Network Analysis:
Minimum Spanning Tree,
The Shortest Path Problem,
Maximal Flow Problem
Definitions

- A **network** consists of a set of **nodes** and a set of **arcs** connecting the nodes
- **Nodes** are also called **vertices** or **points**
- **Arcs** are also called **edges**, **links**, **lines**, or **branches**
- A **network** is also called a **graph** in mathematics
Graphs

A graph is represented by the notation:

\[ G = (N, A), \]

where, \( N \equiv \) the set of nodes

and, \( A \equiv \) the set of arcs in graph \( G \)
Arcs

- An arc is **undirected** if it does not have a specific orientation
- It is **directed** if it has exactly one orientation
- It is **bidirected** if it has two orientations
Conventions

- Circles are used to represent nodes
- Lines are used to indicate arcs
- Arrowheads are used to indicate orientation in a network

Directed Arc

Undirected Arc

Bidirected Arc
Conventions

- Arcs are represented by the ordered two-tuples of the nodes at its endpoints, e.g., \((i,j)\) or \((j,i)\).
- The order of the two-tuples indicates direction of flow through the arc, e.g., \((i,j)\) implies flow from node \(i\) to node \(j\) along arc \((i,j)\).
Paths

- When two nodes are not connected by a single arc, there may be a path between the two nodes through the network.
- A path is a sequence of distinct arcs connecting two nodes of a network.
Paths

- A **directed path** from node i to node j is a sequence of connecting arcs whose direction, if any, is **toward** node j.

- An **undirected path** from node i to node j is a sequence of connecting arcs whose direction, if any, can be **either** toward or away from node j.
Cycles

- A path that begins and ends on the same node is a cycle.
- In a directed network, a cycle is either a directed cycle or an undirected cycle, depending upon whether the path involved is a directed or undirected path.
Connectedness

- Two nodes are said to be connected if the network contains at least one undirected path between them.
- Note, the path does not need to be directed even if the network is directed.
- A connected network (or connected graph) is one where every pair of nodes is connected.
Trees

• If $G=(N,A)$, we may define a subset of $G$ as $\overline{G} = (\overline{N}, \overline{A})$

• A **tree** is defined as a connecting subset $\overline{G}$ containing no **undirected cycles**

• A **spanning tree** is a connected network for all $N$ nodes which contains no undirected cycles
Spanning Trees

- Every spanning tree has exactly \((N-1)\) arcs.
- This is the minimum number of arcs needed to have a connected network, and the maximum number possible without having undirected cycles.
Arc Capacity and Node Types

• Arc capacity = maximum amount of flow that can be carried on directed arc

• A supply node (or source node) has the property that flow out > flow into the node

• A demand node (or sink) has the property that flow into the node > flow out of the node

• A transshipment node (or intermediate node) satisfies conservation of flow, i.e., flow in = flow out
The Shortest Path Problem
(or Shortest Route Problem)

- **Objective** - find the shortest path through a network from a source node to a sink node
- **Algorithm** developed by Dijkstra (1959) to solve this problem
- **It can also be formulated as a linear programming problem**
LP Formulation

\[ Min \ Z = \sum_i \sum_j c_{ij} f_{ij} \]

s.t. \[ \sum_j f_{ij} - \sum_j f_{ji} = 1 \] for \( i \in \) Source node

\[ \sum_j f_{ij} - \sum_j f_{ji} = 0 \] for \( i \in \) Intermediate node

\[ \sum_j f_{ji} - \sum_j f_{ij} = -1 \] for \( j \in \) Sink node

\[ f_{ij} \geq 0 \]

Where \( f_{ij} \) = flow of one unit from node \( i \) to node \( j \).

