LECTURE 13 (October 14th)

Flow Decompositions

Recall the set up from the last lecture about maximum flows & minimum cuts. Here we are trying to find some other optimization structure in a graph

The input is a flow network which is a directed graph $G=(V,E)$, two vertices s and t and capacities $c(e) \geq 0$.

If we think of the edges as pipes and capacities as the capacity of the pipe, the maximum flow problem is asking how quickly can we inject water into the source vertex s and pull it out of the target vertex t.

So, the idea is that we want to compute another function called the flow $f: E \to \mathbb{R}_{\geq 0}$ that assigns non-negative values to each edge and satisfies the following constraints

① $f(u \to v) \geq 0$ — non-negativity constraint

② $f(u \to v) \leq c(u \to v)$ — capacity constraint

③ $\sum_v f(u \to v) = \sum_v f(v \to u)$ for all $u \neq s, t$ — conservation constraint

The goal is to maximize the net flow

$$|f| = \underbrace{\sum_w f(s \to w)}_{\text{flow coming out of } s} - \underbrace{\sum_u f(u \to s)}_{\text{flow coming into } s}$$

$$= \underbrace{\sum_u f(u \to t)}_{\text{flow coming into } t} - \underbrace{\sum_w f(t \to w)}_{\text{flow coming out of } t}$$

So, we are not looking for a combinatorial structure like shortest path or minimum spanning tree but a global function that describes how stuff moves through the network

So, one of the homework questions in the next homework will ask you to figure out when the maximum flow is unique since in general there may be several maximum flows in a network

So, the other piece of this puzzle is the minimum cut problem.

Here we want to compute a partition $V = S \cup T$ where S & T are disjoint and cover all vertices s.t. $s \in S$, $t \in T$ and we minimize

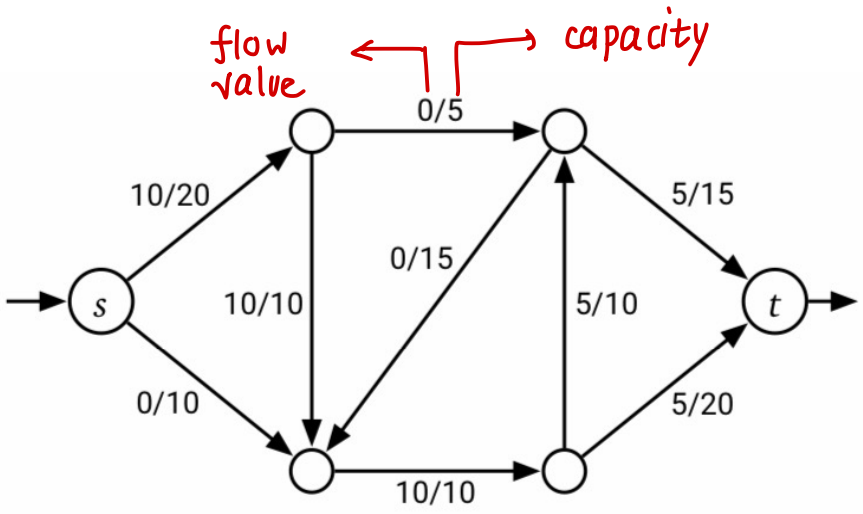
$$\|S, T\| = \sum_{u \in S} \sum_{v \in T} c(u \to v)$$

