Graphs, Flows, and the Ford-Fulkerson Algorithm

Vince Vatter

August 12, 2004

What is a graph?

One way to define a graph is as "a pair $G = (V, E)$ of sets satisfying $E \subseteq [V]^2$." I rather like telling people that graphs are "a tangled mess of dots and lines." Somewhere in between lies the following.

Definition 1. *A (simple) graph is an object with vertices and edges such that every edge connects two different vertices, and no two edges connect the same two vertices.*

When we draw a graph, we draw the vertices as dots, and the edges as lines or curves connecting the dots. Note that I didn't tell you anything about where to draw the dots and edges, so there are many different ways to draw the same graph.

Graphs are good for modeling many different things. For example, you can use graphs to model the Web by thinking of every website as a vertex and edges between vertices as hyperlinks. Or you can model acquaintances by thinking of every person as a vertex and adding an edge between two people if they know each other. The Law of Six Degrees of Separation, most famously observed by the Harvard experimental psychologist Stanley Milgram, concerns this later graph: it says that every two people can be connected by a sequence of at most six other people, so that each person knows the next. For example, a route from Bill to Amy might be: Bill knows Jason who knows Alex who knows Sarah

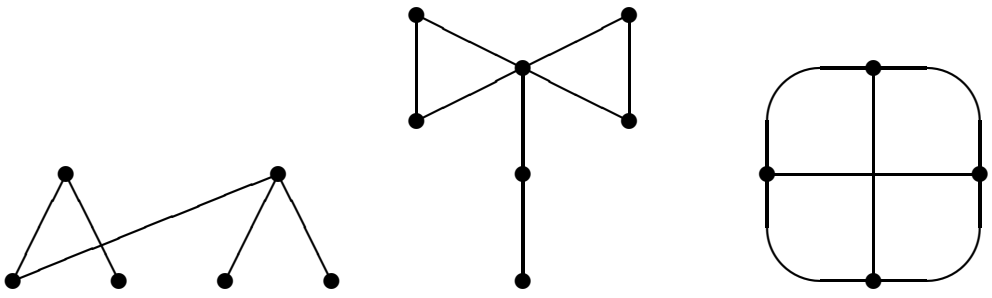


Figure 1: Some graphs

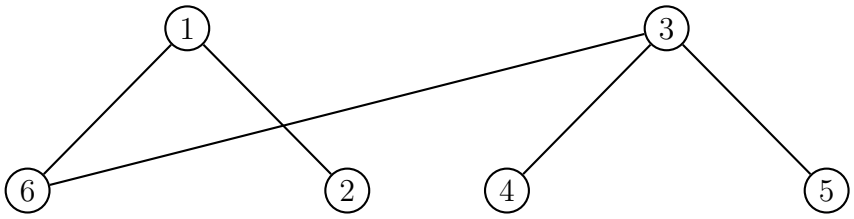


Figure 2: A labeled version of the first graph from Figure 1

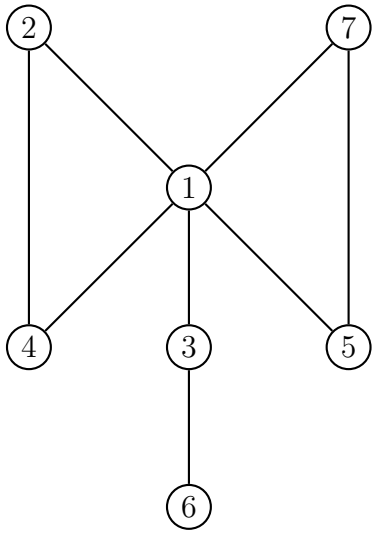


Figure 3: A labeled version of the second graph from Figure 1

who knows Laura who knows Amy. This route went through four intermediaries. You can also model roads, cell phone networks, sewer systems, etc.

It's tough to talk into any more detail about the graphs in Figure 1, because the vertices don't have names. So, for the rest of this document, all vertices will have labels attached to them. To keep things simple, I'm also going to label the vertices $1, 2, 3, \ldots$. Figure 2 shows a labeled version of the first graph from Figure 1 and Figure 3 shows a labeled version of the second graph.

Now that we can talk about individual vertices, we can state a couple definitions.

Definition 2. *The degree of a vertex is the number of edges coming to or going out of it.*

In the graph in Figure 2, vertices 1, 3, and 6 have degree 2 while vertices 2, 4, and 5 have degree 1.

Definition 3. *A path from u to v is a way to get from u to v following edges of the graph, without visiting the same vertex twice.*

In the graph in Figure 2, a path from 1 to 4 is given by the following sequence: 1, 6, 3, 4.

