CS 6363.005.19S Lecture 19—April 2, 2019

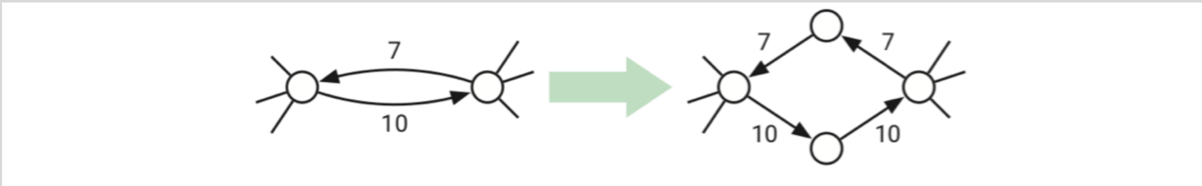
Main topics are #max_flow-min_cut.

Prelude

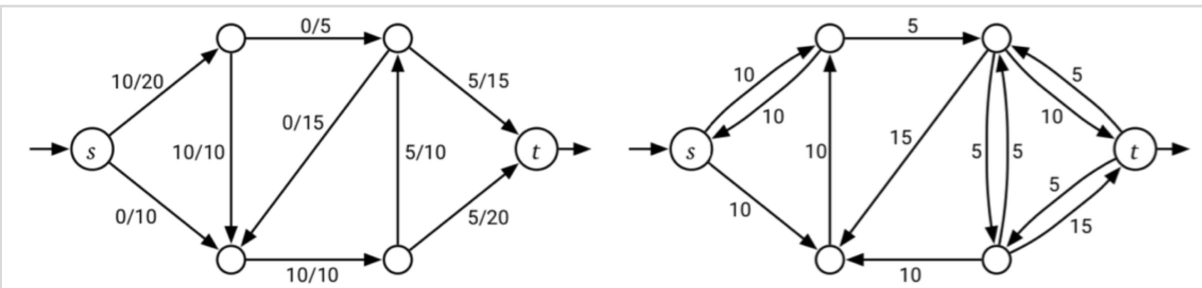
- Homework 4 is due Tuesday April 9.

