ORF 307: Lecture 15

Linear Programming: Chapter 15: Network Flows: Applications

Robert J. Vanderbei

April 11, 2019

Slides last edited on January 25, 2019

Transportation Problem

Each node is one of two types:

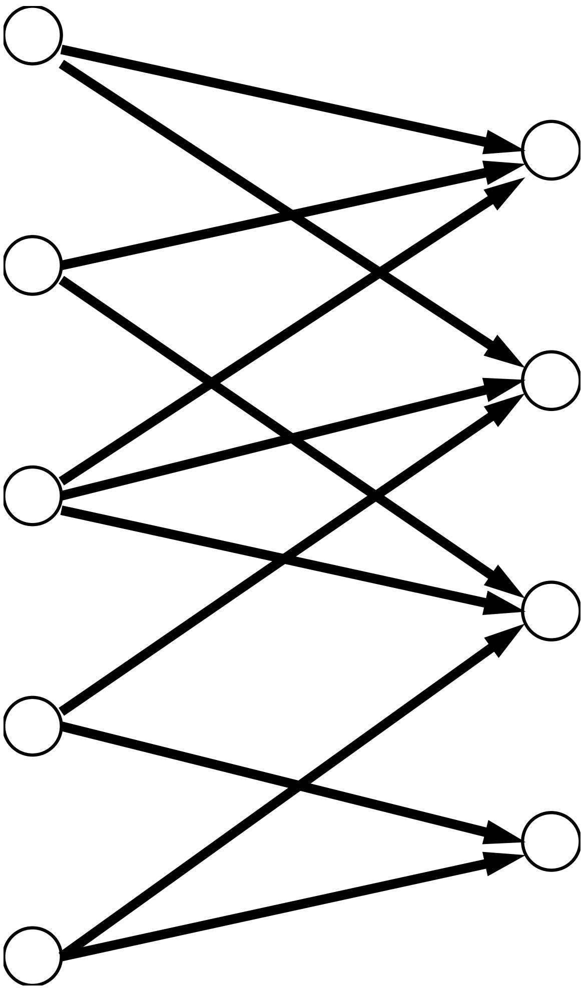
- source (supply) node
- destination (demand) node

Every arc has:

- its tail at a supply node
- its head at a demand node

Such a graph is called *bipartite*.

Notoriously *not planar*.



Assignment Problem (Marriage Problem)

Transportation problem in which

- Equal number of supply and demand nodes.
- Every supply node has a supply of one.
- Every demand node has a demand for one.
- Each supply node is connected to every demand node (called a *complete bipartite graph*).
- Solution is required to be all integers.

Notes:

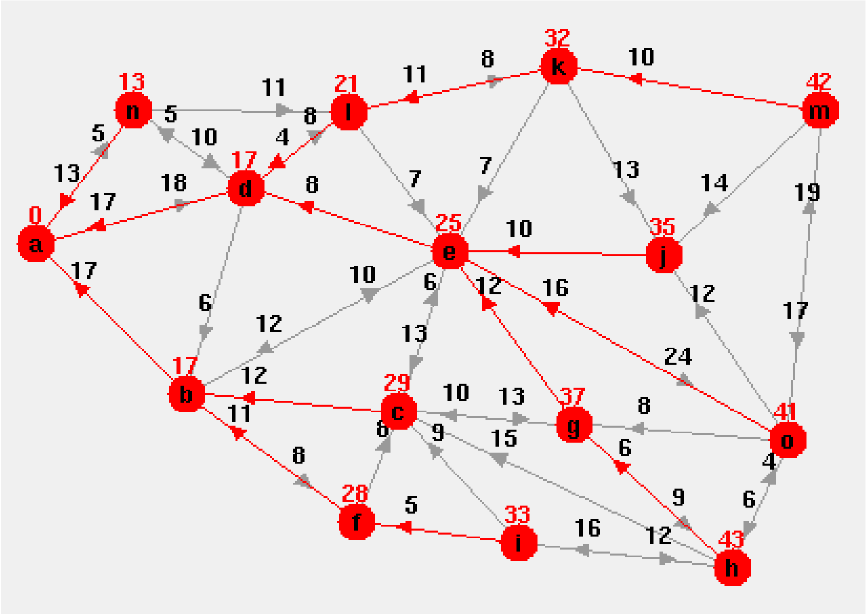
- These problems are very common.
- They are notoriously degenerate ($2n$ constraints but only n nonzero flows).

Shortest Paths Problem

Given:

- Network: $(\mathcal{N}, \mathcal{A})$
- Costs = Travel Times: c_{ij}, $(i, j) \in \mathcal{A}$
- Home (root): $r \in \mathcal{N}$

Problem: Find shortest path from every node in $\mathcal{N}$ to root.



Network Flow Formulations

First Thought...

- Put
$$b_i = \begin{cases} 1 & i = \text{starting point} \\ -1 & i = \text{destination} \end{cases}$$
- Solve min-cost network flow problem.
- Shortest path from source to destination: follow tree arcs.
- Highly degenerate. Most tree arcs have zero flow.

A Better Method

- Put
$$b_i = \begin{cases} 1 & i \neq r \\ -(m-1) & i = r \end{cases}$$
- Shortest path from i to r: follow tree arcs.
- Length (of time) of shortest path $= y_r^* - y_i^*$.

Notation Used in Following Algorithms NOTE: NOT COVERED IN CLASS

- Put v_i = minimum time from i to r
  - Called *label* in networks literature.
  - Called *value* in dynamic programming literature.

Label Correcting Algorithm = Dynamic Prog.

- *Bellman's Equation = Principle of Dynamic Programming*

$$
\begin{aligned}
v_r &= 0 \\
v_i &= \min\{c_{ij} + v_j : (i,j) \in \mathcal{A}\} \\
T &= \{(i,j) \in \mathcal{A} : v_i = c_{ij} + v_j\} \qquad \text{– not necessarily a tree}
\end{aligned}
$$

- *Method of Successive Approximation*
  - Let k denote an iteration counter.
  - Fix root node's value to zero for all iterations: $v_r^{(k)} = 0$ for all k.
  - For all other nodes...
    * Initialize: $v_i^{(0)} = \infty$.
    * Iterate: $v_i^{(k+1)} = \min\{c_{ij} + v_j^{(k)} : (i,j) \in \mathcal{A}\} \quad i \neq r$.
    * Stop: when a pass leaves v_i's unchanged.

- *Complexity*
  - $v_i^{(k)}$ = length of shortest path having k or fewer arcs.
  - Requires at most $m - 1$ passes.
  - n adds/compares per pass.
  - mn operations in total.

Label Setting Algorithm = Dijkstra's Algorithm

Notations:

- F = set of finished nodes (labels are *set*).
- $h_i,\ i \in \mathcal{N}$ = next node to visit after i (heading).

Dijkstra's Algorithm:

- Initialize:

$$F = \emptyset, \qquad v_j = \begin{cases} 0 & j = r \\ \infty & j \neq r \end{cases}$$

- Iterate:
  - While unfinished nodes remain, select the one with smallest v_k. Call it j. Add it to set of finished nodes F.
  - For each unfinished node i having an arc connecting it to j:
    - If $c_{ij} + v_j < v_i$, then set

$$\begin{aligned} v_i &= c_{ij} + v_j \\ h_i &= j \end{aligned}$$

Dijkstra's Algorithm—Complexity

- Each iteration finishes one node: m iterations
- Work per iteration:
  - Selecting an unfinished node:
    - Naively, m comparisons.
    - Using appropriate data structures, a *heap*, $\log m$ comparisons.
  - Update adjacent arcs.
- Overall: $m \log m + n$.