In the last lecture, we saw the max-flow min-cut theorem

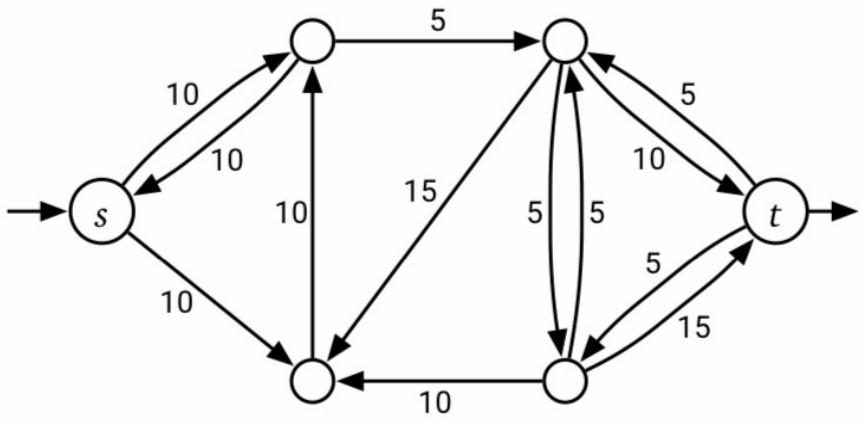
$$|f| = \|S, T\|$$

which also gave us an algorithm to find the maximum flow.
This is the Ford-Fulkerson algorithm. Let us recall how it works

Consider the following flow network and a flow in it. Ford-Fulkerson first builds the residual graph of the flow.

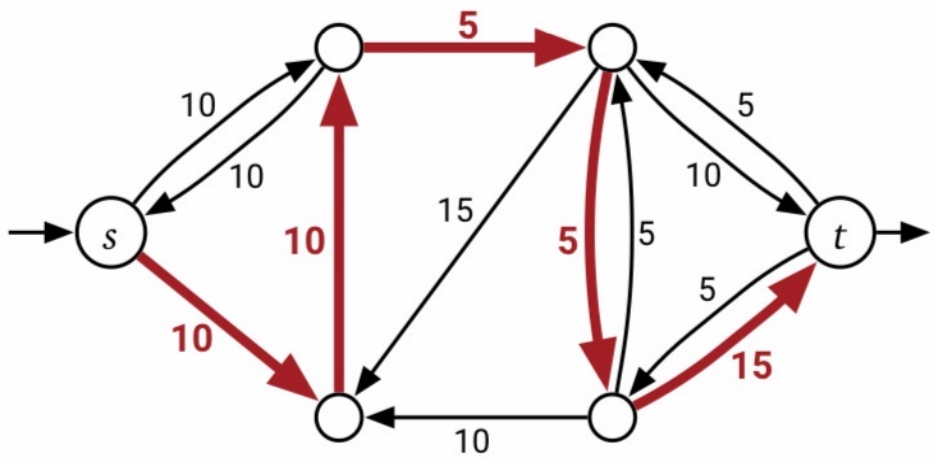


The value of the flow is 10

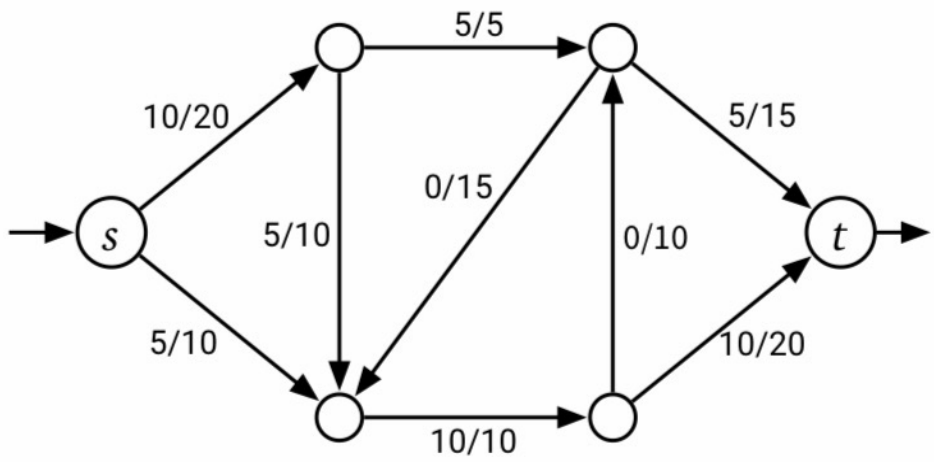


Residual graph of the flow

Next, the algorithm looks at a path from s to t in the residual graph. This is called the augmenting path. The algorithm then pushes as much flow along this path as possible, which equals the minimum capacity along this path.



