<td>0.0913</td>
<td>$243,551</td>
</tr>
<tr>
<td>N₄</td>
<td>N₀, N₁</td>
<td>0.0746</td>
<td>$248,650</td>
</tr>
<tr>
<td>N₅</td>
<td>N₀, N₁</td>
<td>0.0598</td>
<td>$253,579</td>
</tr>
<tr>
<td>N₆</td>
<td>N₀, N₁</td>
<td>0.0467</td>
<td>$273,742</td>
</tr>
<tr>
<td>N₇</td>
<td>N₀</td>
<td>0.0380</td>
<td>$295,012</td>
</tr>
<tr>
<td>N₈</td>
<td>φ</td>
<td>0.0277</td>
<td>$378,794</td>
</tr>
<tr>
<td>N₉</td>
<td>φ</td>
<td>0.0233</td>
<td>$453,861</td>
</tr>
</tbody>
</table>

- N₀, N₁, N₂ are dominated by others
  - Remove them from consideration list.

A directed graph

**Consider Marginal Cost of Delay**

- There are still 6 designs to consider.
- The ratio (C₁-C₂)/(P₂-P₁) gives the cost for delay.
- $5099 buy 17 ms delay (from N₃ to N₄)
  - the cost for delay=5099/0.017=305329.
- Between N₅ and N₆, the cost for delay rises 5 times.

<table>
<thead>
<tr>
<th>Design</th>
<th>Link Delay</th>
<th>Cost</th>
<th>Marginal Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>N₃</td>
<td>0.0913</td>
<td>$243,551</td>
<td>*</td>
</tr>
<tr>
<td>N₄</td>
<td>0.0746</td>
<td>$248,650</td>
<td>305,329</td>
</tr>
<tr>
<td>N₅</td>
<td>0.0598</td>
<td>$253,579</td>
<td>333,041</td>
</tr>
<tr>
<td>N₆</td>
<td>0.0467</td>
<td>$273,742</td>
<td>1,539,160</td>
</tr>
<tr>
<td>N₇</td>
<td>0.0380</td>
<td>$295,012</td>
<td>2,444,828</td>
</tr>
<tr>
<td>N₈</td>
<td>0.0277</td>
<td>$378,794</td>
<td>8,134,175</td>
</tr>
<tr>
<td>N₉</td>
<td>0.0233</td>
<td>$453,861</td>
<td>17,060,682</td>
</tr>
</tbody>
</table>
Tours (Rings)

- Sometime trees are too unreliable (one path only between any 2 nodes).
- There are designs that are far more reliable, yet have only one additional link → Tours.
- It is a possible solution of the traveling salesman problem (TSP).

Definition 3.24:
Given a set of vertices \( \{v_1, v_2, \ldots, v_n\} \), a tour \( T \) is a set of \( n \) edges \( E \), s.t.
- each vertex \( v \) has degree 2 and the graph is connected.
- The tour can be represented as a permutation \( (v_{t1}, v_{t2}, \ldots, v_{tn}) \). There are \( n! \) such permutations
- Since cyclically permuting them and the reverse permutation give the same tour, we have \((n-1)!/2\) tours.

Network Reliability

Definition 3.26:
The reliability of a network is the probability that the functioning nodes are connected by working links.

- Assume the probability of each node working is 1 and the probability of a link failing is \( p \). Typically \( p \) is very small. Let \( q = 1 - p \).
- For a 5-node tree (4 links)
  - \( P(\text{no link failures}) = (1-p)^4 \)
  - \( P_{\text{tree}}(\text{failure}) = 1 - (1-p)^4 \approx 4p \) (if \( p \) is small, \( 4p \) dominates the result)
- For a 5-node tour (5 links)
  - \( P(\text{no link failures}) = (1-p)^5 + 5p(1-p)^4 \) (survive 1 link failure)
  - \( P_{\text{tour}}(\text{failure}) = 1 - ((1-p)^5 + 5p(1-p)^4) = 1 - (q^5 + 5pq^4) = 10p^2q^3 + 10pq^2 + 5p^3q^2 + p^5 \approx 10p^2q^3 \) (if \( p \) is small, \( 10p^2q^3 \) dominates)

The chief benefit of a tour network is increased reliability!
**Ptree(failure) vs. Ptour(failure)**

- When \( p = 10^{-6} \), the reliability difference is 5 orders of magnitude

<table>
<thead>
<tr>
<th>( p )</th>
<th>( 4p(\text{tree}) )</th>
<th>( 10p^2q^3(\text{ring}) )</th>
<th>( \text{Ptree/Pring} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.4</td>
<td>0.0729</td>
<td>5.49</td>
</tr>
<tr>
<td>0.01</td>
<td>0.04</td>
<td>0.0009703</td>
<td>41.22</td>
</tr>
<tr>
<td>0.001</td>
<td>0.004</td>
<td>9.97E-06</td>
<td>401.20</td>
</tr>
<tr>
<td>1.00E-04</td>
<td>0.0004</td>
<td>9.997E-08</td>
<td>4,001.20</td>
</tr>
<tr>
<td>1.00E-05</td>
<td>0.00004</td>
<td>9.9997E-10</td>
<td>40,001.20</td>
</tr>
<tr>
<td>1.00E-06</td>
<td>4E-06</td>
<td>1E-11</td>
<td>400,001.20</td>
</tr>
<tr>
<td>1.00E-07</td>
<td>4E-07</td>
<td>1E-13</td>
<td>4,000,001.20</td>
</tr>
</tbody>
</table>

