Math 3272: Linear Programming[1] Mikhail Lavrov

# Lecture 21: The Ford–Fulkerson algorithm

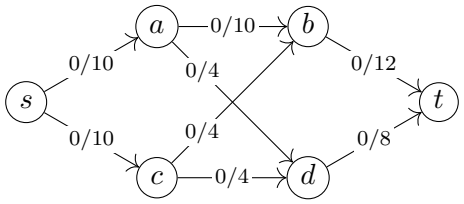
*October 27, 2022* *Kennesaw State University*

## 1 Greedily increasing flow

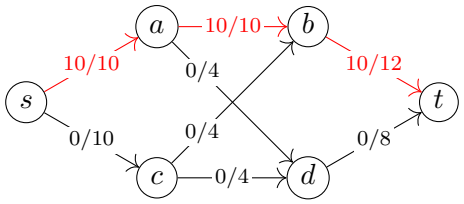
We've found a linear program for max-flow problems. But it turns out that there are better ways to solve this problem than by using a linear program. In the next few lectures, we'll explore some of these approaches.

To build intuition, we'll start by discussing an approach that seems like it ought to work, but doesn't quite finish the job. The idea here is to find directed paths from $s$ to $t$ along which we can increase the flow, and just—keep doing that.

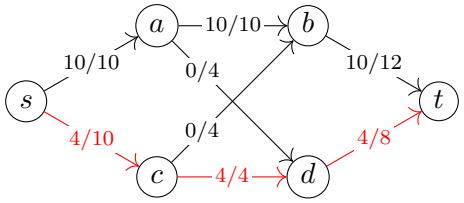
Here's a reasonably typical network that we can use as an example. (As usual, I'll mark an arc with a fraction $x/c$ to represent that $x$ flow is being sent along that arc, and it has a maximum capacity of $c$.)




Our first step is to notice the promising path that goes $s \to a \to b \to t$. The arcs along this path all have capacity at least 10, so we can send 10 flow along this path:

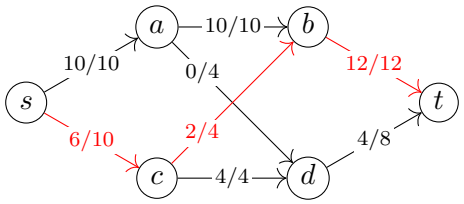



Similarly, there is the path $s \to c \to d \to t$. This one doesn't, let us make as much progress, but the bottleneck capacity along that path is 4, so we can send 4 flow along this path:




[1]This document comes from the Math 3272 course webpage: https://facultyweb.kennesaw.edu/mlavrov/courses/3272-fall-2022.php

And there is room to send 2 more flow along the path $s \to c \to b \to t$ (but no more, because the arc $(b, t)$ reaches its capacity of 12 when we do so):




At this point, we seem to be done. There are no paths that go from $s$ to $t$ along which we can increase the flow.

However, this still not the maximum flow: there are flows with larger value. It's just that we've gotten stuck at a feasible flow where we can't increase all the flows without decreasing all the flows. We need a better strategy.

(Disclaimer: if we had chosen different paths from $s$ to $t$ to use at each step, we could have gotten to the maximum flow. But the point is that we don't know which paths are the right ones, so we need to be smarter than that.)

## 2 An augmenting path

If you try to describe how we can improve the flow in the most recent step, you might say something like "we want to send less flow from $a$ to $b$, and more of it to $d$. This lets us increase the flow from $c$ to $b$." So, there's a super complicated explanation, and you can imagine that making arguments like this would get pretty tricky when the network is bigger.

There's a trick to describing such improvements that's simpler to think about, while still being powerful enough that we can always use it to get to the maximum flow. That trick is to find **augmenting paths**, which are slightly more general paths from $s$ to $t$.

In an augmenting path, we go from $s$ to $t$, but we're allowed to ignore the directions of the arcs: we can travel both forward and backward. For example, the following is a valid augmenting path:

$$s \xrightarrow{6/10} c \xrightarrow{2/4} b \xleftarrow{10/10} a \xrightarrow{0/4} d \xrightarrow{4/8} t$$

Notice that the arc from $a$ to $b$ is being traversed in the "wrong" direction.

