CS 4349 Lecture—November 1, 2017

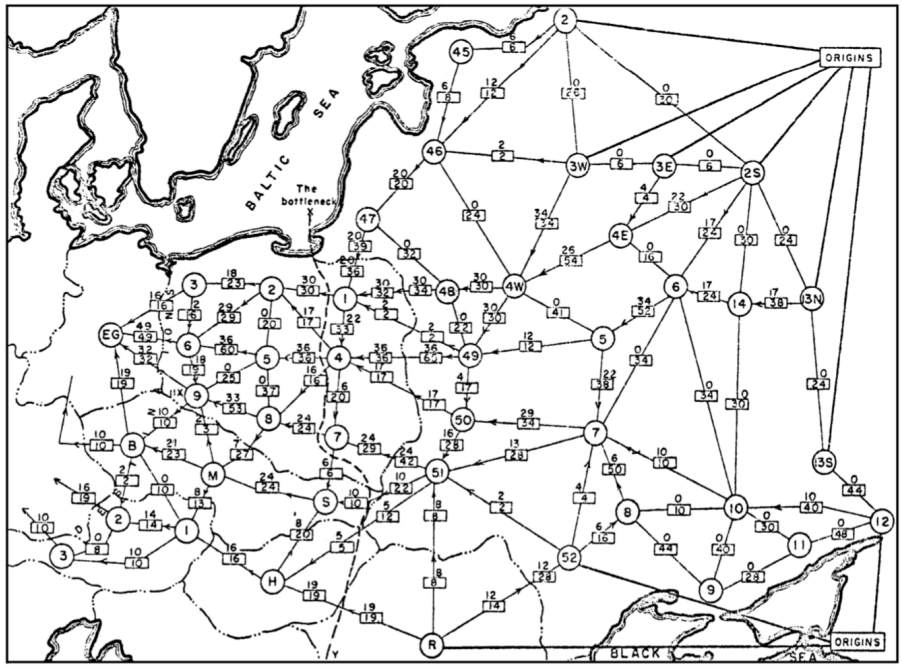
Main topics for #lecture include #maximum_flow and #minimum_cut.

Prelude

- Homework 8 due today.
- Homework 9 due Wednesday, November 8th.

Shipment Rates and Bottlenecks

- "In the mid-1950s, Air Force researcher Theodore E. Harris and retired army general Frank S. Ross published 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 44 vertices, representing geographic regions, and 105 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 results, including the drawing of the network [above], were only declassified in 1999." – Erickson

Today, we're going to talk about how *not* to do these two things by trial and error.

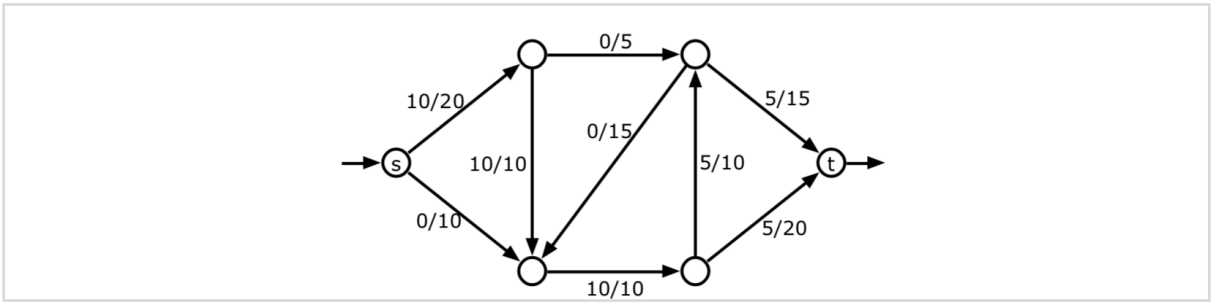
- Specifically, we're going to discuss two problems known as the *maximum flow* problem, and the *minimum cut* problem.
- For both problems, we're given a directed graph G = (V, E) with special vertices s, the