And, \( c_{ij} \) = cost or time required to move one unit from node \( i \) to node \( j \)
Example

Find the shortest path through the following network from O to T

Diagram of the network with labeled edges.
**Example**

\[
\begin{align*}
\text{Min } Z &= 2f_{OA} + 5f_{OB} + 4f_{OC} + 2f_{AB} + 7f_{AD} + 4f_{BD} \\
&\quad + 3f_{BE} + f_{CB} + 4f_{CE} + f_{ED} + 7f_{ET} + 5f_{DT} \\
\text{s.t.} \quad (f_{OA} + f_{OB} + f_{OC}) &= 1 \\
(f_{AB} + f_{AD}) - f_{OA} &= 0 \\
(f_{BD} + f_{BE}) - (f_{AB} + f_{OB} + f_{CB}) &= 0 \\
(f_{CB} + f_{CE}) - f_{OC} &= 0 \\
(f_{ED} + f_{ET}) - (f_{BE} + f_{CE}) &= 0 \\
(f_{DT}) - (f_{AD} + f_{BD} + f_{ED}) &= 0 \\
-(f_{DT} + f_{ET}) &= -1 \\
&\quad f_{ij} \geq 0
\end{align*}
\]
Example

- The preceding LP formulation has 12 variables, one for each arc in the network.
- It has 7 functional constraints, one for each node in the network.
- It was solved in automatic mode using Mathprog software to get: Min Z = 13, for \( f_{OA} = f_{AB} = f_{BE} = f_{ED} = f_{DT} = 1 \), and all other \( f_{ij} = 0 \).
Example

A shortest path through the network from O to T, as determined from LP, is shown by the red arcs below.
Dijkstra’s Algorithm
(Undirected Arcs)

- **Objective of the $n^{th}$ iteration**: Find the $n^{th}$ nearest node to the source (repeat for $n=1, 2, ..., $ until the $n^{th}$ nearest node is the sink)
- **Input for $n^{th}$ iteration**: $(n-1)$ nearest nodes to the source (found in previous iterations), including their shortest path and distance from the source. These nodes, plus the origin, are called *solved nodes*, the others are *unsolved nodes*. 
Dijkstra’s Algorithm

(Undirected Arcs)

- **Candidates for \(n^{th}\) nearest node**: Each solved node directly connected by an arc to one or more unsolved nodes provides one candidate -- the unsolved node with the shortest connecting link (ties provide additional candidates)
Dijkstra’s Algorithm

(Undirected Arcs)

- *Calculation of the $n^{th}$ nearest node*: For each such solved node and its candidate, add the distance between them and the distance of the shortest path from the source to this solved node. The candidate with the smallest such total distance is the $n^{th}$ nearest node (ties provide additional solved nodes), and its shortest path is the one generating this distance.
Example

Find the shortest path through the following network from O to T
## Example

<table>
<thead>
<tr>
<th>n</th>
<th>Solved Nodes Directly Connected to Unsolved Nodes</th>
<th>Closest Connected Unsolved Node</th>
<th>Total Distance Involved</th>
<th>nth Nearest Node</th>
<th>Minimum Distance</th>
<th>Last Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>O</td>
<td>A</td>
<td>2</td>
<td>A</td>
<td>2</td>
<td>OA</td>
</tr>
<tr>
<td></td>
<td>2,3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>O</td>
<td>C</td>
<td>4</td>
<td>C</td>
<td>4</td>
<td>OC</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>2 + 2 = 4</td>
<td>B</td>
<td>4</td>
<td>AB</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>D</td>
<td>2 + 7 = 9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>E</td>
<td>4 + 3 = 7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>E</td>
<td>4 + 4 = 8</td>
<td>E</td>
<td>7</td>
<td>BE</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>D</td>
<td>2 + 7 = 9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>D</td>
<td>4 + 4 = 8</td>
<td>D</td>
<td>8</td>
<td>BD</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>D</td>
<td>7 + 1 = 8</td>
<td>D</td>
<td>8</td>
<td>ED</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>T</td>
<td>8 + 5 = 13</td>
<td>T</td>
<td>13</td>
<td>DT</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>T</td>
<td>7 + 7 = 14</td>
<td>T</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>
Example

- To determine the shortest path, trace backwards from T to O to find:
  T - D - E - B - A - O, or T - D - B - A - O
- So, the two shortest paths at a total distance of 13 miles are:
  - O - A - B - E - D - T, or O - A - B - D - T
Example

Shortest paths shown as dashed lines
The Shortest Path Problem
(Acyclic Algorithm)

