CS 4349 Lecture—November 6, 2017

Main topics for #lecture include #maximum_flow and #minimum_cut, #Ford-Fulkerson, #Edmonds-Karp, and #Dinits.

Prelude

- Homework 9 due Wednesday, November 8th.

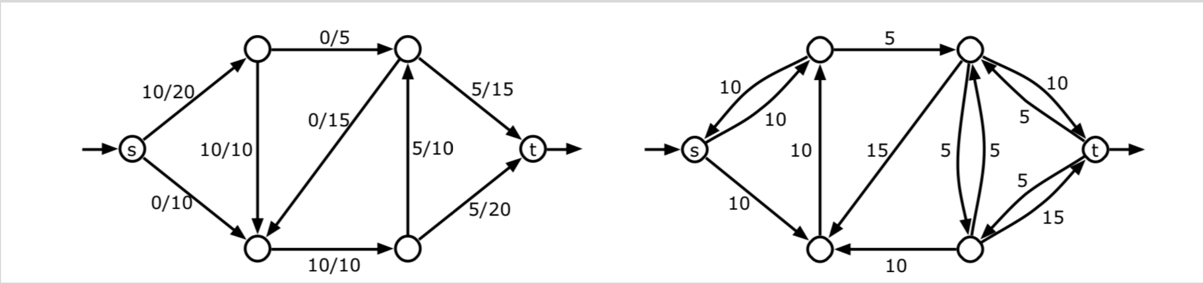
Maxflow-Mincut Review

- An (s, t)-flow is 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)
- |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)
- 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.
- The *maximum flow problem* is to compute a maximum value (s, t)-flow that is feasible with respect to c.

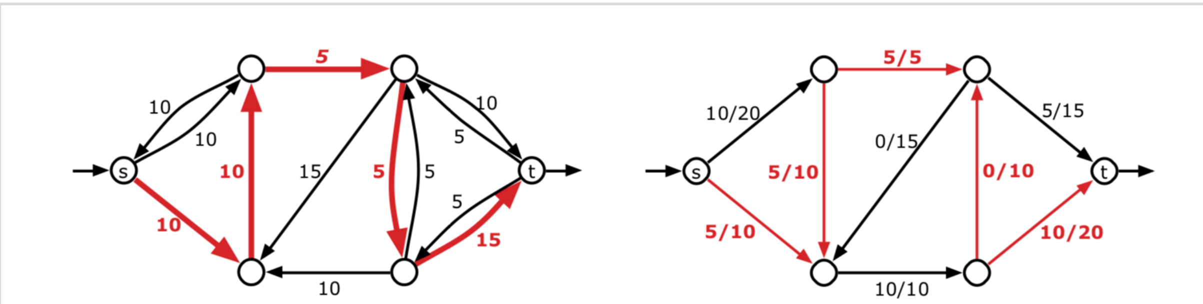
- An *(s, t)-cut* is a partition of the vertices into disjoint subsets S and T, meaning S U T = V and S intersect T = empty, where s in S and t in T.
- The *capacity* of a cut (S, T) is ||S, T|| := sum_{v in S} sum_{w in T} c(v ➔ w)
- The *minimum cut problem* is to compute an (s, t)-cut with minimum capacity.

- 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.
- 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.
- 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

- The *residual graph* $G_f = (V, E_f)$ contains exactly 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.



- Again, the residual graph may not be reduced.
- Suppose there is no path from source s to target t in the residual graph G_f.
- Then let S be the vertices reachable from s in G_f, and let $T = V \setminus S$. We saw last week that (S, T) is a minimum (s, t)-cut where $|f| = \|S, T\|$. So they are a maximum flow and a minimum cut.
- But now suppose there is an *augmenting path* $s = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_r = t$ in G_f.
- Let $F = \min_i c_f(v_i \rightarrow v_{i+1})$ 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 $f' : E \rightarrow R$ where $f'(u \rightarrow v) =$
  - $f(u \rightarrow v) + F$ if $u \rightarrow v$ is in the augmenting path
  - $f(u \rightarrow v) - F$ if $v \rightarrow u$ is in the augmenting path
  - $f(u \rightarrow v)$ otherwise

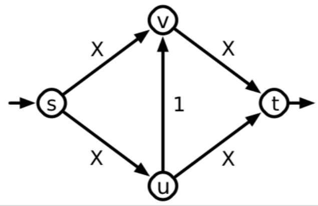


- We saw f' is a feasible (s, t)-flow and $|f'| = |f| + F$.
- In short. Either there is not path from s to t in the residual graph and f is a maximum flow with value equal to the capacity of the minimum cut, or…
- there is an augmenting path from s to t in the residual graph. We can strictly increase the value of f by pushing along that path, meaning f was not a maximum flow to begin with.
- OK, so how to we actually **compute** a maximum (s, t)-flow and minimum (s, t)-cut?

Ford-Fulkerson

- Start with every edge holding 0 flow. Repeatedly augment the flow along *any* (s, t)-path in the residual graph until there is no such path.
- Let's assume capacities are integers for now.