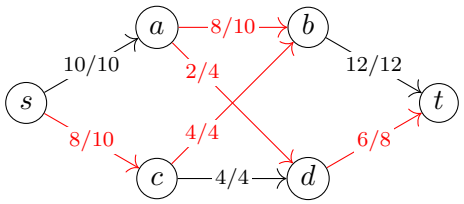
An augmenting path is not allowed to cheat however you like, however. The rule is:

- You can go *forward* along an arc if it's still below its maximum capacity.
- You can go *backward* along an arc if it's at positive capacity.

If we find an augmenting path, we can let $\delta$ be the smallest margin by which it's valid: the smallest amount by which forward arcs are below maximum capacity, or backward arcs are above zero capacity. In this example, $\delta = 2$: we can't increase the flow from $c$ to $b$ by more than 2, because $x_{cb} = 2$ and $c_{cb} = 4$.

Then we can modify our current feasible flow by adding $\delta$ flow along every forward arc of the path, and subtracting $\delta$ flow along every backward arc. In our example, we increase the flow $x_{sc}$ by 2,

increase the flow $x_{cb}$ by 2, *decrease* the flow $x_{ab}$ by 2, increase the flow $x_{ad}$ by 2, and increase the folow $x_{dt}$ by 2. Here is the result, in diagram form:




This is called "augmenting a flow along a path".

It is worth checking that this operation still gives a feasible flow: in other words, that flow conservation is still satisfied at each node other than $s$ and $t$. The good thing is that even in very complicated networks, there are only four cases to consider for how a node is affected by this augmenting step:

1. Suppose we have a node $\cdots \to p \to \cdots$ along the path, visited "in the normal way": we take a forward arc into $p$, and a forward arc out.

   In this case, if we augment by $\delta$, the total flow into $p$ will increase by $\delta$, and so will the total flow out of $p$. Assuming flow was conserved before augmenting, it is still conserved.

2. Now suppose our augmenting path visits some node $q$ in a $\cdots \to q \leftarrow \cdots$ pattern: the arc before $q$ is a forward arc, but the arc after $q$ is a backward arc.

   In this case, if we augment by $\delta$, the forward arc's flow will increase by $\delta$ and the backward arc's flow will decrease by $\delta$. Both arcs contribute to the flow *into* $q$, so the total flow into $q$ will not change. The flow out of $q$ was not affected at all, so flow is still conserved at $q$.

3. If a node $r$ is visited in a $\cdots \leftarrow r \to \cdots$ pattern, something similar to case 2 happens.

   Here, both arcs represent flow out of $r$, and the total change in the flow out of $r$ is 0. Meanwhile, flow into $r$ is unaffected, so flow is still conserved at $r$.

4. Finally, a node $s$ visited in a $\cdots \leftarrow s \leftarrow \cdots$ pattern, where both arcs are backward arcs, is similar to case 1.

   Here, the flow along both arcs decreases by $\delta$. So the total flow into $r$ and the total flow out of $r$ decrease by the same amount. Again, flow is still conserved.

## 3 The residual graph

Augmenting paths are convenient to have, but hard to find. To try to do this systematically, we'll construct a **residual graph**.

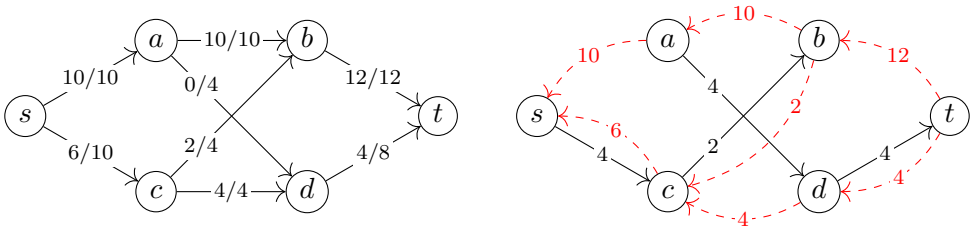
In a residual graph, the nodes will remain the same, but the arcs we consider are different. We will add every arc that corresponds to a direction that an augmenting path could go. Namely:

- We keep an arc of our network in the residual graph if it's still below maximum capacity. We label it with its **residual capacity**, which is the amount of room still left to increase the flow. (For an arc $(i, j)$, the residual capacity is defined to be $c_{ij} - x_{ij}$.)

