INSE6290 Jia Yuan Yu

7: Network flow

Concordia November 19, 2015

Network flow problems come in multiple flavors, but they all share a gaphical model. A graph is a collection of

- Vertices V,
- Directed edges $E \subseteq V \times V$.

On each edge (u, v), we assign a weight $c(u, v)$, which models its capacity.

We have already seen graphs and network flow problems in this course: Transportation Problem, Uncapacitated Facility Location Problem. This lecture focuses on another network flow problem: the maximum flow (max-flow) problem.

1 Maximum flow problem

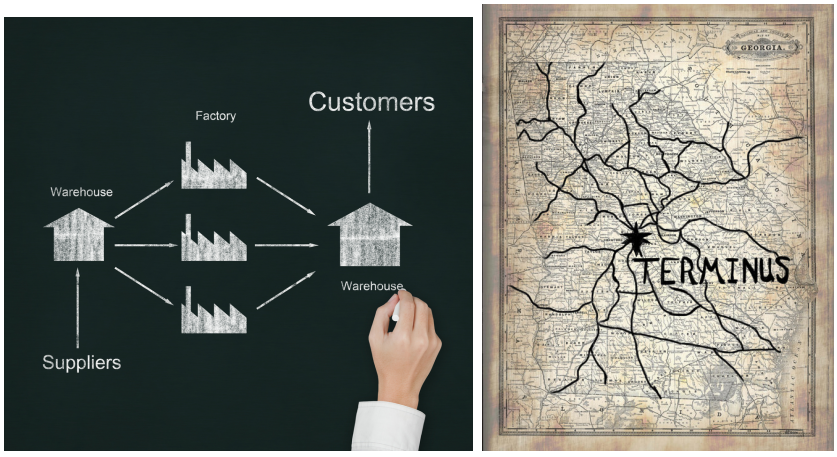

Figure 1: From `http://www.theenergycollective.com/` and The Walking Dead.

How many goods can we produce at vertex s, the source, and send from s to another vertex t, the terminus? In other words, assuming infinite demand at the t vertex, how can we produce goods at vertex s, and ship them through the network without incurring any holding cost for those goods?

Answering this question also solves the following problems:

- Delivery of goods in a truck delivery network (e.g., UPS or FedEx trucks on different legs have available capacities);
- Ticket sale for airlines (vertices are airports, edges are flights);

- Water distribution in a pipe network;
- Electricity distribution.

The decision maker's output is a flow $f : V \times V \to \mathbb{R}$ (one number for each edge) with the following characteristics:

- Capacity: $f(u, v) \in [0, c(u, v)]$;
- Conservation: for each $u \in V$, expect s, t:

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v);$$

- Anti-symmetric shadow edges: for every $(u, v) \in E$, we define $f(v, u) = -f(u, v)$.

Each path in the flow network is a collection of "connected" edges, which represents one possible supply chain.

