Section 8.3: Network Flow

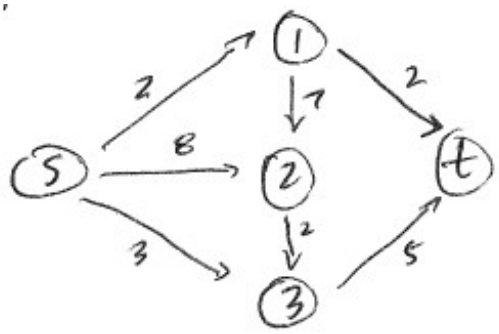
We can think of this as an abstraction of material flowing through the edges, to/from the vertices of a directed graph.

Setup

- Given graph $G = (V, E)$ with *source node* $s \in V$ and *terminal node* $t \in V$ (this is more common than our textbook notation of *so* and *si*).

  Note: A graph G is appropriate for a flow if it has (i) no parallel edges, (ii) no edge enters the source s, and (iii) no edge leaves the terminal t.

- For each edge in E, there is an associated non-negative integer c_e that represents the *capacity* of the edge. See the example below.



Flow

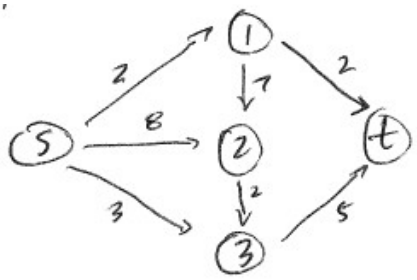
Intuitively, a flow will represent material flowing through the network from the source to the terminal. Each edge $e \in E$ will be assigned a real number f_e that represents how much stuff is being transported. Our textbook uses parentheses to denote the flow and the capacity: $(f_e)c_e$. For example, (2)4 would mean that the edge capacity is 4, and the flow uses 2. Let's define what we mean by flow.
Definition: A **flow** on a graph G is a function f such that:

- The domain is the set of edges, and the output is f_e, the flow on edge $e \in E$.
- The flow must be non-negative and not exceed the capacity of the edge: For each $e \in E$, $0 \leq f_e \leq c_e$.
- For every node (besides source and the terminal), the flow in must be equal to the flow out. That is, the flow is conserved.
- We might add that the flow out of the source must be equal to the flow into the sink. Our textbook adds an edge from the terminal to the source, but that is not common, and I'll typically not put that in.
- Additionally, the **value of the flow** is the total flow out of the source.

In the *Max Flow Problem*, we need to determine f_e for each edge e that maximizes the value of the flow (and satisfies the other constraints). In our earlier example, the sum of the capacities into the terminal node is 7, so the value of the flow on that network cannot be larger than that.

Example: Convert the Max Flow Problem to an LP

You might notice that the definition of a flow actually sets up the linear program. Let's keep to our previous example and set up the max-flow LP.



We want to maximize the sum of the flows out of the source:

$$f_{s1} + f_{s2} + f_{s3}$$

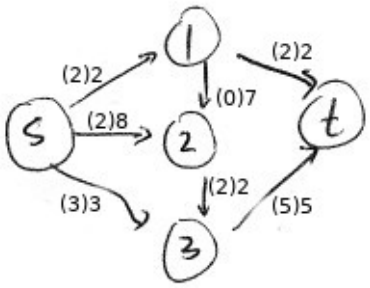
such that (i) the flows are non-negative, do not exceed the capacities, and (ii) flow is conserved at each node, and the sum of the flows out of the source is equal to the sum into the terminal. Here are the constraint equations that provide these:

$$\begin{array}{ll} 0 \le f_{s1} & \le 2 \\ 0 \le f_{s2} & \le 8 \\ 0 \le f_{s3} & \le 3 \\ 0 \le f_{12} & \le 7 \\ 0 \le f_{23} & \le 2 \\ 0 \le f_{1t} & \le 2 \\ 0 \le f_{3t} & \le 5 \end{array}$$

$$\begin{array}{rll} f_{s1} & = f_{1t} + f_{12} & \text{Node 1} \\ f_{s2} + f_{12} & = f_{23} & \text{Node 2} \\ f_{s3} + f_{23} & = f_{3t} & \text{Node 3} \\ f_{s1} + f_{s2} + f_{s3} & = f_{1t} + f_{3t} & \text{Nodes } s, t \end{array}$$