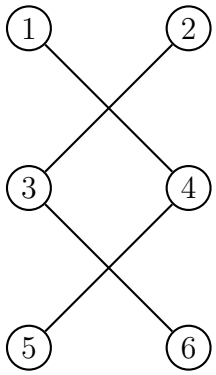


Figure 4: A disconnected graph

Sometimes graphs have very special paths that are called *Hamiltonian*. These are paths that go through every vertex of the graph exactly once. There are no good characterizations known about what graphs have Hamiltonian paths. One of the earliest advances was the following theorem of Dirac from 1952.

Theorem 4. *Suppose a graph has n vertices. If each vertex of the graph has degree at least $n/2$, then the graph has a Hamiltonian path.*

(You might have figured out by now that some of the things I mention won't be on the final. Dirac's Theorem is one of them.)

Definition 5. *A graph is said to be connected if there is a path between every two of its vertices.*

All the graphs we have seen so far have been connected, so Figure shows one that isn't (for example, in this graph you can't get from 1 to 3).

It is often handy to represent graphs with matrices instead of pictures. The matrix we use is called the *incidence matrix* (or *adjacency matrix*). It is given my $M = [m_{i,j}]$ where

$$m_{i,j} = \begin{cases} 1 & \text{if there is an edge from vertex } i \text{ to vertex } j, \\ 0 & \text{otherwise.} \end{cases}$$

For example, the incidence matrix of the graph from Figure 3 is

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

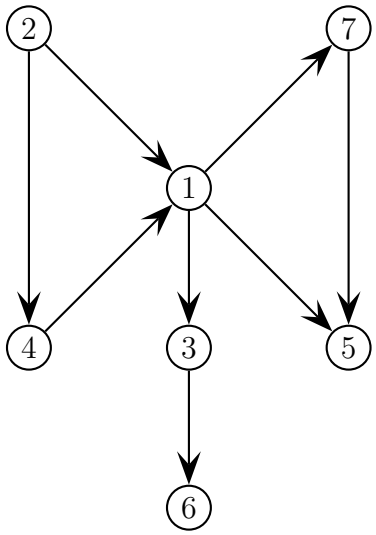


Figure 5: A directed version of the graph from Figure 3

This matrix is always symmetric. That is, M^T is always equal to M for an incidence matrix. This is because whenever there is an edge from vertex i to vertex j, there is also an edge (the same one!) going in the other direction.

So far we have just talked about simple graphs. We can direct the edges of a simple graph to get a *directed graph*, or *digraph* for short. Usually we refer to directed edges as *arcs*. Figure 5 shows a directed version of the graph from Figure 3.

The incidence matrices of digraphs are defined essentially the same as the incidence matrices of graphs, except that now the orientation of the edges matters, so we have

$$m_{i,j} = \begin{cases} 1 & \text{if there is an arc from vertex } i \text{ to vertex } j, \\ 0 & \text{otherwise.} \end{cases}$$

The incidence matrix of the graph in Figure 5 is

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Notice that this matrix is *not* symmetric, since now there can be an arc from vertex i to vertex j without there being an arc in the other direction, from j to i.

Networks and Flows

Now we are going to think of our edges as pipes of varying sizes and our vertices as places where the pipes run together and the material we are piping can change direction.

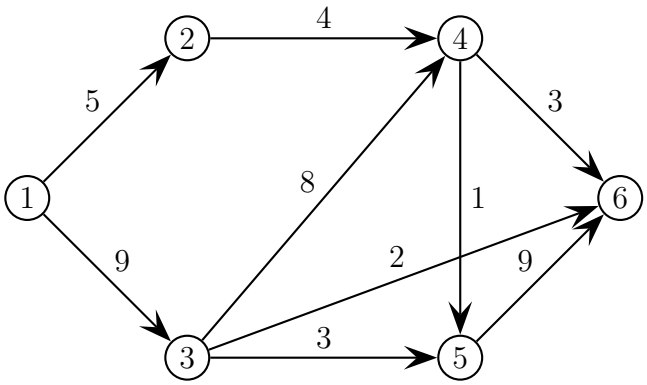


Figure 6: A network

Since pipes either flow one way or the other, we will be working with digraphs. To model the fact that the pipes can be different sizes, we will assign a *capacity* to each arc, which we will write next to the arc. We also have a *source*, which is where the things are being pumped from, and a *sink*, which is where the things are being pumped to. I will always label the source 1 and give the greatest label to the sink. What we get when we do this is called a *network*. Figure 6 shows a network.