- When an arc of our network has positive flow, we add a reverse arc to the residual graph. We also label it with its residual capacity, which is now the amount of room left to decrease the flow. If the arc of our network was $(i, j)$ with flow $x_{ij}$, the residual capacity of the reverse arc $(j, i)$ in the residual graph is also $x_{ij}$.

To see how this helps us find augmenting paths, let's go back to the previous step. On the right is the feasible flow we had at that point; on the left is the residual graph. (The reverse arcs in the residual graphs are drawn as red dashed lines to distinguish them.)

Here is how this shakes out for the flow we had before the last augmenting step we did. (Forward arcs in the residual graph are drawn in black, backward arcs in red.)




In the residual graph, finding an augmenting path is just a matter of of "getting through the maze": we want to see if there is a way to follow the arcs in the residual graph (in their proper directions) to get from $s$ to $t$.

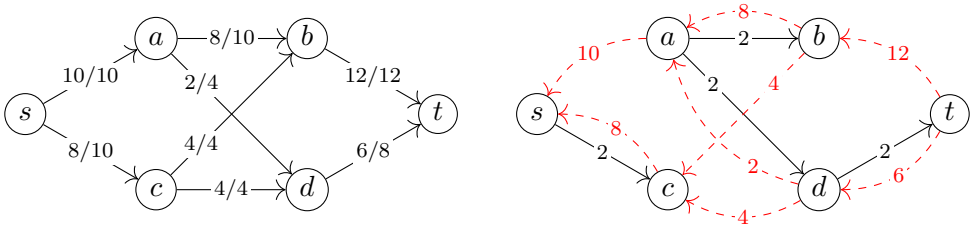
For example, we see that there is a path

$$s \xrightarrow{4} c \xrightarrow{2} b \xrightarrow{10} a \xrightarrow{4} d \xrightarrow{4} t$$

in the residual graph. That corresponds the same augmenting path we found earlier, but now we don't have to follow arcs in unnatural directions to see it. What's more, the value of $\delta$ is now easy to find: it's just the smallest residual capacity along this path.

## 4 Residual graphs and minimum cuts

But now, suppose we draw the residual graph for the flow we get *after* our most recent augmenting path. Here is that flow, and here is its residual graph:




This residual graph is a impossible "maze", and it doesn't take us long to discover this. From node $s$, we can only get to node $c$; from node $c$, we can only return to node $s$. As a result, there is no augmenting path to find for this flow.

This seems disappointing, but actually it's expected, and it's what we want to see. Here's why.

**Theorem 1.** *Suppose that there is no path from $s$ to $t$ in the residual graph (for some feasible flow $\mathbf{x}$ in some network). Then:*

- *The flow $\mathbf{x}$ is a maximum flow.*
- *Let $S$ be the set of all nodes reachable from $s$ in the residual graph (including $s$ itself). Let $T$ be the set of all other nodes. Then $(S, T)$ is a minimum cut, and has capacity equal to the value of $\mathbf{x}$.*

Before we prove the theorem, we can verify that this is true in our network. The value of our flow is 18: that's the total flow leaving $s$ ($10+8$), and also the total flow entering $t$ ($12+6$). Meanwhile, if $S$ is the set of all nodes reachable from $s$ in the residual graph, then $S = \{s, c\}$, which means $T = \{a, b, d, t\}$. The capacity of the cut $(S, T)$ is $c_{sa} + c_{cb} + c_{cd} = 10 + 4 + 4 = 18$.

## 5 Proof of Theorem 1

Things work out in the example above: since the capacity of the cut $(S, T)$ agrees with the value of $\mathbf{x}$, both are optimal. Why is this true in general?

In the previous lecture, when we were proving that the capacity of a cut gives an upper bound on the value of a flow, we showed the following:

$$v(\mathbf{x}) = \sum_{j:(s,j)\in A} x_{sj} - \sum_{i:(i,s)\in A} x_{is} = \sum_{i\in S}\sum_{j\in T} x_{ij} - \sum_{i\in T}\sum_{j\in S} x_{ij} \leq \sum_{i\in S}\sum_{j\in T} c_{ij} - \sum_{i\in S}\sum_{j\in T} 0 = c(S, T).$$

On the left, we have (by definition) the value of the flow $\mathbf{x}$. By some algebraic manipulation, we showed that this is equal to the middle expression: the total flow crossing from $S$ to $T$, minus the total flow crossing from $T$ back to $S$. This is upper bounded by the expression on the right (by using $x_{ij} \leq c_{ij}$ on every term of the first sum, and $x_{ij} \geq 0$ on every term of the middle sum), and the expression on the right is just the capacity of the cut $(S, T)$.