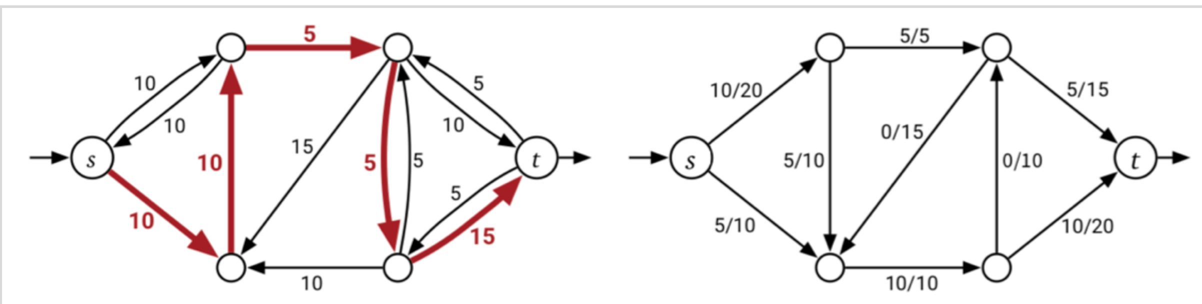
The Maxflow Mincut Theorem

- Last time, we discussed the maximum flow and minimum cut problems.
- We're given a directed graph G = (V, E, c) with capacity function c : E → R_≥0 and two vertices s and t. For simplicity, I'll write c(u → v) = 0 even in cases where u → v is not in E.
- 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)
- 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 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)
- The *minimum cut problem* is to compute an (s, t)-cut with minimum capacity.
- The Maxflow Mincut Theorem: The value of the maximum (s, t)-flow is equal to the capacity of the 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, 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 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 $c\_f : V \times V \rightarrow R$ is based on flow f.
- $c\_f(u \rightarrow v) =$
  - $c(u \rightarrow v) - f(u \rightarrow v)$ if $u \rightarrow v$ in E (or $c(u \rightarrow v) > 0$)
  - $f(v \rightarrow u)$ if $v \rightarrow u$ in E (or $c(v \rightarrow u) > 0$)
  - 0 otherwise
- Remember, we're assuming no pair of edges $u \rightarrow v$ and $v \rightarrow u$ have positive capacity, so only one of those cases holds.
- Since $f(u \rightarrow v) \geq 0$ and $f(u \rightarrow v) \leq c(u \rightarrow v)$, the residual capacities are non-negative.
- But, we may have $c\_f(u \rightarrow v) > 0$ even if $u \rightarrow v$ 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 $s = v_0 \to v_1 \to \dots \to v_r = t$ in G_f.
  - We call this path an *augmenting path*. We'll see why in a second.
  - Let $F = \min_i c_f(v_i \to v_{i+1})$ 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 $f' : E \to R$ where $f'(u \to v) =$
    - $f(u \to v) + F$ if $u \to v$ is in the augmenting path
    - $f(u \to v) - F$ if $v \to u$ is in the augmenting path
    - $f(u \to v)$ otherwise
  - Again, graph G's edges are reduced, so exactly one case holds.
  - Above is what pushing 5 units of flow along an augmenting path looks like.
  - 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 $u \to v$ is on the augmenting path,
      - $f'(u \to v)$
        - $= f(u \to v) + F$ by definition of f'
        - $> f(u \to v)$
        - ≥ 0.
      - Also $f'(u \to v)$
        - $= f(u \to v) + F$ by definition of f'
        - $\leq f(u \to v) + c_f(u \to v)$ by definition of F
        - $= f(u \to v) + c(u \to v) - f(u \to v)$ by definition of c_f
        - $= c(u \to v)$
    - If $v \to u$ is on the augmenting path,
      - $f'(u \to v)$
        - $= f(u \to v) - F$ by definition of f'
        - $< f(u \to v)$
        - $\leq c(u \to v)$.
      - Also, $f'(u \to v)$
        - $= f(u \to v) - F$ by definition of f'
        - $\geq f(u \to v) - c_f(v \to u)$ by definition of F
        - $= f(u \to v) - f(u \to v)$ 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 $T = V \setminus S$.
- Partition (S, T) is an (s, t)-cut, and for every u in S and v in T, we have
  - $0 = c_f(u \rightarrow v) = c(u \rightarrow v) - f(u \rightarrow v)$ if $u \rightarrow v$ in E and
  - $0 = c_f(u \rightarrow v) = f(v \rightarrow u)$ if $v \rightarrow u$ in E
- In other words, f saturates every edge from S to T and avoids every edge from T to S.
- Remember from last Thursday that statement implies $|f| = \|S, T\|$. 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.

- 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 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.

Analysis and Picking Paths

- So there will always be an augmenting path if we don't have a maximum flow.
- And that suggests an algorithm: initially start with 0 flow on every edge. Then iteratively compute the residual graph and find an augmenting path for every new flow we create. The value of our flow will increase in each iteration.
- But will this process actually reach the maximum flow?
- First, let's assume all the capacities are integers. This has a few repercussions.
  - The initial flow is all integers since 0 is an integer.
  - If we assume inductively that f is all integers, then all the residual capacities are integers.
  - Meaning F is a positive integer.
  - Meaning f' is all integers and $|f'| \geq |f| + 1$.
- So if f^* is the maximum flow, we do at most $|f^*|$ augmentations and f^* is all integers.
- We can build the residual graphs in O(E) time each, so these $|f^*|$ augmentations take $O(E |f^*|)$ time total.
- But there's two issues with this analysis.
- $O(E |f^*|)$ means we have a *pseudo-polynomial time* algorithm. It runs in time polynomial in E and $|f^*|$, but $|f^*|$ may not be polynomial in the input size.
- In fact, we can write down capacities of size 2^X using only X bits, so the running time may actually be exponential in the input size.
- The algorithm is often efficient in practice, though, or in situations where you can guarantee $|f^*|$ is small.
- The other issue with this analysis is that we're assuming the capacities are integers. But flows and capacities are still well-defined using real numbers, and the maxflow-mincut

theorem is still true.