Let us give the various arc capacities names before going on:

$c_{i,j}$ = the capacity of the arc from vertex i to vertex j (0 if there is no such arc).

We could put these into a matrix if we wanted. Below is the capacity matrix for the graph in Figure 6.

$$\begin{bmatrix} 0 & 5 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 8 & 3 & 2 \\ 0 & 0 & 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We want to figure out how to pump the maximum amount possible from the source to the sink. In doing so we will construct a *flow*. A flow will tell each pipe how much stuff to, well, pipe. So a flow will be specified by a set of numbers $x_{i,j}$ which tell how much stuff the pipe from i to j will pipe. (Like the $c_{i,j}$s, we will have $x_{i,j} = 0$ if there is no arc from i to j.)

Flows have to satisfy several restrictions. First, a flow can't have a pipe piping more than it can pipe, so we will insist that

$$x_{i,j} \leq c_{i,j} \text{ for all valid } i \text{ and } j. \tag{1}$$

But we're getting ahead of ourselves. The first constraint we should have noticed is that pipes can't pipe negative amounts:

$$x_{i,j} \geq 0 \text{ for all valid } i \text{ and } j. \tag{2}$$

The third and final constraint is known as Kirchkoff's Law. It says that the amount flowing into a vertex must equal the amount flowing out of the vertex (unless that vertex is the source or the sink). This comes from the fact that our vertices are just intersections, and don't have any storage capacity of their own. The amount flowing out of vertex i is given by

$$\sum_{k=1}^{n} x_{i,k},$$

where n is the number of vertices in our network, and the amount flowing into vertex i is given by

$$\sum_{k=1}^{n} x_{k,i}.$$

So in mathematical terms Kirchkoff's Law says

$$\sum_{k=1}^{n} x_{i,k} \;=\; \sum_{k=1}^{n} x_{k,i} \text{ for all valid } i \text{ except for the source and the sink.} \tag{3}$$

That's it for the constraints. What about the objective function? We want to maximize the amount of stuff that gets pumped from the source to the sink. Because of Kirchkoff's Law, everything that leaves the source must eventually get to the sink, so the amount of stuff that gets pumped from the source to the sink is

$$\sum_{k=1}^{n} x_{1,k}.$$

(Here we are using our convention that the source is vertex 1.) By Kirchkoff's Law again, this is the same as the amount of stuff that enters the sink:

$$\sum_{k=1}^{n} x_{k,n}.$$

(And here we are using our convention that the sink is vertex n.)

Putting this, (1), (2), and (3) together, we can write the problem as a linear programming problem.

$$\begin{array}{l}
\text{Maximize } z = \sum_{k=1}^{n} x_{1,k} \\
\text{subject to} \\
\begin{array}{rcl}
\sum_{k=1}^{n} x_{i,k} & = & \sum_{k=1}^{n} x_{k,i} \text{ for all } i \text{ between 2 and } n-1 \\
x_{i,j} & \leq & c_{i,j} \text{ for all } i \text{ and } j \text{ between 1 and } n \\
x_{i,j} & \geq & 0 \text{ for all } i \text{ and } j \text{ between 1 and } n
\end{array}
\end{array}$$

Since we usually like to get all our variables on the left-hand side, we can rewrite this as

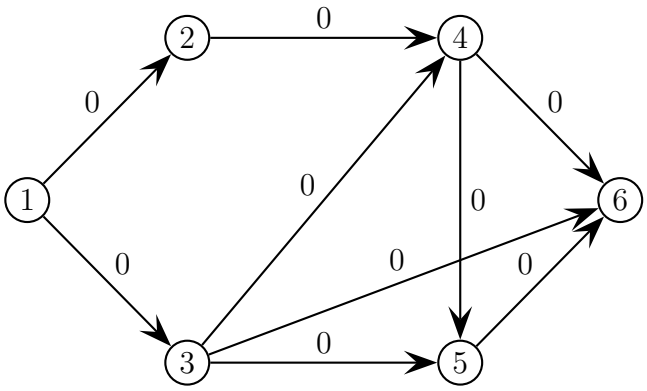
$$\begin{array}{l}
\text{Maximize } z = \sum_{k=1}^{n} x_{1,k} \\
\text{subject to} \\
\begin{array}{rcl}
\sum_{k=1}^{n} x_{i,k} - \sum_{k=1}^{n} x_{k,i} & = & 0 \text{ for all } i \text{ between 2 and } n-1 \\
x_{i,j} & \leq & c_{i,j} \text{ for all } i \text{ and } j \text{ between 1 and } n \\
x_{i,j} & \geq & 0 \text{ for all } i \text{ and } j \text{ between 1 and } n
\end{array}
\end{array}$$

Now we just have to solve this! But, there are n^2 variables and $2n^2 + n$ constraints, so the Simplex Method might not be the way to go. Instead, we will use the...