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

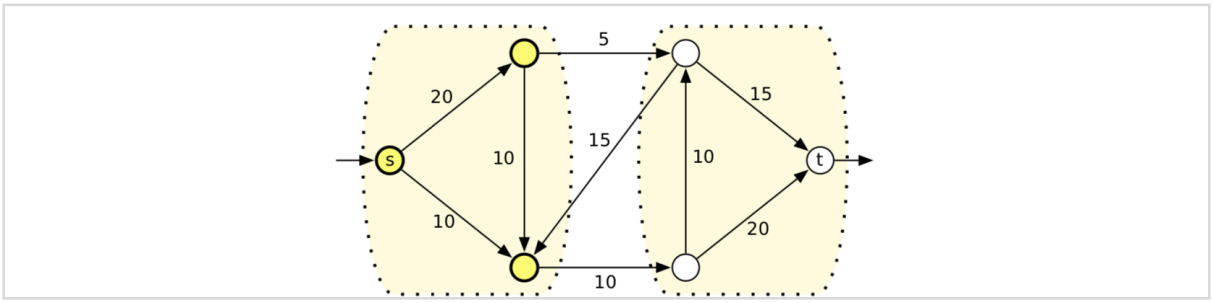
- An *(s, t)-flow* is a way of assigning values to the edges that models how material flows through a network. You could also imagine the network as a series of tubes. We're just measuring how fast water moves through them.
- Formally, its a function f : E ➔ R≥0 that satisfies the *conservation constraint* 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's I'm using the convention that f(u ➔ v) = 0 if there is no edge u ➔ v.
- |f| is the *value* of the flow f. It is the net flow *out of* vertex s.
  - |f| := 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.
  - Define partial f(v) := sum_w f(v ➔ w) - sum_u f(u ➔ v), the net flow out of vertex v.
  - partial f(v) = 0 for all vertices v not equal to s or t, so
    - 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 |f| = partial f(s) = -partial f(t)
- 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.
- Flow f is *feasible* with respect to c if f(e) ≤ c(e) for every edge e.
- 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)
  - Again, I'm being lazy here and just 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.



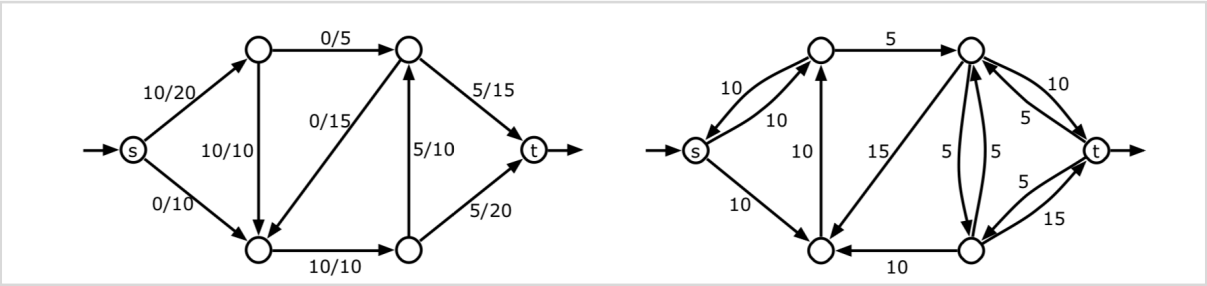
- The *minimum cut problem* is to compute an (s, t)-cut with minimum capacity.
- One way to think about the problem is that the minimum (s, t)-cut is the cheapest way to disrupt all flow from s to t. And we can make that relationship formal.
- Claim: The value of *any* feasible (s, t)-flow is at most the capacity of *any* (s, t)-cut.
  - Choose any flow f and cut (S, T). Now settle in for some exciting algebra.
  - |f| = sum_w f(s ➔ w) - sum_u f(u ➔ s) by definition
    - = sum_{v in S} (sum_w (v ➔ w) - sum_u f(u ➔ v)) by the conservation constraints, because t is not in S
    - = sum_{v in S} (sum_{w in T} f(v ➔ w) - sum_{u in T} f(u ➔ v)) because I was adding *and* subtracting the flow to and from every vertex w in S
    - ≤ sum_{v in S} sum_{w in T} f(v ➔ w) since f(u ➔ v) ≥ 0 for all u in T
    - ≤ sum_{v in S} sum_{w in T} c(v ➔ w) since f(v ➔ w) ≤ c(v ➔ w)
    - = ||S, T|| by definition
- We can look at the two inequality lines to learn something else. The only way those inequalities can be strict is if we have flow going on an edge u ➔ v with u in T and s in V or if f(v ➔ w) < c(v ➔ w) for some v in S and w in 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.
- And: if we have a flow f and cut (S, T) that satisfies this equality condition, f must be a maximum flow and (S, T) must be a minimum cut.