Augmenting path



New flow

The proof of the max-flow min-cut theorem boils down to the following

- if there is no augmenting path from s to t, then we already have a maximum flow
- If we take S = all vertices reachable from s and T = everything else, then this gives a cut which has the same value as the current flow and certifies that we have both a maximum flow and a minimum cut.

So, Ford-Fulkerson algorithm is

Augmenting Paths Algorithm

Initialize $f \leftarrow 0$

$G_f \leftarrow G$

While there is a path p from s to t in G_f
  push flow along p
  rebuild G_f

Return f

So, what we saw last time is that *if this algorithm halts*, it returns a maximum flow, but does it always halt?

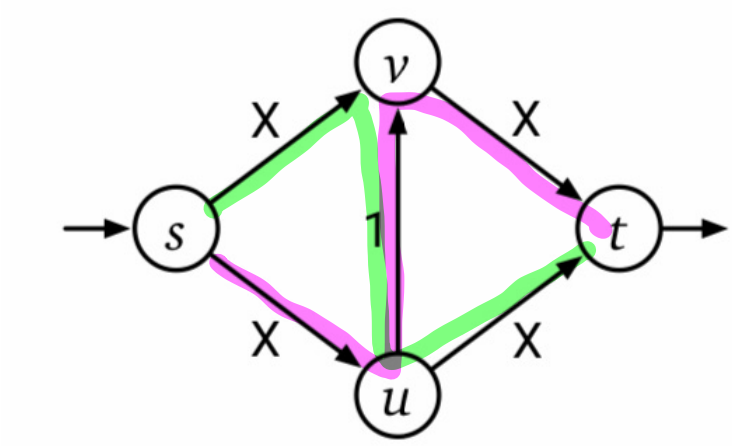
We have the following integrality theorem:

If all capacities are integers, Ford-Fulkerson always returns an integer maximum flow.

The proof of this is via induction. Every time the algorithm finds a new flow its value is always an integer that's larger but there is a finite cap, so it must halt after $|f^*|$ iterations, where $|f^*|$ is the maximum flow value.

Thus, the running time of Ford-Fulkerson is $O(|E| \cdot |f^*|)$.

However, consider the following example where X is a very large integer



The maximum flow value here is $2X$.

However, Ford-Fulkerson does not specify which augmenting path to choose so, if an adversary always picks the path to make it run as slowly as possible, it will always choose the highlighted red and green paths alternating them. You can see that for this particular flow network, the number of iterations is exactly $2X$. So, Ford-Fulkerson can take $|f^*|$ iterations if we do not specify how to choose the augmenting paths.

However, for the above graph, how much information are we given — the input takes $O(\log X) + O(1)$ bits to describe. So, running time of Ford-Fulkerson, which is X, is exponential in the input size.

So, the obvious response is we choose our paths badly in this algorithm. Can we choose them in a clever way?

[1] Among all augmenting paths, choose the path with the maximum capacity This runs in time

$$O(|E|^2 V \cdot \log |f^*|)$$

We will discuss this in a second but note that still depends on the flow and requires integer capacities to work.