Before we look at the **Ford-Fulkerson** algorithm, let's see if we can reason out a good flow from the example. First, let's just find a path from s to t- For example, $s-1-t$ is a reasonable path, and both edges have a capacity of 2, so we can max that out and make $f_{s1} = 2$ and $f_{1t} = 2$. Any other paths? We can go $s-3-t$, and we see capacities of 3 and 5. Look for the smallest capacity along the path (it is called the bottleneck capacity), and that will be flow for the path. In this case, $f_{s3} = 3 = f_{3t}$, and so far the value of the flow is 5.

Some vocabulary: The paths we're finding are called *augmenting paths* because we're using them to augment the flow. Are there any augmenting paths remaining? We still have $s-2-3-t$ left, and it looks like the bottleneck capacity is 2. If we check the sum of the flow into the terminal node, we see that we have made the maximum capacity of 7 units.

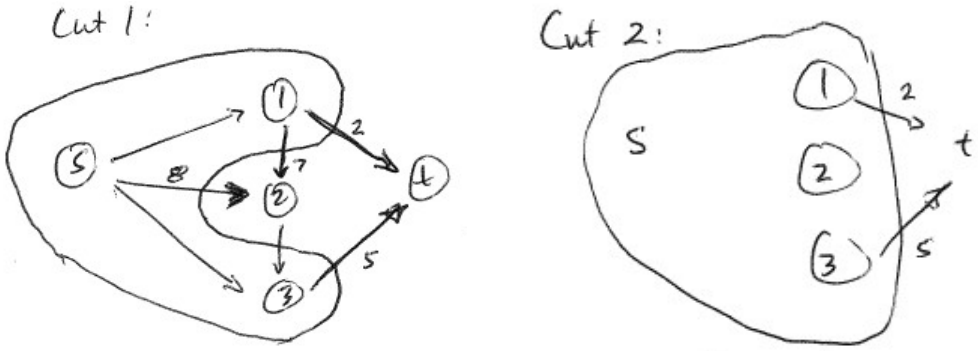


Cuts

Definition: Given a graph G, a **cut**(A, B) is a partition of the vertices in V into (distinct) sets $s \in A, t \in B$.

Definition: The **capacity of** cut(A, B) is the sum of the capacities of the edges that run FROM A TO B. For notation, we might say that:

$$\text{cap}(A, B) = \sum_{e \text{ out of } A} c_e$$



In the example labeled "Cut 1", set $A = \{s, 1, 3\}$ and set $B = \{2, t\}$. The capacity of the cut is the sum of the capacities of the edges leading out of A into B. In this case, we have edges $(1, t), (1, 2), (3, t)$ and $(s, 2)$. Summing these capacities:

$$\text{cap}(A, B) = 2 + 7 + 5 + 8 = 22$$

Similarly, in the example labeled "Cut 2", set $A = \{s, 1, 2, 3\}$ and $B = \{t\}$. The capacity of this cut is then the sum of the capacities of the edges going to t, which is $2 + 5 = 7$.

Cuts and Flows

Given any flow f and any cut (A, B), we get what has been called **the Flow Value Lemma**.

$$\sum_{e \text{ out of } A} f_e - \sum_{e \text{ in to } A} f_e = \text{val}(f)$$

Using the flow that we found, let's compute some of these quantities using Cut 1, then Cut 2.

- Cut 1:
  - The capacity of the cut is 22. That is, $\text{cap}(A, B) = 22$.
  - The sum of the flows out of A: The 4 edges were: $(1, t), (1, 2), (3, t), (s, 2)$, with $2 + 0 + 3 + 2 = 7$.
  - The sums of the flows into A: $(2, 3)$ is the only edge going in, so that's 2.
  - Therefore, the net flow across the cut is $val(f) = 7 - 2 = 5$.
- For cut 2, you might verify that the net flow across the cut is 7.

