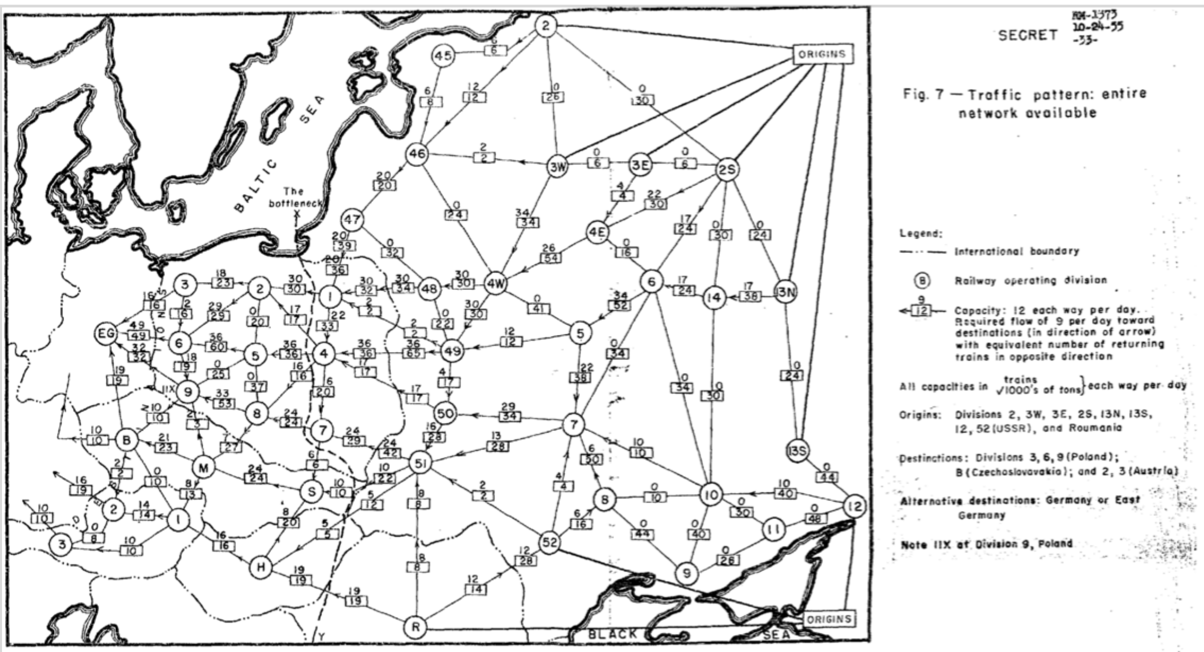
CS 4349.003.19F Lecture 20—November 4, 2019

Main topics for #lecture include #maximum_flow and #minimum_cut.

Shipment Rates and Bottlenecks

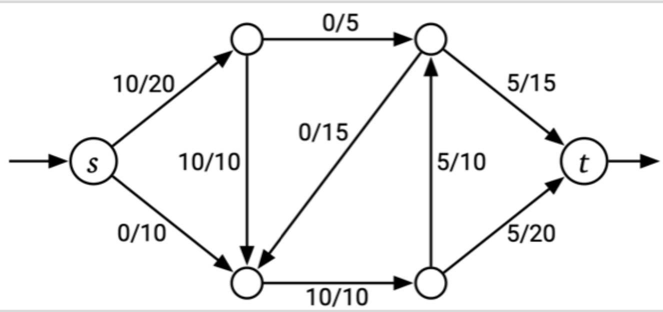
- Now, let's get started on one last subject in graph algorithms, and the one I think is the most interesting.
- "In the mid-1950s, U. S. Air Force researcher Theodore E. Harris and retired U. S. Army general Frank S. Ross wrote a classified report studying the rail network that linked the Soviet Union to its satellite countries in Eastern Europe. The network was modeled as a graph with vertices, representing geographic regions, and edges, representing links between those regions in the rail network. Each edge was given a weight, representing the rate at which material could be shipped from one region to the next. Essentially by trial and error, they determined both the maximum amount of stuff that could be moved from Russia into Europe, as well as the cheapest way to disrupt the network by removing links (or in less abstract terms, blowing up train tracks), which they called "the bottleneck". Their report, which included the drawing of the network [below], was only declassified in 1999. " – Erickson



- We're going to talk about how *not* to do these two things by trial and error.
- Specifically, we're going to discuss two problems known as the *maximum flow* problem, and the *minimum cut* problem.
- For both problems, we're given a directed graph G = (V, E) with special vertices s, the *source*, and t, the *target* or *sink*.
- The maximum flow measures how much material can be transported from s to t.
- The minimum cut measures how much damage we need to do to separate s from t.