The Maxflow Mincut Theorem

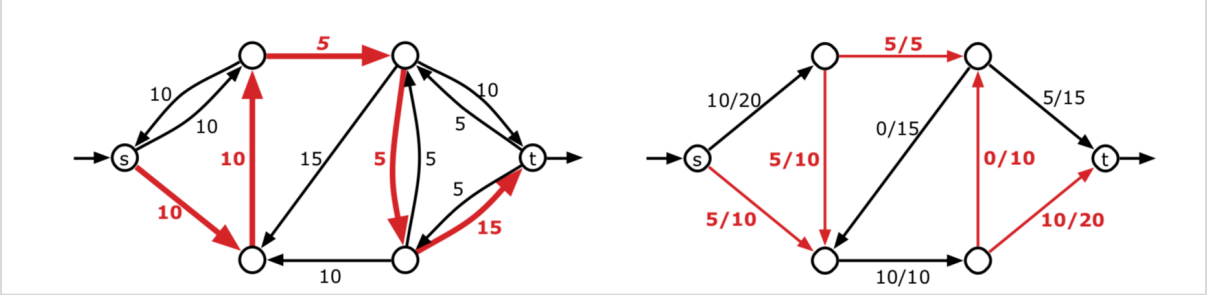
- I just showed you that the value of any flow, including the maximum flow, is at most the capacity of any cut, including the minimum cut.
- So the value of the maximum flow is at most the capacity of the minimum cut.
- But the surprising thing, and the thing that makes algorithms for this problem possible, 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 (Ford-Fulkerson): In any flow network with source s and target t, the value of the maximum (s, t)-flow is equal to the capacity of the minimum (s, t)-cut.
- We'll spend the rest of today proving the maxflow mincut theorem. It turns out the proof practically gives us an algorithm for both problems.
- To make life easier, we'll assume the capacity function is *reduced*. For every pair of vertices u and v, either c(u → v) = 0 or c(v → u) = 0. Of if you prefer, the graph contains at most one of those two edges.
  - 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 a vertex x, replace u → v with a path u → x → v, and set c(u → x) = c(x → v) = c(u → v).

- Alright, so here's one idea that won't work. A flow models the movement of material through the network, right? So let's try to push as much material through the network as possible until we've blocked our progress.
- The problem here is that we may make a silly choice at the beginning. We need some way to correct for these decisions!
- The main idea for the proof will be the idea of residual capacities and graphs that basically describe how to correct for our mistakes.
- Let f be any feasible flow. The *residual capacity* function c_f : V x V → R is based on the current flow f.
- c_f(u → v) =
  - c(u → v) - f(u → v) if u → v in E (or c(u → v) > 0)
  - f(v → u) if v → u in E (or c(v → u) > 0)
  - 0 otherwise
- Remember, we're assuming no pair of edges u → v and v → u have positive capacity, so only one of those cases holds.
- Since f(u → v) ≥ 0 and f(u → v) ≤ c(u → v), the residual capacities are non-negative.

- But, we may have c(u ➡ v) > 0 even if u ➡ 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.
- So 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.
- So now we're ready to proceed with the proof of the maxflow mincut theorem.
- 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, we have
    - 0 = c_f(u ➡ v) = c(u ➡ v) - f(u ➡ v) if u ➡ v in E and
    - 0 = c_f(u ➡ v) = f(v ➡ u) if v ➡ u in E
  - In other words, f saturates every edge from S to T and avoids every edge from T to S.
  - So from what we saw earlier |f| = ||S, T||. The value of the flow can't get higher and the capacity of the cut can't get lower, so f is a maximum flow and (S, T) is a minimum cut and their values are equal.
- But now suppose there is a path s = v_0 ➡ v_1 ➡ … ➡ v_r = t in G_f.
  - We call this path an *augmenting path*. We'll see why in a second.
  - Let F = min_i c_f(v_i ➡ v_{i+1}) 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 ➡ R where f'(u ➡ v) =
    - f(u ➡ v) + F if u ➡ v is in the augmenting path
    - f(u ➡ v) - F if v ➡ u is in the augmenting path
    - f(u ➡ v) otherwise
  - Again, graph G's edges are reduced, so exactly one case holds.
  - Here's what pushing 5 units of flow along an augmenting path looks like.



- So, does f′ satisfy the conservation constraints?
  - For any v_i with 0 < i < r, if v_{i-1} ➔ v_i and v_i ➔ v_{i+1} are both in E, then we just increased both by F for 0 net change to v.
  - If v_{i-1} ➔ v_i and v_{i+1} ➔ v_i are both in E, then we just increased the first by F and decreased the second, so again 0 net change. The last case is symmetric.
- So f′ is still an (s, t)-flow. But is it feasible? Let u ➔ v be any edge in G.
  - If u ➔ v is on the augmenting path, then f′(u ➔ v) > f(u ➔ v) ≥ 0. Also f′(u ➔ v)
    - = f(u ➔ 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 is on the augmenting path then f′(u ➔ v) < 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. But F > 0 so f is *not* a maximum flow.
- In short. Either there is not 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, or…
- there is an augmenting path from s to t in the residual graph. We can strictly increase the value of f by pushing along that path, meaning f was not a maximum flow to begin with.
- These two observations are going to be the key to designing algorithms for the maximum flow and minimum cut problems, but we'll have to hold that thought until next week.