For networks which have directed arcs, but no directed loops (cycles), i.e., an acyclic network, a recursive algorithm which employs a labeling procedure for each node is used to find the shortest path from a source to a sink node.
Nomenclature

- $u_i$ = shortest distance from the source node to an immediately preceding node to node i
- So, if there are n nodes in the network

\[
  u_j = \min_i \{u_i + d_{ij}\}
\]

where $i$ is predecessor node linked to node $j$ by a direct arc, and

$d_{ij} = \text{distance between node } j \text{ and its predecessor } i$
Labeling Procedure

- The labeling procedure associates a unique label with any node j:

  \[ \text{node j label} = [u_j, n] \]

  where \( n \) is the node immediately preceding j that leads to the shortest distance \( u_j \), i.e.,

  \[ u_j = \min_i \left\{ u_i + d_{ij} \right\} \]

  \[ = u_n + d_{nj} \]

  By definition, the label of the source node is \( [0, --] \), and \( u_1 = 0 \)
Example

Use the acyclic algorithm to label the nodes and find the shortest route from node 1 to node 6 in the following network.
## Example

<table>
<thead>
<tr>
<th>Node j</th>
<th>Computation of $u_j$</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$u_1 = 0$</td>
<td>[0, --]</td>
</tr>
<tr>
<td>2</td>
<td>$u_2 = 0 + 3 = 3$, from 1</td>
<td>[3, 1]</td>
</tr>
<tr>
<td>3</td>
<td>$u_3 = \min_{1,2} {0 + 2; 3 + 5} = 2$, from 1</td>
<td>[2, 1]</td>
</tr>
<tr>
<td>4</td>
<td>$u_4 = \min_{2,3} {3 + 7; 2 + 4} = 6$, from 3</td>
<td>[6, 3]</td>
</tr>
<tr>
<td>5</td>
<td>$u_5 = \min_{2,4} {3 + 5; 6 + 6} = 8$, from 2</td>
<td>[8, 2]</td>
</tr>
<tr>
<td>6</td>
<td>$u_6 = \min_{4,5} {6 + 3; 8 + 4} = 9$, from 4</td>
<td>[9, 4]</td>
</tr>
</tbody>
</table>
Example

• The shortest route through the network is obtained by starting at node 6 and tracing backward through the nodes using the labels generated in the preceding table:
  (6) - [9,4] - (4) - [6,3] - (3) - [2,1] - (1)

• So the optimal node sequence is:
  1 - 3 - 4 - 6, and the minimum distance = 9
The Shortest Path Problem
(Cyclic Algorithm)

Dijkstra’s Algorithm

• If a network has one or more directed loops, i.e., cycles, the acyclic algorithm will not find the shortest path from source to sink.

• The cyclic algorithm permits reevaluation of nodes in a cycle through the use of two types of labels: temporary and permanent.
The labels look the same as for the acyclic network, i.e., \([d, n]\), where \(d\) is the shortest distance so far available at the current node, and \(n\) is the immediate predecessor node responsible for realizing the distance \(d\).
Cyclic Algorithm

1. Assign the permanent label $[0, --]$ to the source node.

2. Consider all nodes that can be reached directly from the source node and compute their temporary labels.

3. The next permanent label is selected from among all temporary labels as the one having the minimum $d$ in label $[d, n]$, ties are broken arbitrarily.
Cyclic Algorithm

4. Repeat step 3 until the last (sink) node has been given a permanent label. In each iteration, a temporary label of a node may be changed only if the new label yields a smaller distance, $d$. 
Example

Alter the previous example slightly so that there are cycles, and re-solve for the shortest path using Dijkstra’s Algorithm

![Graph diagram]

**Example Solution**

1. Start at node 1
2. Move to node 2
3. Move to node 3
4. Move to node 4
5. Move to node 5
6. Move to node 6

The shortest path is 1 → 2 → 3 → 4 → 5 → 6 with a total weight of 17.
Example

- The shortest route through the network is obtained by starting at node 6 and tracing backward through the nodes using the permanent labels generated in the algorithm:
  \[(6) - [9,4]^* - (4) - [6,3]^* - (3) - [2,1]^* - (1)\]

- So the optimal node sequence remains:
  \[1 - 3 - 4 - 6\], and the minimum distance = 9
Minimum Spanning Tree

• This problem involves choosing for the network the links that have the shortest total length while providing a path between each pair of nodes.