Using the flow value lemma, it's easy to see one relationship between the value of the flow and the capacity of a cut. Once again, given an arbitrary flow f and cut with partitions (A, B), we have:

$$\begin{aligned} \text{val}(f) &= \sum_{e \text{ out of } A} f_e - \sum_{e \text{ in to } A} f_e \\ &\leq \sum_{e \text{ out of } A} f_e \\ &\leq \sum_{e \text{ out of } A} c_e = \text{cap}(A, B) \end{aligned}$$

The Max-Flow, Min-Cut Theorem

The maximum value of an $s-t$ flow is equal to the minimum capacity over all $s-t$ cuts. Basically, if we can find a cut whose capacity is equal to the current flow, we've found the maximum.

To whet your curiousity a bit, we'll note that it is possible to construct the max-flow problem as a primal linear program, and the min-cut problem as its dual- We'll leave that for later perhaps.

In our previous example, we see that Cut 2 gave a capacity of 7, and that was also the value of the flow. In that case, the theorem says that we have a maximum flow.

Graphs and Residual Graphs

For our graph G, there is a very useful associated graph G_f called the **residual graph**. Our textbook is confusing on this point, so let's see if we can formalize this concept a bit. This is all followed by several examples, followed by the **Ford-Fulkerson Algorithm**.

How to build the Residual Graph

Given graph G with a given flow, the following steps will create the residual graph G_f. We'll see that the residual graph helps us to determine if we need to continue to look for more augmenting paths or not (which is very helpful). To build the residual graph,

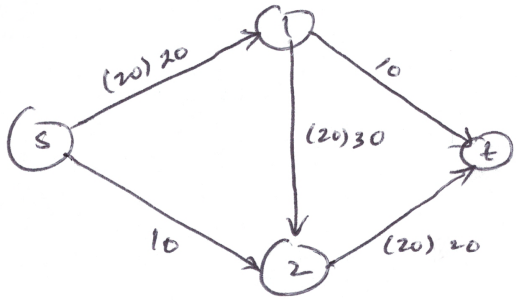
- For each edge $e = (u, v)$ and $f_e > 0$ on graph G, create a backwards edge going from v to u on graph G_f with a "residual capacity" along the backwards edge f_e.
- For each edge $e = (u, v)$ and $f_e < c_e$ on graph G, create a forward edge from u to v on graph G_f with residual capacity $c_e - f_e$.

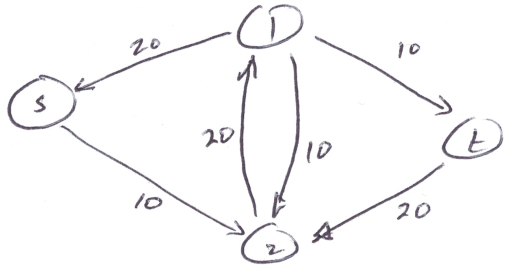
That's it. Let's try it out- Below I've drawn a graph G with a flow. Let's compute the residual graph G_f- We'll go through each edge of the graph G on the top, and the bottom graph will be the residual graph G_f.

- $(s, 1)$. Create backward edge $(1, s)$, label with 20. Since $f_e = c_e = 20$, no forward edge left.
- $(s, 2)$. Flow $f_e = 0$, so do nothing (keep the edge and capacity).
- $(1, 2)$. Create backward edge $(2, 1)$ label with 20. Now,

$$c_e - f_e = 30 - 20 = 10$$

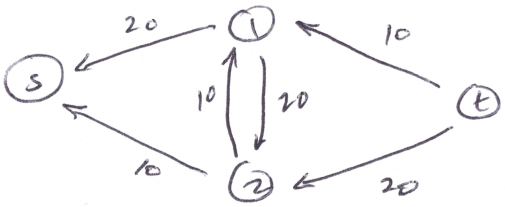
  so forward edge has label 10.
- $(1, t)$. Flow $f_e = 0$, so keep the forward edge and the capacity 10.
- $(2, t)$. Create backward edge $(t, 2)$, label with 20. No forward edge.