Maximum Flow

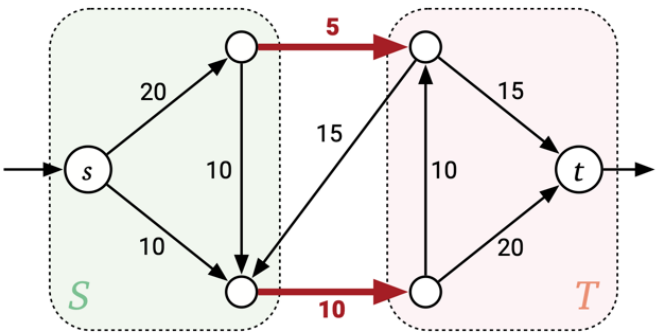
- An *(s, t)-flow* is a way of assigning values to the edges that models how material flows through a network. You could also imagine the network as a series of tubes or pipes. We're measuring how water, or trains, moves through them.
- Formally, its a function f : E → R≥0 that satisfies the *conservation constraint* at every vertex v expect maybe s and t:
    - sum_u f(u → v) = sum_w f(v → w)
    - In other words, flow into v must equal flow out.
    - Here I'm using the convention that f(u → v) = 0 if there is no edge u → v.
- Let partial f(v) := sum_w f(v → w) - sum_u f(u → v) denote the net flow out of v. The conservation constraints say partial f(v) = 0 for all v except s and t.
- |f| is the *value* of the flow f. It is the net flow *out of* vertex s.
    - |f| := partial f(s) = sum_w f(s → w) - sum_u f(u → s)
- It turns out the value of f is also equal to the net flow *into* t:
    - sum_v partial f(v) = partial f(s) + partial f(t)
    - But every edge leaves one vertex and enters another, meaning the sum of the net flows out of vertices must equal 0.
    - So sum_v partial f(v) = 0, implying partial(t) = -partial(s) = |f|.
- OK, so the name of the problem implies we want to maximize the flow from s to t. So we need some limit on how much flow we'll send through an edge.
- We'll use a *capacity* function c : E → R≥0 where c(e) is a non-negative capacity for an edge. Think of it as the width of the pipe or throughput of the railline.
- Flow f is *feasible* with respect to c if f(e) ≤ c(e) for every edge e.
- In particular, f *saturates* edge e if f(e) = c(e) and *avoids* e if f(e) = 0.
- Here's an example of a feasible (s, t)-flow of value 10.



- The *maximum flow problem* is to compute a maximum value (s, t)-flow that is feasible with respect to c.
- We'll eventually get to algorithms for this problem, but first let's talk about minimum cuts.

Minimum Cut

- An *(s, t)-cut* is a partition of the vertices into disjoint subsets S and T, meaning S U T = V and S intersect T = empty, where s in S and t in T.
- Again, we'll work with a capacity function c : E → R≥0. The *capacity* of a cut (S, T) is the sum of capacities for edges that start in S and end in T.
    - ||S, T|| := sum_{v in S} sum_{w in T} c(v → w)
    - Similar to before, we're assuming c(v → w) = 0 if v → w is not in the graph.
- This definition is asymmetric. Edges that start in T and end in S don't matter at all when defining the capacity of the cut.
- Here's an example of an (s, t)-cut of capacity 15. Yes, 15. That backwards edge does not count.



- The *minimum cut problem* is to compute an (s, t)-cut with minimum capacity.
- One way to think about the problem is that the minimum (s, t)-cut is the cheapest way to disrupt all flow from s to t. And we can make that relationship formal.
- Lemma: The value of *any* feasible (s, t)-flow f is at most the capacity of *any* (s, t)-cut (S, T).

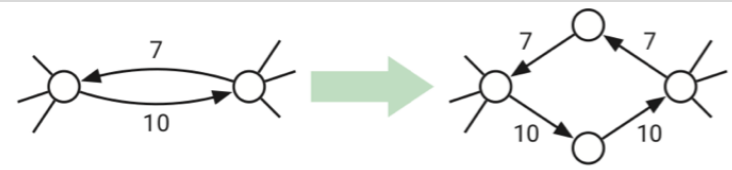
$$
\begin{aligned}
|f| &= \partial f(s) && \text{[by definition]} \\
&= \sum_{v\in S} \partial f(v) && \text{[conservation constraint]} \\
&= \sum_{v\in S}\sum_{w} f(v\to w) - \sum_{v\in S}\sum_{u} f(u\to v) && \text{[math, definition of } \partial\text{]} \\
&= \sum_{v\in S}\sum_{w\notin S} f(v\to w) - \sum_{v\in S}\sum_{u\notin S} f(u\to v) && \text{[removing edges from } S \text{ to } S\text{]} \\
&= \sum_{v\in S}\sum_{w\in T} f(v\to w) - \sum_{v\in S}\sum_{u\in T} f(u\to v) && \text{[definition of cut]} \\
&\leq \sum_{v\in S}\sum_{w\in T} f(v\to w) && \text{[because } f(u\to v)\geq 0\text{]} \\
&\leq \sum_{v\in S}\sum_{w\in T} c(v\to w) && \text{[because } f(v\to w)\leq c(v\to w)\text{]} \\
&= \|S,T\| && \text{[by definition]}
\end{aligned}
$$