• These links must be chosen so that the resulting network forms a tree that spans all the given nodes of the network, i.e., a spanning tree with minimum total length of the links.
MST Algorithm

1. Start with any node and join it to its closest node in the network. The resulting two nodes now form a connected set, and the remaining nodes comprise the unconnected set.

2. Choose a node from the unconnected set that is closest to any node in the connected set and add it to the connected set.
MST Algorithm

3. Redefine the connected and unconnected sets accordingly. Repeat the process until the connected set includes all the nodes in the network.

4. Ties may be broken arbitrarily; however, ties indicate the existence of alternative minimal spanning trees!
Beginning with node 1, compute a MST for this network
Example

Let $c =$ the set of connected nodes at any iteration

$\overline{c} =$ the set of unconnected nodes at any iteration

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>$c$</th>
<th>$\overline{c}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${1}$</td>
<td>${2,3,4,5,6}$</td>
</tr>
<tr>
<td>2</td>
<td>${1,2}$</td>
<td>${3,4,5,6}$</td>
</tr>
<tr>
<td>3</td>
<td>${1,2,5}$</td>
<td>${3,4,6}$</td>
</tr>
<tr>
<td>4</td>
<td>${1,2,4,5}$</td>
<td>${3,6}$</td>
</tr>
<tr>
<td>5</td>
<td>${1,2,4,5,6}$</td>
<td>${3}$</td>
</tr>
<tr>
<td>6</td>
<td>${1,2,3,4,5,6}$</td>
<td>${}$</td>
</tr>
</tbody>
</table>
Example

Minimal Spanning Tree

Min. Total Distance = 16
Kruskal’s algorithm

1. Arrange all edges in a list (L) in non-decreasing order
2. Select edges from L, and include that in set T, avoid cycle.
3. Repeat 3 until T becomes a tree that covers all vertices
Kruskal’s Algorithm
Kruskal’s Algorithm

\begin{tabular}{|c|c|}
\hline
\{1,2\} & 12 \\
\{3,4\} & 12 \\
\{1,8\} & 13 \\
\{4,5\} & 13 \\
\{2,7\} & 14 \\
\{3,6\} & 14 \\
\{7,8\} & 14 \\
\{5,6\} & 14 \\
\{5,8\} & 15 \\
\{6,7\} & 15 \\
\{1,4\} & 16 \\
\{2,3\} & 16 \\
\hline
\end{tabular}
Kruskal’s Algorithm

\[
\begin{array}{c|c}
\{1,2\} & 12 \\
\{3,4\} & 12 \\
\{1,8\} & 13 \\
\{4,5\} & 13 \\
\{2,7\} & 14 \\
\{3,6\} & 14 \\
\{7,8\} & 14 \\
\{5,6\} & 14 \\
\{5,8\} & 15 \\
\{6,7\} & 15 \\
\{1,4\} & 16 \\
\{2,3\} & 16 \\
\end{array}
\]
Kruskal’s Algorithm

\[
\begin{array}{|c|c|}
\hline
\{1,2\} & 12 \\
\{3,4\} & 12 \\
\{1,8\} & 13 \\
\{4,5\} & 13 \\
\{2,7\} & 14 \\
\{3,6\} & 14 \\
\{7,8\} & 14 \\
\{5,6\} & 14 \\
\{5,8\} & 15 \\
\{6,7\} & 15 \\
\{1,4\} & 16 \\
\{2,3\} & 16 \\
\hline
\end{array}
\]
Kruskal’s Algorithm

\[
\begin{array}{|c|c|}
\hline
\{1,2\} & 12 \\
\{3,4\} & 12 \\
\{1,8\} & 13 \\
\{4,5\} & 13 \\
\{2,7\} & 14 \\
\{3,6\} & 14 \\
\{7,8\} & 14 \\
\{5,6\} & 14 \\
\{5,8\} & 15 \\
\{6,7\} & 15 \\
\{1,4\} & 16 \\
\{2,3\} & 16 \\
\hline
\end{array}
\]
Kruskal’s Algorithm