- If that's the case, then every edge will carry an integer amount of flow at the end of every iteration.
  - True before we do anything, since 0 is an integer.
  - If all flow values and capacities are (inductively) integers, then residual capacities are integers.
  - Which means we add an integer amount of flow along the next augmenting path.
- So: if all capacities are integers, there is a maximum flow where the flow through every edge is an integer.
- Each augmenting path pushes at least one unit of flow.
- So if the maximum flow is f^* , we can only do |f^*| iterations.
- In each iteration, we can build the residual graph and do a whatever-first-search to find a residual path in O(E) time. The total running time *assuming integer capacities* is O(E |f^*|).
- When |f^*| is small or even in many practical settings where it isn't, this algorithm is plenty fast, but that's not always the case.
- Let X be some large integer.



- This graph has a maximum flow of 2X, but Ford-Fulkerson might send one unit of flow through that middle edge forward or backward in every iteration. So Theta(X) time total.
- We can encode this graph using only O(log X) bits. So the running time may be exponential in the problem size!
- And this whole time, we've been assuming capacities are integers. If they are arbitrary real numbers, you can set up an example where this algorithm will **never** terminate. And the flows computed may not even converge to the real maximum flow.

Edmonds-Karp 1: Fat Pipes

- But the reason Ford-Fulkerson may be slow is that we never said which paths to augment along. If we choose more carefully, maybe we can find the maximum flow more quickly.
- Both algorithms I'll discuss were discovered by Edmonds and Karp in the 1970s.
- Edmonds-Karp: Choose the augmenting path with the largest bottleneck (so you can send as much flow as possible right now).
- You can find this path using a variant of Jarník's minimum spanning tree algorithm: Build a spanning tree from s in the residual graph, repeatedly adding edges of largest residual capacity that leave the tree.
- So O(E log V) time per iteration.
- So how many iterations are there?

- Let f be the current flow and f' be the maximum flow *in the current residual graph G_f*.
- Let e be the bottleneck edge in the current iteration, so we're about to push c_f(e) units of flow.
- S: vertices reachable with higher residual capacity than c_f(e) edges; T = V \ S.
- So (S, T) is an (s, t)-cut and every edge spanning it has capacity at most c_f(e).
- ||S, T|| ≤ |E| * c_f(e). But |f'| ≤ ||S, T||. so c_f(e) ≥ |f'| / |E|.
- So pushing down the maximum-bottleneck path multiplies the residual maximum flow value by (1 - 1 / |E|) or less.
- After |E| * ln |f^* | iterations, the residual value of the maximum flow is at most |f^* | * (1 - 1 / |E|)^(E * ln |f^* |) < |f^* | * exp(-ln |f^* |) = 1.
- In other words, we can't do another augmentation after |E| * ln |f^* | iterations if the capacities are integers, because there won't be an integral amount of flow left to push.
- The total running time *assuming integer capacities* is O(E^2 log V log |f^* |).
- This running time *is* polynomial in the problem size.

Edmonds-Karp 2 (and Dinits): Short Pipes

- Edmonds-Karp (again): Choose an augmenting path with the fewest number of edges.
- Can be found in O(E) time by running a breadth-first search in the residual graph.
- Now to bound the number of iterations.
- Let f_i be the flow after i iterations, and G_i = G_{f_i}. We have f_0 is zero everywhere and G_0 = G.
- Let level_i(v) be the unweighted shortest path distance from s to v in G_i.
- Lemma: level_{i+1}(v) ≥ level_i(v) for all vertices v and non-negative integers i.
  - We'll do induction on the shortest path distance from s in G_{i+1}.
  - level_i(s) = 0 for all i. Check.
  - Let s ➡ … ➡ u ➡ v be a shortest path to v in G_{i+1}.
  - If u ➡ v is in G_i, then level_i(v) ≤ level_i(u) + 1.
  - If u ➡ v is not in G_i, then we must have pushed along v ➡ u to create it for u ➡ v. Meaning v ➡ u was on the shortest s to t path. So level_i(v) = level_i(u) - 1 ≤ level_i(u) + 1.
  - Either way, level_{i+1}(v) =
    - level_{i+1}(u) + 1 because u ➡ v is on the shortest path
    - ≥ level_i(u) + 1 by the inductive hypothesis
    - ≥ level_i(v) from above cases
- Lemma: Any edge u ➡ v disappears from the residual graph at most V / 2 times.
  - Suppose u ➡ v is in G_i and G_{j+1} but not in G_{i+1}, …, G_j for some i < j.
  - u ➡ v must be in the ith augmenting path, so level_i(v) = level_i(u) + 1.
  - and v ➡ u must be in the jth augmenting path, so level_j(v) = level_j(u) - 1.

  - So, level_j(u) = level_j(v) + 1 ≥ level_i(v) + 1 = level_i(u) + 2.
  - So the distance from s to u increased by 2 between the disappearance and reappearance of u ➡ v. Every level is less than V or infinite (if there is no path to u), so an edge can disappear at most V / 2 times.
- There are E edges so E V / 2 disappearances total. Each augmentation makes its bottleneck edge disappear, so there are at most E V / 2 iterations.
- The total running time is O(VE^2).
- And this running time is correct even for arbitrary non-negative *real* number capacities.