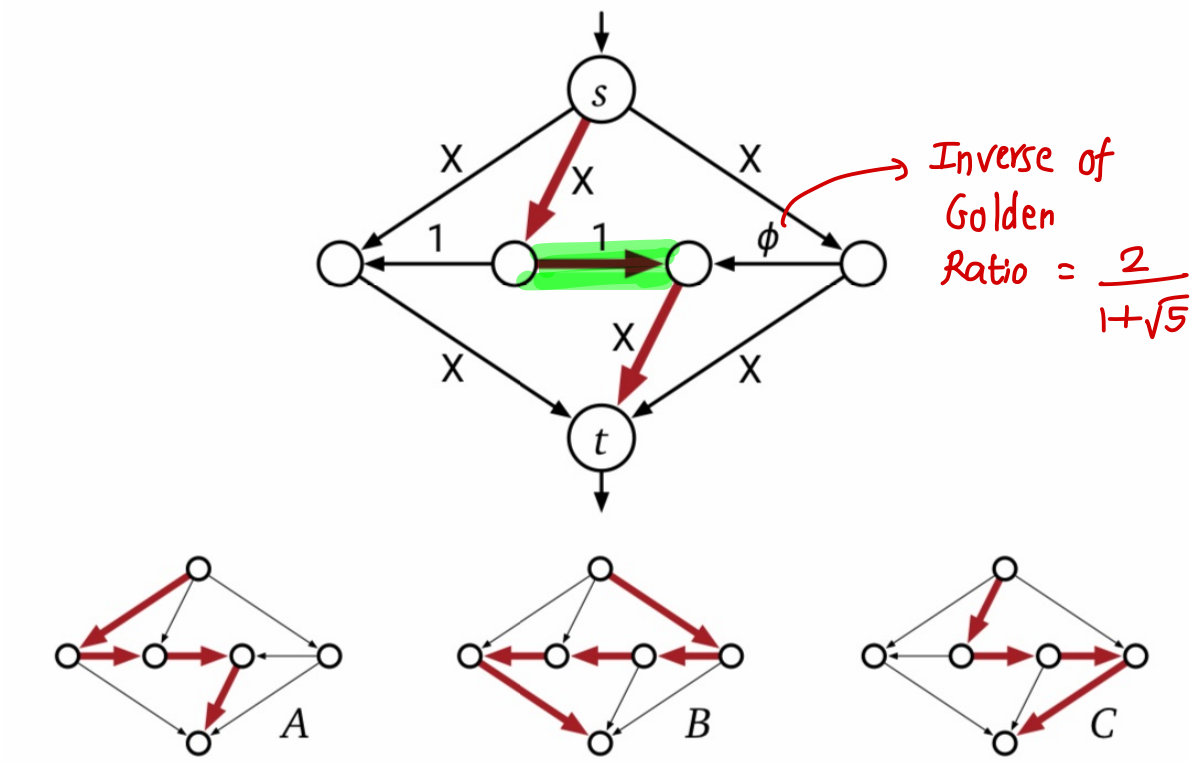
Finding the fattest augmenting path can be done with a small modification to Dijkstra's algorithm in near-linear time. (Instead of looking at the minimum path length, we look for maximum among the minimum edges) The difficult part here is to analyze the number of iterations.

[2] Another way of choosing an augmenting path is to take the shortest augmenting path (i.e. the minimum number of edges, not weights). This can be done in linear time and it does not depend on the flow value

This is called the Edmonds-Karp algorithm.

But the above still uses that the capacities are integers. What if we allow irrational capacities? Here we allow the arithmetic operations to be done by some subroutine which is quite natural.

However, in this case Ford-Fulkerson may never halt as the following example shows:



Inverse of Golden Ratio = $\frac{2}{1+\sqrt{5}}$

This is still a well-behaved irrational number and we can still do arithmetic operations in O(1) time.

But consider the residual capacity of the green edge here while we do an infinite sequence of augmentations.

The first path we augment is highlighted by red in the top figure. This adds one unit of flow. Once we have done this the green edge is now pointing to the left in the residual graph. Now we augment along path B. It flips the middle edge again. Now we can use path C which flips it again. Then B again followed by A. So, the pattern is

Top, B, C, B, A, B, C, B, A, ...

The math is in the notes, but it turns out every time after the first augmentation if we do this the flow increases by the following values

Top, B, C, B, A, B, C
1 ϕ ϕ ϕ^2 ϕ^2 ϕ^3 ϕ^3

So, the number get smaller and smaller and even if we run this forever the total flow value is at most

$$1 + 2\sum_{i=1}^{\infty} \phi^i < 7$$

However, it is easy to see that the maximum flow is $2X+1 \gg 7$.

So, Ford-Fulkerson does not even work here, again if we are not careful how we are choosing the path.

If we use the fattest augmenting path here, it turns out that we can get stuck in an infinite loop but at least we converge to the right answer

If we use the shortest augmenting path, we always terminate in polynomial time.