\[
\begin{align*}
\{1,2\} & : 12 \\
\{3,4\} & : 12 \\
\{1,8\} & : 13 \\
\{4,5\} & : 13 \\
\{2,7\} & : 14 \\
\{3,6\} & : 14 \\
\{7,8\} & : 14 \\
\{5,6\} & : 14 \\
\{5,8\} & : 15 \\
\{6,7\} & : 15 \\
\{1,4\} & : 16 \\
\{2,3\} & : 16
\end{align*}
\]
Kruskal’s Algorithm

---

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1,2}</td>
<td>12</td>
</tr>
<tr>
<td>{3,4}</td>
<td>12</td>
</tr>
<tr>
<td>{1,8}</td>
<td>13</td>
</tr>
<tr>
<td>{4,5}</td>
<td>13</td>
</tr>
<tr>
<td>{2,7}</td>
<td>14</td>
</tr>
<tr>
<td>{3,6}</td>
<td>14</td>
</tr>
<tr>
<td>{7,8}</td>
<td>14</td>
</tr>
<tr>
<td>{5,6}</td>
<td>14</td>
</tr>
<tr>
<td>{5,8}</td>
<td>15</td>
</tr>
<tr>
<td>{6,7}</td>
<td>15</td>
</tr>
<tr>
<td>{1,4}</td>
<td>16</td>
</tr>
<tr>
<td>{2,3}</td>
<td>16</td>
</tr>
</tbody>
</table>
Kruskal’s Algorithm

Skip \{7,8\} to avoid cycle

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1,2}</td>
<td>12</td>
</tr>
<tr>
<td>{3,4}</td>
<td>12</td>
</tr>
<tr>
<td>{1,8}</td>
<td>13</td>
</tr>
<tr>
<td>{4,5}</td>
<td>13</td>
</tr>
<tr>
<td>{2,7}</td>
<td>14</td>
</tr>
<tr>
<td>{3,6}</td>
<td>14</td>
</tr>
<tr>
<td>{7,8}</td>
<td>14</td>
</tr>
<tr>
<td>{5,6}</td>
<td>14</td>
</tr>
<tr>
<td>{5,8}</td>
<td>15</td>
</tr>
<tr>
<td>{6,7}</td>
<td>15</td>
</tr>
<tr>
<td>{1,4}</td>
<td>16</td>
</tr>
<tr>
<td>{2,3}</td>
<td>16</td>
</tr>
</tbody>
</table>

Skip
Kruskal’s Algorithm

Skip \{5,6\} to avoid cycle

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1,2}</td>
<td>12</td>
</tr>
<tr>
<td>{3,4}</td>
<td>12</td>
</tr>
<tr>
<td>{1,8}</td>
<td>13</td>
</tr>
<tr>
<td>{4,5}</td>
<td>13</td>
</tr>
<tr>
<td>{2,7}</td>
<td>14</td>
</tr>
<tr>
<td>{3,6}</td>
<td>14</td>
</tr>
<tr>
<td>{7,8}</td>
<td>14</td>
</tr>
<tr>
<td>{5,6}</td>
<td>14</td>
</tr>
<tr>
<td>{5,8}</td>
<td>15</td>
</tr>
<tr>
<td>{6,7}</td>
<td>15</td>
</tr>
<tr>
<td>{1,4}</td>
<td>16</td>
</tr>
<tr>
<td>{2,3}</td>
<td>16</td>
</tr>
</tbody>
</table>

Skip {5,6} to avoid cycle
Kruskal’s Algorithm

MST is formed
Prim’s algorithm

- Start form any arbitrary vertex
- Find the edge that has minimum weight from all known vertices
- Stop when the tree covers all vertices
Prim’s algorithm

Start from any arbitrary vertex
Prim’s algorithm

The best choice is \{1,2\}

What’s next?

Weight: 12
Prim’s algorithm

After \{1,8\} we may choose \{2,7\} or \{7,8\}

There are more than one MST

What’s next?