- Using irrational capacities, you can set up examples where every augmentation gets smaller and smaller and smaller. You always get higher value flows, but you never get a maximum flow. There's not even a guarantee that you'll approach a maximum flow value in the limit!
- Of course, computers don't actually store real numbers, but if your floating point additions or comparisons start doing rounding, you may actually enter an infinite loop where you never make real progress on increasing the flow value!
- But here's the trick. We get to choose which augmenting paths to use. If we pick carefully, maybe the algorithm will run faster.
- Both of the following algorithms were discovered by Edmonds and Karp (and others) in the 1970s.

Edmonds-Karp 1: Fat Pipes

- Edmonds-Karp: Choose the augmenting path with the largest bottleneck F (so you can send as much flow as possible right now).
- You can find this path using a variant of the Prim-Jarník 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 to find each augmenting path.
- So how many augmenting paths 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

$$|f^*| \cdot (1 - 1/E)^{E \cdot \ln|f^*|} < |f^*| e^{-\ln|f^*|} = 1.$$

- 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, but it still assumes integer capacities.

Edmonds-Karp 2: 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 $\text{level}_i(v)$ be the unweighted shortest path distance from s to v in G_i.
- Lemma: $\text{level}_{i}(v) \geq \text{level}_{i-1}(v)$ for all vertices v and non-negative integers i.
  - We'll do induction on $\text{level}_i(v)$.
  - $\text{level}_i(s) = 0 = \text{level}_{i-1}(s)$. Check.
  - If we cannot reach v from s $\text{level}_i(v) = \infty \geq \text{level}_{i-1}(v)$. Check.
  - Otherwise, let $s \rightarrow \ldots \rightarrow u \rightarrow v$ be a shortest path to v in G_{i}.
  - $\text{level}_i(v) = \text{level}_{i}(u) + 1$, so the induction hypothesis shows $\text{level}_i(u) \geq \text{level}_{i-1}(u)$.
  - If $u \rightarrow v$ is in G_{i-1}, then $\text{level}_{i-1}(v) \leq \text{level}_{i-1}(u) + 1$.
  - If $u \rightarrow v$ is not in G_{i-1}, then we must have pushed along $v \rightarrow u$ to create residual capacity in $u \rightarrow v$. Meaning $v \rightarrow u$ was on the shortest s to t path. So $\text{level}_{i-1}(v) = \text{level}_{i-1}(u) - 1 \leq \text{level}_{i-1}(u) + 1$.
  - Either way, $\text{level}_i(v) = \text{level}_i(u) + 1 \geq \text{level}_{i-1}(u) + 1 \geq \text{level}_{i-1}(v)$
- Lemma: Any edge $u \rightarrow v$ disappears from the residual graph at most $V / 2$ times.
  - Suppose $u \rightarrow v$ is in G_i and G_{j+1} but not in $G_{i+1}, \ldots, G_j$ for some $i < j$.
  - $u \rightarrow v$ must be in the ith augmenting path, so $\text{level}_i(v) = \text{level}_i(u) + 1$.
  - and $v \rightarrow u$ must be in the jth augmenting path, so $\text{level}_j(v) = \text{level}_j(u) - 1$.
  - So, $\text{level}_j(u) = \text{level}_j(v) + 1 \geq \text{level}_i(v) + 1 = \text{level}_i(u) + 2$.
  - So the distance from s to u increased by 2 between the disappearance and reappearance of $u \rightarrow v$. 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.

- There have been many more algorithms discovered since these. The fastest one known today was described by Orlin in 2012 and runs in $O(VE)$ time.
- Very few people understand this algorithm. I'm afraid I am not one of them.
- But for the purposes of doing homework or exams, you should feel free to cite it.
- **Maximum flows and minimum cuts can be computed in $O(VE)$ time.**