- Now, look at the two inequality lines. The first is an equality if and only if there is no flow going from T to S. The second is an equality if and only if the flow saturates every edge from S to T.

- In other words: |f| = ||S, T|| if and only if f saturates every edge from S to T and avoids every edge from T to S. In this case, we can't make |f| any bigger, so f must be a maximum flow. Also, we can't make ||S, T|| any smaller, so and (S, T) must be a minimum cut.

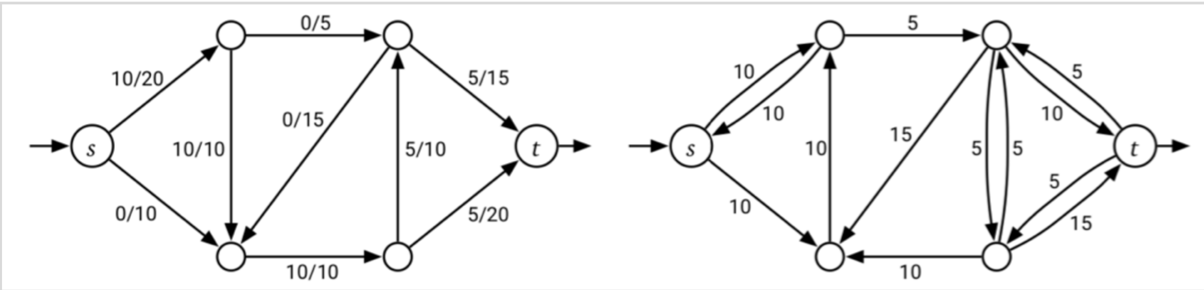
The Maxflow Mincut Theorem

- The surprising thing, and the thing most algorithms for this problem rely upon, is that the value of the maximum flow is always *equal* to the capacity of the minimum cut.
- This was shown by Ford and Fulkerson in 1954 and independently by Elias, Feinstein, and Shannon in 1956.
- The Maxflow Mincut Theorem: In any flow network with source s and target t, the value of a maximum (s, t)-flow is equal to the capacity of a minimum (s, t)-cut.
- To make the proof and subsequent algorithms easier, we'll assume the capacity function is *reduced*. For every pair of vertices u and v, at most one of edge u ➔ v or edge v ➔ u is in E. Or if you prefer, c(u ➔ v) = 0 or c(v ➔ u) = 0.
    - We can enforce this assumption by modifying the graph a bit. If both u ➔ v and v ➔ u appear in the graph, we'll add two vertices x and y, replace u ➔ v with a path u ➔ x ➔ v, replace v ➔ u with v ➔ y ➔ u, set c(u ➔ x) ⬅ c(x ➔ v) ⬅ c(u ➔ v), and set c(v ➔ y) ⬅ c(y ➔ u) ⬅ c(v ➔ u).

