

ORF 522: Lecture 12

Linear Programming: Chapter 14 Network Flows: Applications

Robert J. Vanderbei

October 22, 2013

Slides last edited at 9:34pm on Monday 21st October, 2013

Transportation Problem

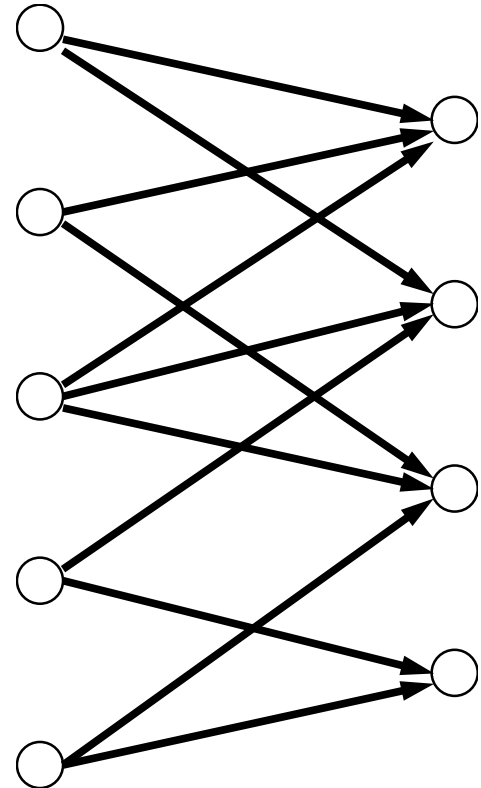
Each node is one of two types:

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

Every arc has:

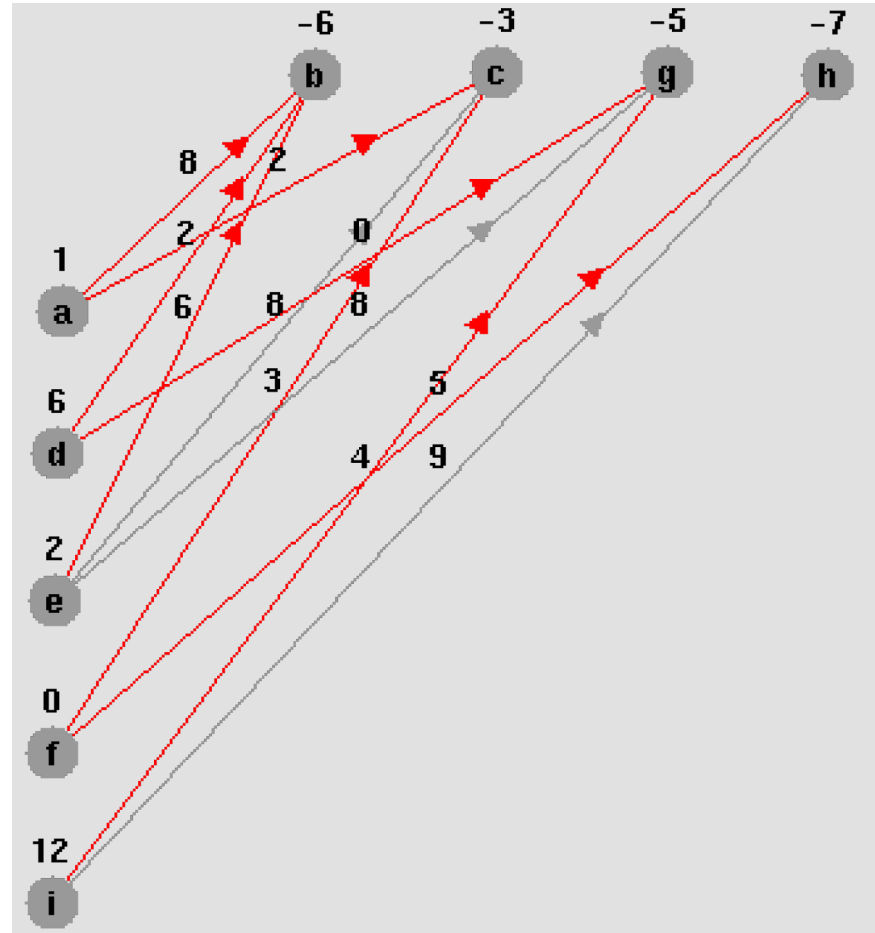
- its tail at a supply node
- its head at a demand node

Such a graph is called *bipartite*.



Solving with Pivot Tool

Data:



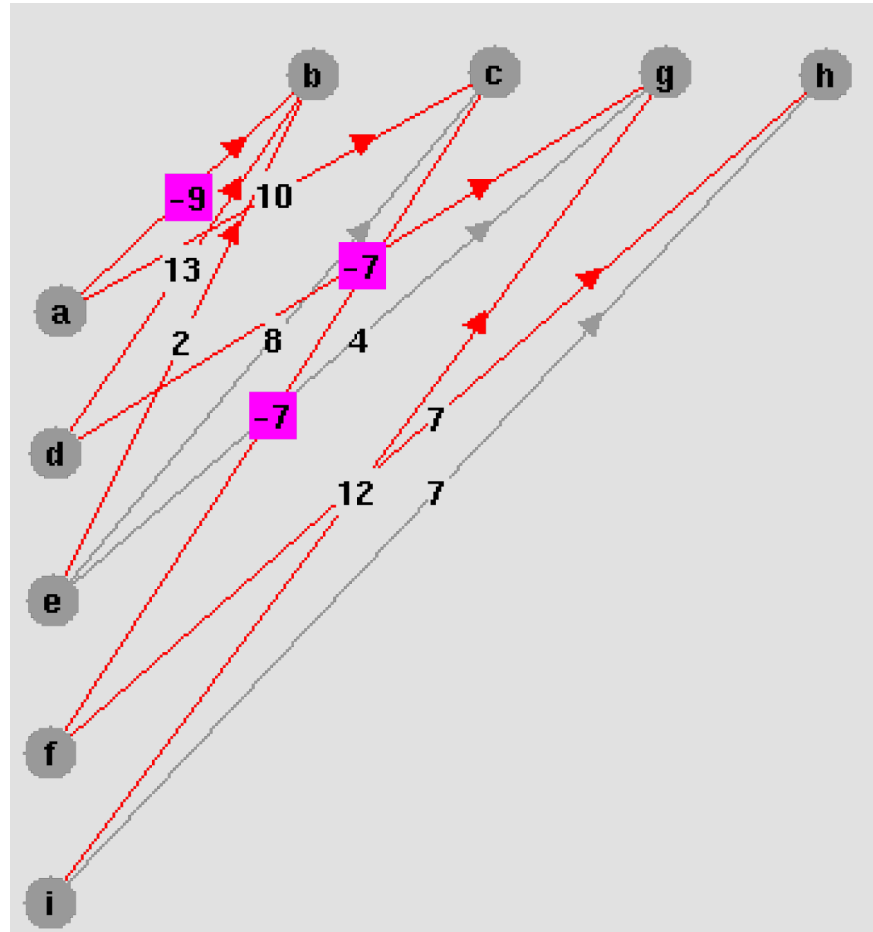
Best to arrange:

- supply nodes vertically on left
- demand nodes horizontally across top

Note that arc data appears as a neat table.

Tree Solution

Leaving arc: (a,b)
Entering arc: (i,h)
Etc., etc., etc.



Assignment Problem

Transportation problem in which

- There are an 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:

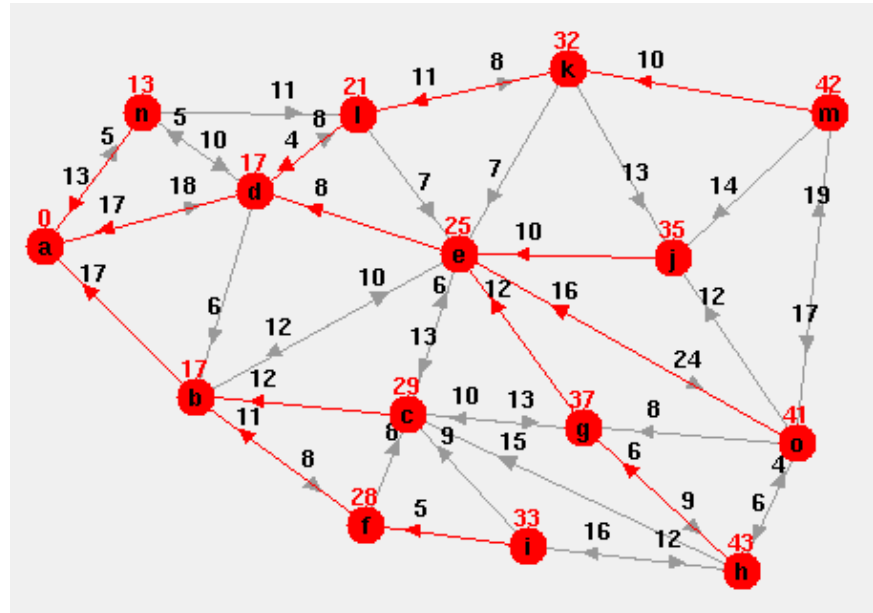
- These problems are very common.
- They are notoriously degenerate ($2n$ constraints but only n nonzero flows).

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 Formulation

- Put

$$b_i = \begin{cases} 1 & i \neq r \\ -(m-1) & i = r \end{cases}$$

- Solve min-cost network flow problem.
- Shortest path from i to r : follow tree arcs.
- Length (of time) of shortest path = $y_r^* - y_i^*$.

Notation Used in Following Algorithms

- Put $v_i = \text{min. time from } i \text{ to } r$
 - Called *label* in networks literature.
 - Called *value* in dynamic programming literature.

Label Correcting Algorithm

Dynamic Programming

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

$$v_r = 0 \tag{1}$$

$$v_i = \min\{c_{ij} + v_j : (i, j) \in \mathcal{A}\} \tag{2}$$

$$T = \{(i, j) \in \mathcal{A} : v_i = c_{ij} + v_j\} \quad \text{– not necessarily a tree} \tag{3}$$

- *Method of Successive Approximation*

– Initialize: $v_i^{(0)} = \begin{cases} 0 & i = r \\ \infty & i \neq r \end{cases}$

– Iterate: $v_i^{(k+1)} = \begin{cases} 0 & i = r \\ \min\{c_{ij} + v_j^{(k)} : (i, j) \in \mathcal{A}\} & i \neq r \end{cases}$

– Stop: when a pass leaves v_i 's unchanged.

Label Correcting Algorithm—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:

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

- 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

$$v_i = c_{ij} + v_j \tag{4}$$

$$h_i = j \tag{5}$$

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: $m \log m + n$.