---

**TSP Problem**

**Definition 3.25:**

Given a set of vertices \((v_1, v_2, ..., v_n)\) and a distance function \(d: V \times V \rightarrow \mathbb{R}^+\), the traveling salesman problem is to find the tour \(T\) such that

\[
\sum_{i=1}^{n} d(v_{t_i}, v_{t_{i+1}})
\]

is a minimum. In this we identify \(v_{t_{n+1}}\) with \(v_{t_1}\).

(how should a salesman travel the graph in such a way that each node is visited once and total distance of the tour is minimal.)
Building Tours - Nearest Neighbor Algorithm

- TSP is NP-hard
  - No polynomial-time algorithm to solve the problem exactly.
- Heuristic algorithm – nearest neighbor algorithm
  - Start at a distinguished root node. Set current_node = root node.
  - Loop until all nodes in the tour
    - Go through the node list, find the node closest to the current_node that is not in the tour. Let this node be best_node.
    - Create an edge between current_node and best_node.
    - Set current_node to be best_node.
  - Finally create an edge between the last node and root node.

What is Wrong with This Tour?

- The diagonals are longer than the sides
**Uncrossing the Tour**

First time

2nd time

---

**Creditable Algorithms**

- Creditability problem: N-N produces solutions improvable by hands.
- Definition 3.27: A heuristic optimization algorithm produces a creditable result if the result is a local optimum for the problem. Otherwise, it produces an uncreditable result.
  - Prim’s and Dijkstra’s solve the MST and PST, always creditable
- “We are asking absence of stupidity, not performance.”
- Definition 3.28: A suite of network design problems is a set of triples (Locationsi, Traffici, Costsi) for i=1,…,|S|
- Definition 3.29: A creditability test is a program test (net, traffic, cost) that takes a network problem as input and return OK or FAIL depending on whether or not test() can manipulate net into another valid network of lower costs. (e.g., crossing line test)
- Definition 3.30: Given a suite of network design problems S, a design algorithm A, and a creditability test t(), then

\[
C_s(A) = \frac{|\{net \in S \mid test(net) = OK\}|}{|S|}
\]
Creditability of Tours built by N-N Algorithm

<table>
<thead>
<tr>
<th>Sites</th>
<th>50 trials</th>
<th>500 trials</th>
<th>5000 trials</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>64.00%</td>
<td>59.40%</td>
<td>58.00%</td>
</tr>
<tr>
<td>8</td>
<td>56.00%</td>
<td>52.40%</td>
<td>47.94%</td>
</tr>
<tr>
<td>10</td>
<td>34.00%</td>
<td>37.80%</td>
<td>39.84%</td>
</tr>
<tr>
<td>15</td>
<td>22.00%</td>
<td>21.80%</td>
<td>22.62%</td>
</tr>
<tr>
<td>20</td>
<td>10.00%</td>
<td>13.80%</td>
<td>12.58%</td>
</tr>
<tr>
<td>30</td>
<td>8.00%</td>
<td>3.20%</td>
<td>3.84%</td>
</tr>
<tr>
<td>40</td>
<td>0.00%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test program in appendix B

- Creditability falls dramatically

A More Creditable N-N Heuristic

- More creditable N-N heuristic differs the original in two ways:
  - Improvement 1: find the closest node to the list of nodes in the tour, instead of to the last node in the list of the tour (like Prim’s).
  - Improvement 2: The closest node can be added to any place in the tour, not just append to the end. The insert location minimizes the following formula:

\[
\text{dist}(N_i, \text{best_node}) + \text{dist}(\text{best_node}, N_j) - \text{dist}(N_i, N_j)
\]

- Calculate the cost of the 2 new edges in the graph minus the saving for removing the existing edge; i and j are two adjacent nodes in the partial built tour.
- * reduce one crossing line possibly!

- Intuitively, you try to find a place where \(N_i, \text{best_node}, N_j\) is as flat as possible (follow straight line).