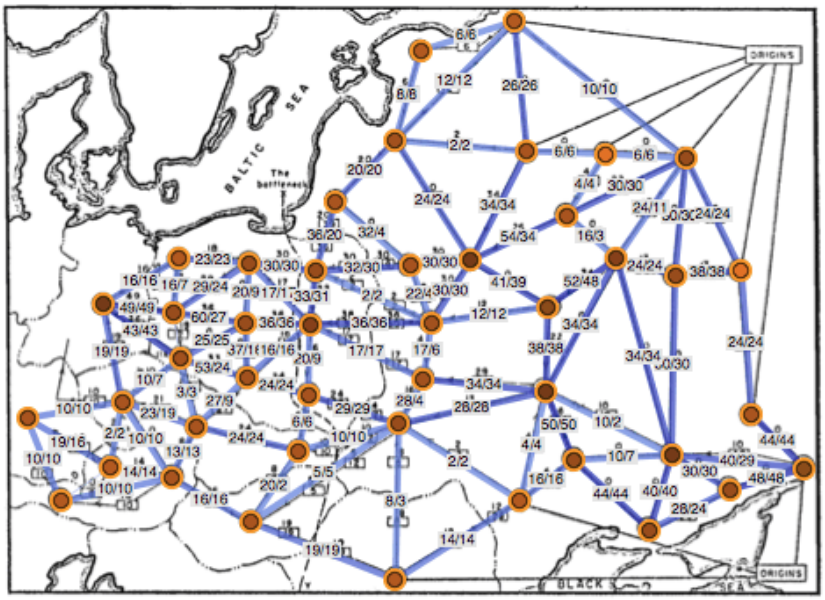

Figure 2: Max flow (min cut) problem in the context of the Soviet rail network (cf. classical paper `http://www.dtic.mil/dtic/tr/fulltext/u2/093458.pdf`). The origin s is to the east of the map, the destination t is to the west.

2 Linear programming

We can solve this problem by linear programming:

$$\begin{aligned}
\max_{\{x_{uv}:(u,v)\in E\}} \quad & \sum_{v:(s,v)\in E} x_{sv} \\
\text{subject to} \quad & \sum_{u:(u,v)\in E} x_{uv} = \sum_{w:(v,w)\in E} x_{vw}, \quad \text{for } v \in V \setminus \{s,t\}, \\
& x_{uv} \le c(u,v), \quad \text{for } (u,v) \in E, \\
& x_{uv} \ge 0, \quad \text{for } (u,v) \in E.
\end{aligned}$$

We simply replaced each output $f(u,v)$ by the decision variable x_{uv}, and expressed the objective and constraints with linear functions.

There are many ways to solve linear programs (cf. Lecture 5).

3 Ford-Fulkerson

Linear programming solutions for the max-flow problem may take too long, in which case, we can use the Ford-Fulkerson algorithm.

Example 3.1 (Wikipedia)**.** The idea behind the algorithm is as follows: as long as there is a path from the source (start node) to the sink (end node), with available capacity on all edges in the path, we send flow along one of the paths. Then we find another path, and so on. A path with available capacity is called an augmenting path.

We can solve the max-flow problem as follows:

1. Input: $G_c = (V, E)$, special vertices s, t, and capacity function c.
2. Start with a flow $f = 0$.
3. While there is a path p from s to t:
   (a) (Find an augmenting path in G_c) Find a path p from s to t in G_c.
   (b) Find $v(p) = \min\{c_f(u,v) : (u,v) \in p\}$.
   (c) Update $f(u,v) \leftarrow f(u,v) + v(p)$ for all $(u,v) \in p$.
   (d) (Update the residual network G_c) For each edge $(u,v) \in p$:
      i. $c(u,v) \leftarrow c(u,v) - v(p)$ (Increased flow results in decreased capacity).
      ii. $c(v,u) \leftarrow c(v,u) + v(p)$ (Shadow edge capacity).
4. Output f.

We illustrate this algorithm on the Example of Figure 3.
Let's consider another example in Figure 3.