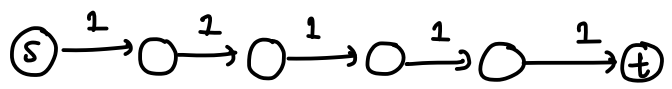
It turns out that max flow can be solved in $O(|V| \cdot |E|)$ time → You can use this for homework & exams. In fact, there is major progress recently in some cases!

Note that if all capacities are one, $O(|V| \cdot |E|)$ is the running time of Ford-Fulkerson. To explain, why this is a natural time bound in general, let us look at Flow Decompositions.

Flow Decompositions

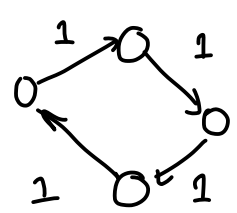
There are two different ways of describing a flow.

- The first one we saw before, a flow is a function on edges satisfying non-negativity, capacity & conservation constraints.
- The second way of describing flows that is very useful for applications is to think of flows as a sum of paths & cycles.



Given any path from s to t, we can think of this path as encoding a flow with value one.

But it is also true that cycles are flows – if we send one unit of flow along any cycle in the graph it is a flow.



So, if we have two flows f & f' in the graph

$f + f'$ is also a flow
αf is also a flow for any scalar $\alpha > 0$.

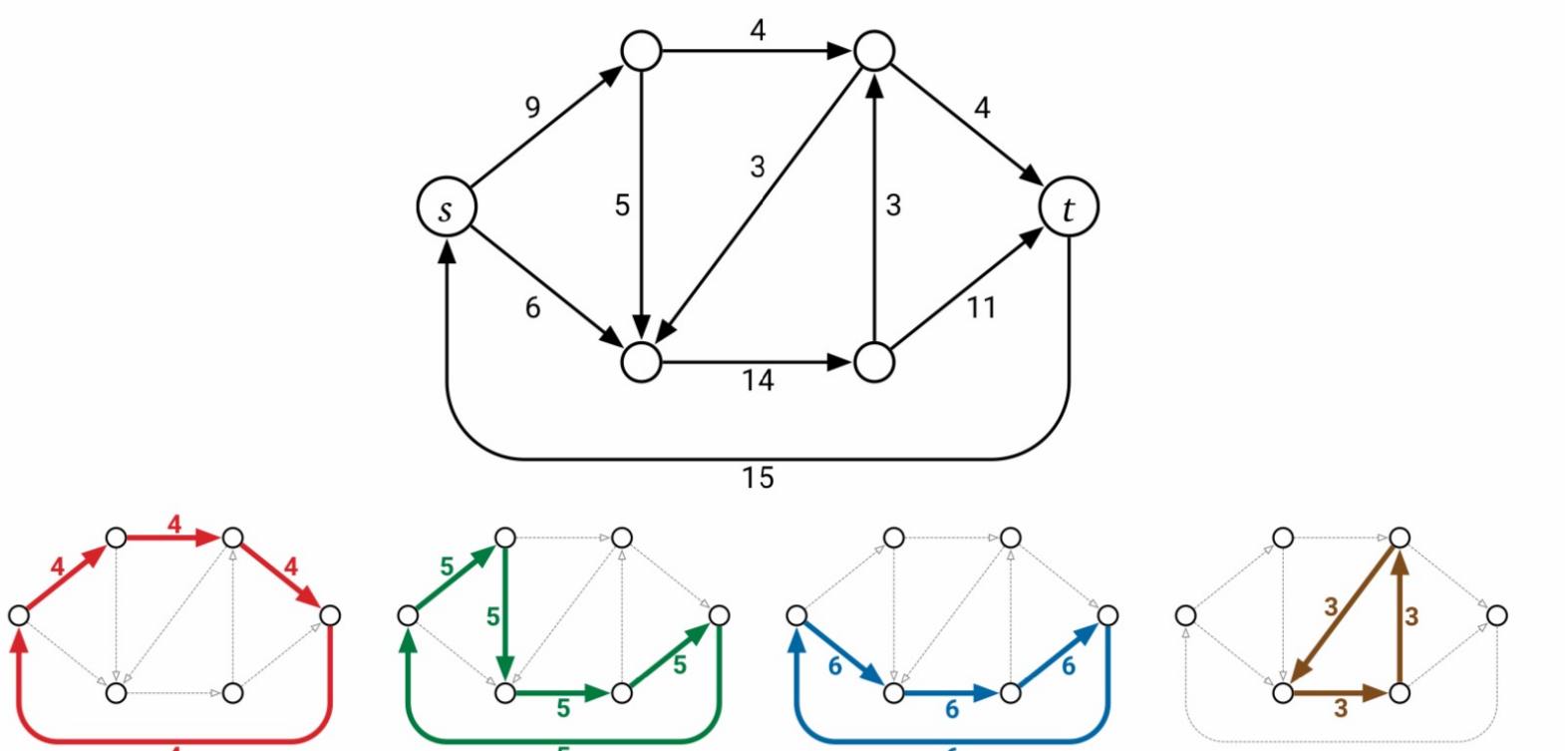
Above we are ignoring the capacity constraints (you can think of them as ∞ for the time being)