Weight: 12 + 13
Prim’s algorithm

Let’s choose \{2, 7\} at this moment

We are free to choose \{5,8\} or \{6,7\} but not \{7,8\} because we need to avoid cycle
Prim’s algorithm

Let’s choose \{5,8\} at this moment

The best choice is now \{4,5\}
Prim’s algorithm

The best choice is now \{4,3\}
Prim’s algorithm

The best choice is now \{3, 6\} or \{5, 6\}.
Prim’s algorithm

A MST is formed
Weight = 93
Prim’s algorithm – more complex
Prim’s algorithm – more complex
Prim’s algorithm
Prim’s algorithm
Prim’s algorithm
Prim’s algorithm
Prim’s algorithm

![Graph with Prim's algorithm example]
Prim’s algorithm
Prim’s algorithm
Compare Prim and Kruskal

- Both have the same output → MST
- Kruskal’s begins with forest and merge into a tree
- Prim’s always stays as a tree
- If you don’t know all the weight on edges → use Prim’s algorithm
- If you only need partial solution on the graph → use Prim’s algorithm
Compare Prim and Kruskal

Complexity

Kruskal: $O(N\log N)$
- comparison sort for edges

Prim: $O(N\log N)$
- search the least weight edge for every vertex
Why do we need MST?

• a reasonable way for clustering points in space into natural groups
• can be used to give approximate solutions to hard problems
Minimizing costs

• Suppose you want to provide solution to:
  – electric power, Water, telephone lines, network setup

• To minimize cost, you could connect locations using MST

• However, MST is not necessary the shortest path and it does not apply to cycle
Maximal Flow Problem

- For networks with one source node and one sink node
- All other nodes are transshipment nodes
- Each arc has a maximum allowable capacity
- **Objective:** Find the maximum amount of flow between the source and the sink through the network
Solution Procedure

• There are a number of “labeling” procedures
• For small problems, the max-flow, min-cut theorem works well
Max-Flow Min-Cut Theorem

The maximum amount of flow from the source node to the sink node equals the minimum cut value for all cuts of the network.
Definitions

- A **cut** (or cut-set) is any set of directed arcs containing at least one arc from every directed path from the source node to the sink node.

- The **cut value** is the sum of the arc capacities of the arcs (in the specified direction) of the cut.
Example

Find the maximal flow through the following network using the max-flow min-cut theorem approach. Labels on the arcs are maximum flow capacities.
## Example

<table>
<thead>
<tr>
<th>Source to Sink Directed Paths</th>
<th>Arcs in the Directed Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – 2 – 5</td>
<td>(1,2), (2,5)</td>
</tr>
<tr>
<td>1 – 2 – 4 – 5</td>
<td>(1,2),(2,4),(4,5)</td>
</tr>
<tr>
<td>1 – 4 – 5</td>
<td>(1,4),(4,5)</td>
</tr>
<tr>
<td>1 – 3 – 4 – 5</td>
<td>(1,3),(3,4),(4,5)</td>
</tr>
<tr>
<td>1 – 3 – 5</td>
<td>(1,3),(3,5)</td>
</tr>
</tbody>
</table>
## Example

<table>
<thead>
<tr>
<th>Cutsets</th>
<th>Cut Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,2),(1,3),(1,4)</td>
<td>60</td>
</tr>
<tr>
<td>(1,2),(1,4),(3,4),(3,5)</td>
<td>85</td>
</tr>
<tr>
<td>(1,2),(1,3),(4,5)</td>
<td>45</td>
</tr>
<tr>
<td>(2,5),(2,4),(1,4),(1,3)</td>
<td>73</td>
</tr>
<tr>
<td>(2,5),(2,4),(1,4),(3,4),(3,5)</td>
<td>68</td>
</tr>
<tr>
<td>(1,2),(4,5),(3,5)</td>
<td>70</td>
</tr>
<tr>
<td>(2,5),(4,5),(3,5)</td>
<td>63</td>
</tr>
<tr>
<td>(1,3),(2,5),(4,5)</td>
<td>53</td>
</tr>
</tbody>
</table>

Min-Cut
Observation

The difficulty in applying the max-flow min-cut theorem is in making sure that all cutsets have been enumerated and evaluated!
Maximum Flow