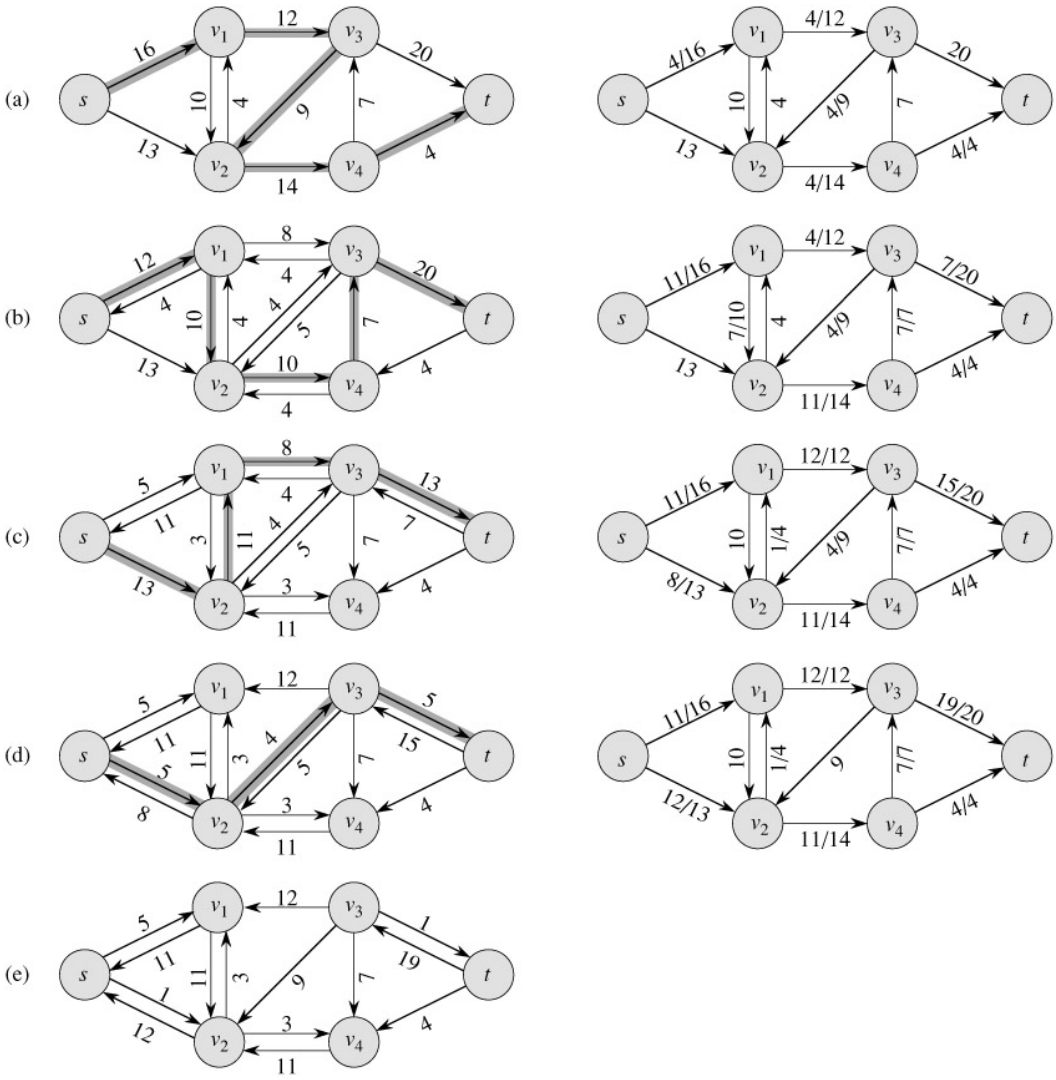

Figure 3: Max flow example from Introduction to Algorithms.

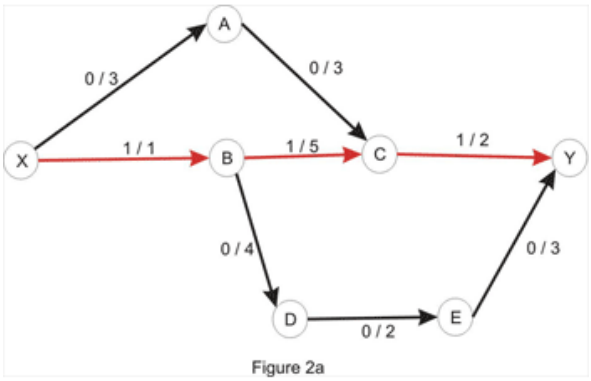

Figure 4: Max flow example from `https://www.topcoder.com/community/data-science/data-science-tutorials/maximum-flow-section-1/`.

4 Reading material

- Ford-Fulkerson: Introduction to Algorithms by Cormen, Leiserson, Rivest, and Stein.
- More applications of network flows: A full course on network flows.