Now let's think about what happens in the special case where $S$ is the set of all vertices reachable from the source in the residual graph. This means that there can be no residual arc from any node $i \in S$ to any node $j \in T$.

There are two kinds of residual arcs.

- Forward residual arcs $i \to j$ corresponding to arcs $(i, j)$ with $x_{ij} < c_{ij}$.

  If there are no such arcs from $S$ to $T$, then for every $i \in S$ and $j \in T$, we have $x_{ij} = c_{ij}$.

- Backward residual arcs $i \leftarrow j$ corresponding to arcs $(i, j)$ with $x_{ij} > 0$.

  If there are no such arcs from $S$ to $T$, then for every $i \in T$ and $j \in S$, we have $x_{ij} = 0$.

As a result, for the cut we get from the residual graph, the $\leq$ inequality is actually a $=$ equation. We replaced $x_{ij}$ by $c_{ij}$ only in cases where we already had $x_{ij} = c_{ij}$, and we replaced $x_{ij}$ by $c_{ij}$ only in cases where we already had $x_{ij} = 0$.

Therefore the value of the flow $\mathbf{x}$ is equal to the capacity of the cut $(S, T)$.

The optimality of both of the flow and the cut follows. We know the capacity of the cut $(S, T)$ is an upper bound on the value of any flow; since $\mathbf{x}$ achieves that upper bound, it is a maximum flow

(no other flow can be better). Similarly, the value of $\mathbf{x}$ is a lower bound on the capacity of any cut: since $(S, T)$ achieves that lower bound, it is a minimum cut.

## 6 The Ford–Fulkerson algorithm

This leads us to an algorithm for trying to find a maximum flow in a network without using linear programming.

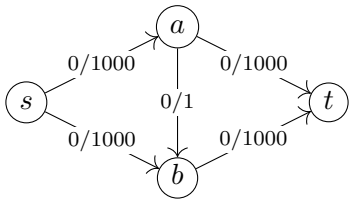
Start at a feasible flow: for example, the flow $\mathbf{x} = \mathbf{0}$. Then repeat the augmenting step we've developed:

1. Construct the residual graph for $\mathbf{x}$.
2. Attempt to find a path from $s$ to $t$ in the residual graph. If such a path exists, it gives us an augmenting path, which we use to improve $\mathbf{x}$ and go back to step 1.
3. If no such path exists, we use Theorem 1 to obtain a minimum cut whose capacity is equal to the value of $\mathbf{x}$. We know $\mathbf{x}$ is optimal, and the cut gives us a certificate of optimality.

As with the simplex algorithm, there is one more thing left to worry about. Ford–Fulkerson lets us always keep getting a better flow, and only stops when we reach a maximum flow, but are we guaranteed to actually reach it?

Unfortunately, as stated so far, there is no such guarantee in general. We *can* say that we'll eventually be done in cases where all the capacities $c_{ij}$ are integers. In this case, at every step, the value of the flow increases by at least 1, and so eventually it will reach $\sum_{j:(s,j)\in A} c_{sj}$, which is an upper bound on the value of the maximum flow. Similarly, if all the capacities are rational numbers, the flow always increases by at least $\frac{1}{d}$, where $d$ is the greatest common denominator of all the capacities.

But this is a very bad upper bound. Consider the following example:




The maximum flow here has value 2000, which can be reached in just 2 steps. But if we have poor judgement, and alternate between the augmenting paths $s \to a \to b \to t$ and $s \to b \leftarrow a \to t$, we increase the flow by 1 at each step, and only finish is 2000 steps.

Things are even worse if some capacities are irrational numbers. Then there is an example in which a poor choice of augmenting paths means we never even get close to the maximum flow.

Fortunately, there is a simple rule to avoid this situation. If we always pick the shortest augmenting path available at every step, then it can be shown that we'll always reach the maximum flow after at most $n \cdot m$ steps (in a network with $n$ nodes and $m$ arcs). This refinement is called the Edmonds–Karp algorithm.