• Max-flow problem:
  – A directed graph $G=\langle V,E \rangle$, a capacity function on each edge $c(u,v) \geq 0$ and a source $s$ and a sink $t$. A flow is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:
    • Capacity constraints: for all $u,v \in V$, $f(u,v) \leq c(u,v)$.
    • Skew symmetry: for all $u,v \in V$, $f(u,v) = -f(v,u)$.
    • Flow conservation: for all $u \in V-\{s,t\}$, $\sum_{v \in V} f(u,v) = 0$, or to say, total flow out of a vertex other $s$ or $t$ is 0, or to say, how much comes in, also that much comes out.
  – Find a maximum flow from $s$ to $t$. 
Figure 26.1  (a) A flow network $G = (V, E)$ for the Lucky Pack Company’s trucking problem. The Vancouver factory is the source $s$, and the Winnipeg warehouse is the sink $t$. Pucks are shipped through intermediate cities, but only $c(u, v)$ crates per day can go from city $u$ to city $v$. Each edge is labeled with its capacity. (b) A flow $f$ in $G$ with value $|f| = 19$. Only positive flows are shown. If $f(u, v) > 0$, edge $(u, v)$ is labeled by $f(u, v)/c(u, v)$. (The slash notation is used merely to separate the flow and capacity; it does not indicate division.) If $f(u, v) \leq 0$, edge $(u, v)$ is labeled only by its capacity.
Ford-Fulkerson method

- Contains several algorithms:
  - Residue networks
  - Augmenting paths

```
FORD-FULKERSON-METHOD(G, s, t)
1   initialize flow f to 0
2   while there exists an augmenting path p
3      do augment flow f along p
4   return f
```
Residual Networks

- Given a flow network $G=<V,E>$ and a flow $f$,
  - the residual network of $G$ induced by $f$ is $G_f=<V,E_f>$ where $E_f=\{(u,v)\in V\times V: c_f(u,v)=c(u,v)-f(u,v), \text{ and } c_f(u,v)>0\}$
  - a network with left capacity >0, also a flow network.
Residual network and augmenting path

Figure 26.3 (a) The flow network $G$ and flow $f$ of Figure 26.1(b). (b) The residual network $G_f$ with augmenting path $p$ shaded; its residual capacity is $c_f(p) = c(v_2, v_3) = 4$. (c) The flow in $G$ that results from augmenting along path $p$ by its residual capacity 4. (d) The residual network induced by the flow in (c).
Residual network and flow theorem

- **Lemma:**
  - Let $G=<V,E>$ be a flow network with source $s$ and sink $t$, and let $f$ be a flow,
  - Let $G_f$ be the residual network of $G$ induced by $f$, and let $f'$ be a flow of $G_f$.
  - Define the flow sum: $f+f'$ as:
    - $(f+f')(u,v)=f(u,v)+f'(u,v)$, then
    - $f+f'$ is a flow in $G$ with value $|f+f'|=|f|+|f'|$.

- **Proof:**
  - Capacity constraint, skew symmetry, and flow conservation and finally $|f+f'|=|f|+|f'|$. 
Augmenting paths

- Let $G=<V,E>$ be a flow network with source $s$ and sink $t$, and let $f$ be a flow,
- An augmenting path $p$ in $G$ is a simple path from $s$ to $t$ in $G_f$, the residual network of $G$ induced by $f$.
- Each edge $(u,v)$ on an augmenting path admits some additional positive flow from $u$ to $v$ without violating the capacity constraint.
- Define residual capacity of $p$ is the maximum amount we can increase the flow:
  - $c_f(p)=\min\{c_f(u,v): (u,v) \text{ is on } p.\}$
Augmenting path

• **Lemma:**

  - Let $G=<V,E>$ be a flow network with source $s$ and sink $t$, let $f$ be a flow, and let $p$ be an augmenting path in $G$. Define $f_p : V \times V \to R$ by:
    
    $f_p(u,v) = \begin{cases} 
    c_f(p) & \text{if } (u,v) \text{ is on } p. \\
    -c_f(p) & \text{if } (v,u) \text{ is on } p. \\
    0 & \text{otherwise}
    \end{cases}$

  - Then $f_p$ is a flow in $G_f$ with value $|f_p| = c_f(p) > 0$.