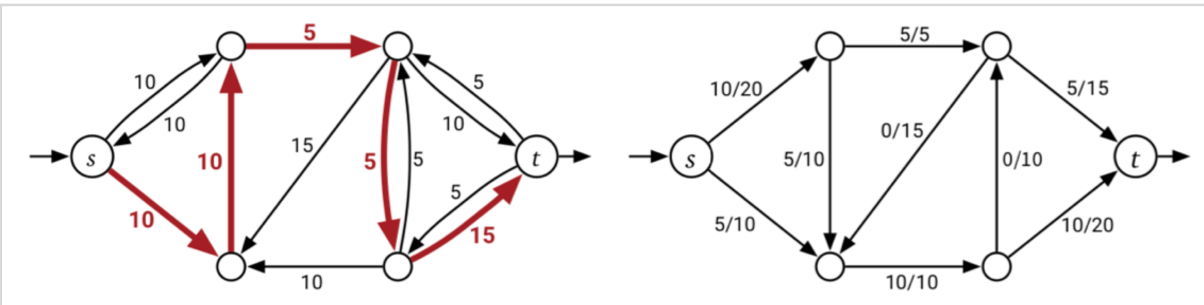


- Now suppose we have a flow f. If we can modify f to increase its value, then it must not be a maximum flow. On the other hand, I'll show you a minimum cut of equal capacity if we can't increase f's value.
- Now, how should we update f to increase its value? You can imagine pushing some material through the network along a single path like sending a single train from s to t.
- Unfortunately, there may not be a path from s to t along which we can send more flow. We may need to reduce the flow on some edges to increase f's value.
- The main idea will be to encode how much more flow we can add to some edges and how much flow we can *undo* from others by defining a different graph.
- The *residual capacity* function c_f : V x V ➔ R is based on flow f.
- c_f(u ➔ v) =
    - c(u ➔ v) - f(u ➔ v) if u ➔ v in E
    - f(v ➔ u) if v ➔ u in E
    - 0 otherwise

- Remember, we're assuming no pair of edges $u \to v$ and $v \to u$ have positive capacity, so only one of those cases holds.
- Since $f(u \to v) \geq 0$ and $f(u \to v) \leq c(u \to v)$, the residual capacities are non-negative.
- But, we may have $c_f(u \to v) > 0$ even if $u \to v$ is not an edge in the graph G.
- So we define a new graph called the *residual graph* $G_f = (V, E_f)$ where E_f is the all the edges with positive residual capacity.
- Let's look at an example. The original graph with some flow f is on the left. The residual graph G_f is on the right.



- You might notice that the residual graph is not necessarily reduced. We have two edges on the left with positive capacity 10.
- Now, suppose we have flow f and we've computed the residual graph G_f. There is either a path from s to t in G_f or there isn't.
- Suppose there is a path P from s to t in G_f.
  - We call P an *augmenting path*. We'll see why in a second.
  - Let $F = \min_{u \to v \text{ in } P} c_f(u \to v)$ be the maximum amount of flow we can "push" through the augmenting path in G_f.
  - By push, I mean we define a new flow $f' : E \to R$ where $f'(u \to v) =$
    - $f(u \to v) + F$ if $u \to v$ in P
    - $f(u \to v) - F$ if $v \to u$ in P
    - $f(u \to v)$ otherwise
  - Again, graph G's edges are reduced, so exactly one case holds.
  - Here, we push 5 units of flow along an augmenting path.



  - We don't change the net flow out of any vertex except s and t, so f' is still an (s, t)-flow.
  - But is it feasible? Consider any edge $u \to v$ in E.
    - If $u \to v$ in P,
      - $f'(u \to v) = f(u \to v) + F > f(u \to v) \geq 0$
      - Also $f'(u \to v) = f(u \to v) + F$ by definition of f'

            - ≤ f(u ➜ v) + c_f(u ➜ v) by definition of F
            - = f(u ➜ v) + c(u ➜ v) - f(u ➜ v) by definition of c_f
            - = c(u ➜ v)
    - If v ➜ u in P,
        - f'(u ➜ v) = f(u ➜ v) - F < f(u ➜ v) ≤ c(u ➜ v).
        - Also, f'(u ➜ v) = f(u ➜ v) - F by definition of f'
            - ≥ f(u ➜ v) - c_f(v ➜ u) by definition of F
            - = f(u ➜ v) - f(u ➜ v) by definition of c_f
            - = 0
  - So f' is a feasible (s, t)-flow.
  - Finally, only the first edge of the augmenting path leaves s, so |f'| = |f| + F > |f|. We made some progress! I guess f wasn't a maximum s,t-flow.
- Now, suppose there is no path from source s to target t in the residual graph G_f.
  - Let S be the vertices reachable from s in G_f, and let T = V \ S.
  - Partition (S, T) is an (s, t)-cut, and for every u in S and v in T:
    - If u ➜ v in E, then 0 = c_f(u ➜ v) = c(u ➜ v) - f(u ➜ v)
    - If v ➜ u in E, then 0 = c_f(u ➜ v) = f(v ➜ u)
  - In other words, f saturates every edge from S to T and avoids every edge from T to S.
  - So, f is a maximum flow and (S, T) is a minimum cut!
- In short, either there is an augmenting path from s to t in the residual graph and we can strictly increase the value of f by pushing along that path, meaning f was not a maximum flow to begin with.
- … or there is no path from s to t in the residual graph and f is a maximum flow with value equal to the capacity of the minimum cut.
- On Wednesday, we'll turn this proof into an actual algorithm for computing maximum flows and minimum cuts.