So, one way to think about this is that the set of paths & cycles give me a way to decompose a flow into elementary pieces with simple combinatorial structure.

The easiest way to describe this is considering only flows with value zero. This can be accomplished by adding extra edges in the flow network. For example, if we ignore the edge from t to s with flow 15 below, this is describing a flow with value 15 from s to t, but we have added that one edge to push the flow along that edge. Now, the conservation constraint holds at every vertex including s & t. The net flow is zero. This is what is referred to as a circulation.

Here is a way to decompose any circulation in a set of cycles. It is a simple algorithm — look for any cycle where every edge is carrying a positive amount of flow. Reduce the flow value as much as possible without making anything negative. Repeat until all edges have zero flow values.

We are doing something like Ford-Fulkerson but ignoring capacities at the moment. (But no residual graphs here)



Thus, any circulation can be decomposed into $\leq |E|$ cycles in $O(|V| \cdot |E|)$ time.

In general, a flow will decompose into sum of paths & cycles by ignoring the extra edge we added.

Thus, any flow is a weighted sum of flows & cycles. If we throw away all the cycles, it does not change the value of the sum So, we have reduced to the case of a flow where the edges that carry flow define a DAG. The value of the flow is the sum of weights of the paths. This means that there is an acyclic max flow. Moreover, if the flow is integral, the weights are integral.

The above also gives an algorithm to decompose the flow into weighted sum of paths & cycles in $O(|V| \cdot |E|)$ time. There are atmost $O(|E|)$ components & finding each one takes $O(|V|)$ time.

There are flow where any flow decomposition has total complexity $\Omega(VE)$. Thus, any algorithm that constructs flow, one path or cycle at a time will take $\Omega(|V| \cdot |E|)$ time. In particular, any variant of Ford-Folkerson runs in $\Omega(|V| \cdot |E|)$ time.

For the fattest augmenting path algorithm, one can show that each the fattest path must carry $1/|E|$ fraction of the flow so, one can show that the running time is $O(|E|^2 V \log |f^*|)$ with some work.

For the shortest augmenting path, as we run the algorithm various directed edges disappear from the residual graphs because they are saturated and then later reappear because we push flow backwards. Each time we remove an edge and put it back again, the distance between s and t has to increase, so for every edge there are $O(|V|)$ times where this happens. To do this for all edges, we need $O(|V| \cdot |E|)$ iterations. Each iteration requires $O(|E|)$ time to find the shortest augmenting path, so this takes $O(|V| \cdot |E|^2)$ time.

Do we need to compute all the flow decompositions to compute the max flow? This was a long standing open problem and there has been a lot of progress

- Orlin in 2012 gave an $O(|V| \cdot |E|)$ time algorithm
- Chen et al. in 2022 gave an algorithm $O(|E|^{1+o(1)} \log U)$ time where U is the maximum capacity
- Bernstein et al. in 2024 gave an algorithm that runs in $O(|V|^{2+o(1)} \log U)$ time. This algorithm is under the hood an augmenting path algorithms.