• **Corollary:**

  - Define $f' = f + f_p$, then $f'$ is a flow with value $|f'| = |f| + |f_p| > |f|$. 
Basic Ford-Fulkerson algorithm

\begin{algorithm}
\textbf{Ford-Fulkerson}(G, s, t)
1 \textbf{for} each edge \((u, v) \in E[G]\) \textbf{do} \(f[u, v] \leftarrow 0\)
2 \hspace{1cm} \(f[v, u] \leftarrow 0\)
3 \textbf{while} there exists a path \(p\) from \(s\) to \(t\) in the residual network \(G_f\) \textbf{do} \(c_f(p) \leftarrow \min \{c_f(u, v) : (u, v) \text{ is in } p\}\)
4 \hspace{1cm} \textbf{for} each edge \((u, v)\) in \(p\) \textbf{do} \(f[u, v] \leftarrow f[u, v] + c_f(p)\)
5 \hspace{2cm} \(f[v, u] \leftarrow -f[u, v]\)
\end{algorithm}

Running time: if capacities are in irrational numbers, the algorithm may not terminate. Otherwise, \(O(|E||f^*|)\) where \(f^*\) is the maximum flow found by the algorithm: \textbf{while} loop runs \(f^*\) times, increasing \(f^*\) by one each loop, finding an augmenting path using depth-first search or breadth-first search costs \(|E|\).
Execution of Ford-Fulkerson

Figure 26.5 The execution of the basic Ford-Fulkerson algorithm. (a)-(d) Successive iterations of the while loop. The left side of each part shows the residual network $G_f$ from line 4 with a shaded augmenting path $p$. The right side of each part shows the new flow $f$ that results from adding $f_p$ to $f$. The residual network in (a) is the input network $G$. (e) The residual network at the last while loop test. It has no augmenting paths, and the flow $f$ shown in (d) is therefore a maximum flow.
An example of loop $|f^*|$ times

Figure 26.6  (a) A flow network for which FORD-FULKERSON can take $\Theta(E \cdot |f^*|)$ time, where $f^*$ is a maximum flow, shown here with $|f^*| = 2,000,000$. An augmenting path with residual capacity 1 is shown. (b) The resulting residual network. Another augmenting path with residual capacity 1 is shown. (c) The resulting residual network.

Note: if finding an augmenting path uses breadth-first search, i.e., each augmenting path is a shortest path from $s$ to $t$ in the residue network, while loop runs at most $O(|V||E|)$ times, so the total cost is $O(|V||E|^2)$. Called Edmonds-Karp algorithm.
Network flows with multiple sources and sinks

- Reduce to network flow with single source and single sink
- Introduce a supersource $s$ which is directly connected to each of the original sources $s_i$ with a capacity $c(s,s_i) = \infty$
- Introduce a supersink $t$ which is directly connected from each of the original sinks $t_i$ with a capacity $c(s_i,t) = \infty$
Maximum bipartite matching

• Matching in a undirected graph $G=(V,E)$
  – A subset of edges $M \subseteq E$, such that for all vertices $v \in V$, at most one edge of $M$ is incident on $v$.

• Maximum matching $M$
  – For any matching $M'$, $|M| \geq |M'|$.

• Bipartite: $V=L \cup R$ where $L$ and $R$ are distinct and all the edges go between $L$ and $R$.

• Practical application of bipartite matching:
  – Matching a set $L$ of machines with a set $R$ of tasks to be executed simultaneously.
  – The edge means that a machine can execute a task.
Finding a maximum bipartite matching

• Construct a flow network $G'=(V',E',C)$ from $G=(V,E)$ as follows where $V=L \cup R$:
  – $V'=V \cup \{s,t\}$, introducing a source and a sink
  – $E'=((s,u): u \in L) \cup E \cup \{(v,t): v \in R\}$
  – For each edge, its capacity is unit 1.
• As a result, the maximum flow in $G'$ is a maximum matching in $G$. 