- \(O(2n^2)\) complexity instead of \(O(n^2)\) for basic N-N algorithm.
Examples of MC-NN Heuristic

Results of MC-NN Algorithm

<table>
<thead>
<tr>
<th>Sites</th>
<th>50 trials</th>
<th>500 trials</th>
<th>5000 trials</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>98.00%</td>
<td>94.80%</td>
<td>95.32%</td>
</tr>
<tr>
<td>8</td>
<td>92.00%</td>
<td>92.60%</td>
<td>92.78%</td>
</tr>
<tr>
<td>10</td>
<td>90.00%</td>
<td>90.80%</td>
<td>91.04%</td>
</tr>
<tr>
<td>15</td>
<td>86.00%</td>
<td>82.60%</td>
<td>82.18%</td>
</tr>
<tr>
<td>20</td>
<td>80.00%</td>
<td>72.60%</td>
<td>74.06%</td>
</tr>
<tr>
<td>30</td>
<td>54.00%</td>
<td>60.40%</td>
<td>58.94%</td>
</tr>
<tr>
<td>40</td>
<td>56.00%</td>
<td>55.00%</td>
<td>48.84%</td>
</tr>
</tbody>
</table>

° It performs much better
° Instead selecting nearest neighbor, we can select the furthest neighbors (they are found isolated at the end of N-N algorithms and have poor choice to join the tour).
TSP Tour Do not Scale

- Scalability: average hop count increases almost linearly
- At 100 node tour, for the average hop count, TSP tour is twice as bad as MST

**Theorem 3.3:**
Given uniform traffic any TSP tour of n nodes has \( \text{avg(hops)} = \frac{(n+1)}{4} \)
if n is odd, and \( \frac{n^2}{4(n-1)} \) if n is even.

<table>
<thead>
<tr>
<th>Sites</th>
<th>avg(hops) MST</th>
<th>avg(hops) TSP tour</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1.8000</td>
<td>1.500</td>
</tr>
<tr>
<td>10</td>
<td>3.1778</td>
<td>2.777</td>
</tr>
<tr>
<td>20</td>
<td>4.4158</td>
<td>5.263</td>
</tr>
<tr>
<td>50</td>
<td>8.5159</td>
<td>12.755</td>
</tr>
<tr>
<td>100</td>
<td>13.9479</td>
<td>25.252</td>
</tr>
</tbody>
</table>

2-Connectivity Graph

- Tours are 2-connected. They survive the lost of a node/link.

**Definition 3.31:**
Given a connected graph \( G=(V,E) \), the vertex \( v \) is an articulation point if removing the vertex and all the attached edges disconnects the graph.
* In tree, any node has degree > 1 is a articulation point.

**Definition 3.32:**
If a connected graph \( G=(V,E) \) has no articulation points, then the graph is 2-connected.
Un-obvious Articulation Point

Easy to detect  Not easy to detect

2-Connectivity Theorem

Theorem 3.4:
Suppose G1 = (V1,E1) and G2 = (V2,E2) are 2-connected graphs with V1 ∩ V2 = φ. Let v1, v2 ∈ V1 and v3, v4 ∈ V2. Then the graph G with vertices V1 ∪ V2 and Edges E1 ∪ E2 ∪ (v1, v3) ∪ (v2, v4) is 2-connected.

- **Goal**: produce a design that is 2-connected and has fewer hops
- **Divide and Conquer**:
  - divide the nodes into clusters, find the tour for each cluster, treat a cluster a node and connect clusters with a ring of edges that do not share the same vertex
  - For n clusters, the connected tours will have n more edges (more reliable) and shorter average hop count.
### Original 20-node Tour

#### Parameters
- `ACCESS_LINK_TYPE`: 56
- `ACCESS_TOPOLOGY`: STAR
- `ALPHA`: 0.00
- `AVG_HOPS`: 5.263
- `CLUSTER_MODE`: THRESH
- `COST_FILTER`: 30000
- `CS_DODESIGN`: FALSE
- `CS_MAXDIAMETER`: 1.00

#### Function
- **Links**: F1024
- **Nodes**: End Node, Backbone, Open Node

### 2 Cluster Tour

#### Parameters
- `NET20-M`

#### Function
- **Links**: F1024
- **Nodes**: End Node, Backbone, Open Node
Join the Tours

2 more links reduced average hops from 5.263 to 3.895

Connecting multiple 2-connected clusters

Suppose that \( G = (V, E) \) is a 2-connected graph with \( |V| > 2 \). Suppose that each node \( v \in V \) is replaced by a 2-connected graph \( G_v \). Suppose each edge \( e = (u, w) \in E \) is replaced by an edge \( e' \) from \( u' \in G_u \) to \( v' \in G_w \). Then if no 2 of these replacement edges have a common vertex, the graph \( H = (\bigcup V_i, \bigcup E_i \cup E') \) is a 2-connected graph.
4-cluster Design for 50 nodes

Single tour:
HOP = 12.755

4-cluster:
HOP = 6.8971

Homework #5

Problem 3.4: ...weights of all edges are unique (different).
* use both Kruskal’s algorithm, and Prim’s algorithm to prove

Problem 3.7:
1. Explain why for the 20-node random problem avg(hops)_{SPT} = 1.9000.
2. Prove for a start on n nodes, S, that avg(hops)_{SPT} = 2 - 2/n.

Problem 3.8:
1. A ring can take the failure of any link and continue to operate. Using the same 5 nodes as in Figure 3.9 (not 3.15), design a network that can withstand the loss of any 2 links and continue to operate.
2. Extend the table 3.15 (not 3.11) to show the P (failure) for your network.
* only consider the dominant term (since the failure probability of each link is very small)

Problem 3.9