You might recognize the utility of the residual graph right away- Using the residual graph, it is easy to find augmenting paths from s to t by following the arrows. The "backward edge" may be confusing at first-

If you go on a backward edge, you can think of returning the flow that was already used. For example, I see a nice path $s-2-1-t$ that has a capacity of 10.



Can you draw the new residual graph? Here it is to the left, and using the residual graph, it is very clear there are no longer any paths from the source to the terminal, so we have found the maximum flow (can you write down the flow from the residual?), which is 30. I think we're ready for the algorithm now!

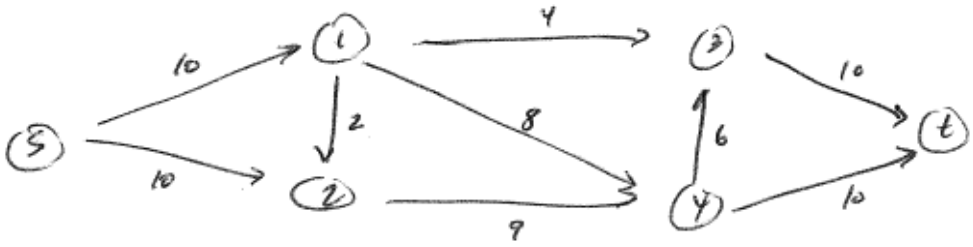
The Ford-Fulkerson Algorithm

First we'll state the algorithm which is really how we've been finding the maximum flow in our two examples.

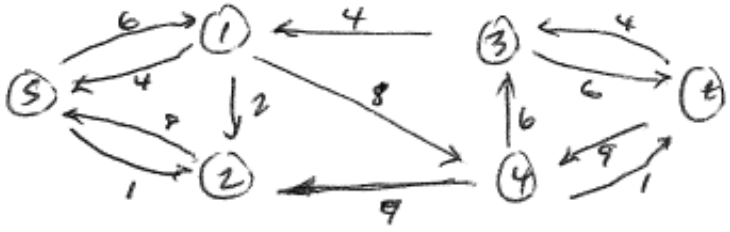
1. Find an $s-t$ directed path. Find the bottleneck b for the path, and add this value to the flow f_e to each edge in the path (for graph G).
   - No path? Stop. The maximum flow has been found.
2. Build the residual graph G_f for the flow following the steps below (these are the same as before):
   - For each edge $e=(u,v)$ and $f_e > 0$, create a backwards edge going from v to u with a "residual capacity" along the backwards edge f_e.
   - For each edge $e=(u,v)$ and $f_e < c_e$, create a forward edge from u to v with residual capacity $c_e - f_e$.
3. Repeat.

As a side remark, the graphs we've been working with have been simple enough that we don't necessarily need the residual graph, but this graph is used for other things, so its a good idea to become accustomed to it (and the notation).

Example 2



Here is a graph- Let's go through the Ford-Fulkerson algorithm. To begin, we'll push 4 through the top path, $s-1-3-t$ and we might as well take care of the bottom arc as well and push 4 through $s-2-4-t$. The residual graph is given below:

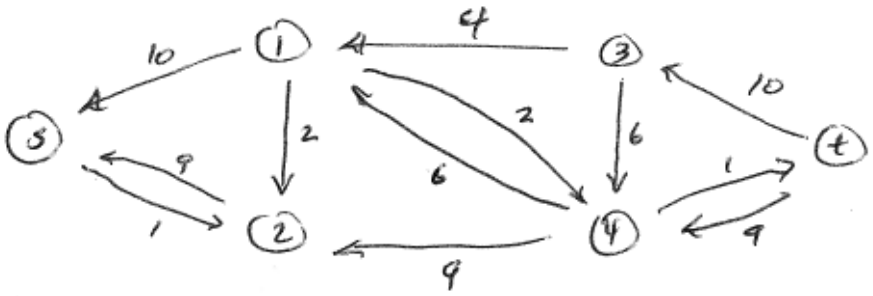


How about path

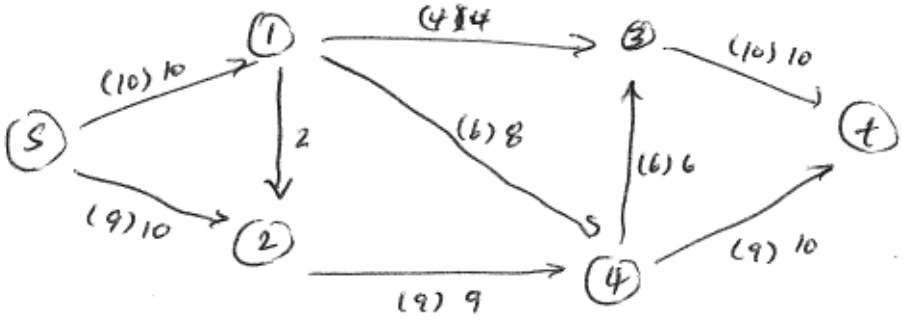
$s-1-4-3-t$

The bottleneck value is 6.

The next residual graph is given below.



From the source, we are forced to go to node 2. But at node 2, there is nowhere else to go. We're done. Here is the original graph G with the flow values listed.



Let's take a random cut, say $A = \{s, 2, 3\}$ and $B = \{1, 4, t\}$. To find the capacity of the cut, consider all edges going from a node of A to a node of B. In this case, the edges are:

$$(s, 1), (2, 4), (3, t) \quad \Rightarrow \quad \text{cap}(A, B) = 10 + 9 + 10 = 29$$

We can also consider $A = \{s, 1, 2\}$ and $B = \{3, 4, t\}$. In this case, the capacity of the cut is $4 + 8 + 9 = 21$. We might also take $A = \{s\}$. In that case, the capacity of the cut is $10 + 10 = 20$. Is it possible to construct a cut that is equal to the maximum flow? We'll state the algorithm without proof.

Finding the Minimum Cut

Use the residual graph:

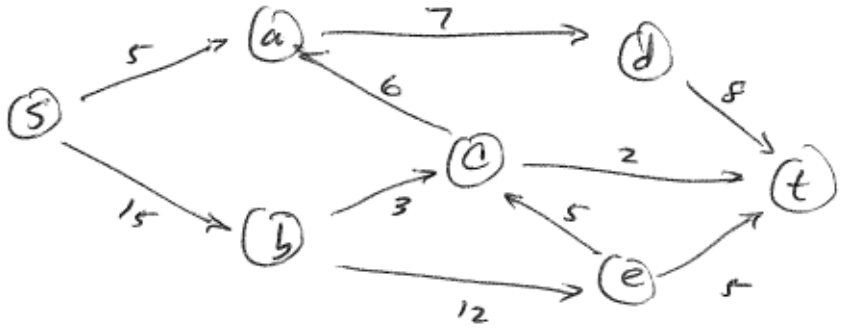
- Find all nodes that are reachable from the source.
- The cut consists of the edges from nodes reachable from the source to the nodes that are not reachable from the source.

In our example above, that means $A = \{s, 2\}$ and $B = \{1, 3, 4, t\}$. The capacity of this cut is using edges $(s, 1)$ and $(2, 4)$ only, so it is $10 + 9 = 19$, and that is the value of the flow. Good job!

Example:

Given below is a network with capacities. Use the Ford-Fulkerson algorithm to determine the maximum flow, and then determine a cut that satisfies the Max-Flow, Min-Cut Theorem. (You should practice using the residual graphs, which will be given below in the solution).

Here's the graph G:



Don't look below until you've tried it!

A flow for graph G and the residual graph G_f are shown below. The max flow was 14, and a cut was $A = \{s, b\}$.