Ford-Fulkerson Algorithm

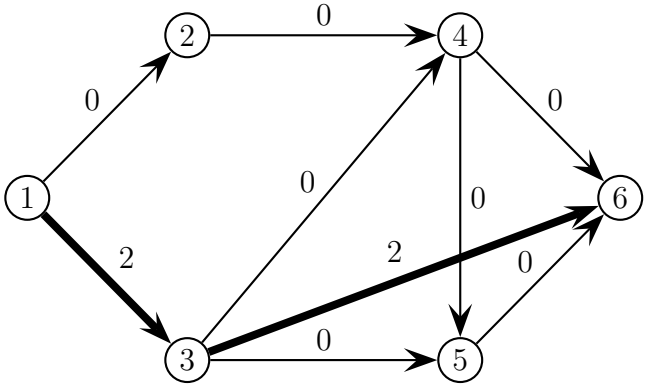
The Ford-Fulkerson Algorithm is really quite natural. We start with no flow at all, that is, with every $x_{i,j}$ set equal to 0. Then we find what is called an *augmenting path* from the source to the sink. This is, as it says, a path from the source to the sink, that has excess capacity. We then figure out how much more we could pipe down that path and add this to the flow we are building.

We will apply the algorithm to the network from Figure 6. We start with the flow set equal to 0 everywhere:



Now we need to find an augmenting path. There are millions of them (well, maybe not exactly millions, I count six). Let take the path $1 \to 3 \to 6$ to start with. Now we have to figure out what how much more stuff we can pipe down this route. The pipe from $1 \to 3$ has capacity 9 ($c_{1,3} = 9$), and is currently piping 0 ($x_{1,3} = 0$ right now), so it could pipe 9 more. The pipe from $3 \to 6$ has capacity 2 and is currently piping 0, so it could pipe 2 more. We can only pipe the minimum of these numbers, so we will only be able to send 2 units of stuff down this path.

Then we update the flow:

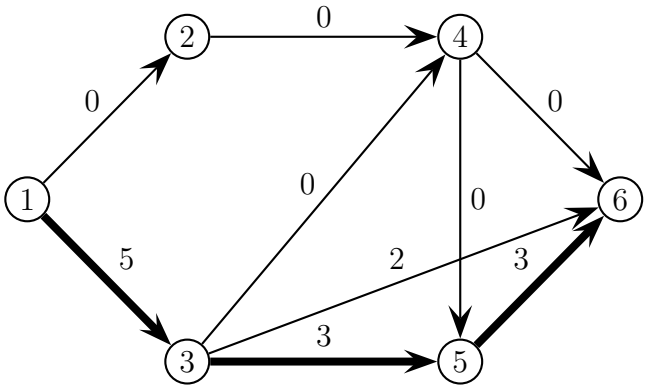


We are now ready to repeat. Let's take the augmenting path $1 \to 3 \to 5 \to 6$. For

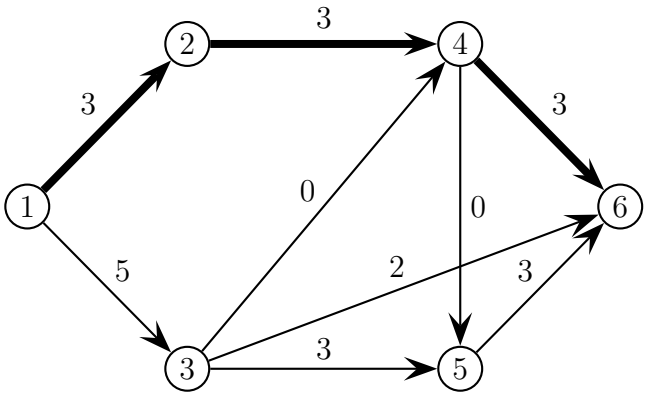
each arc we check how much spare capacity it has:

arc	total capacity		current load		excess capacity
$1 \to 3$	9	$-$	2	$=$	7
$3 \to 5$	3	$-$	0	$=$	3
$5 \to 6$	9	$-$	0	$=$	9

The smallest excess capacity is 3, so that's what we'll add to the flow, which is shown below:



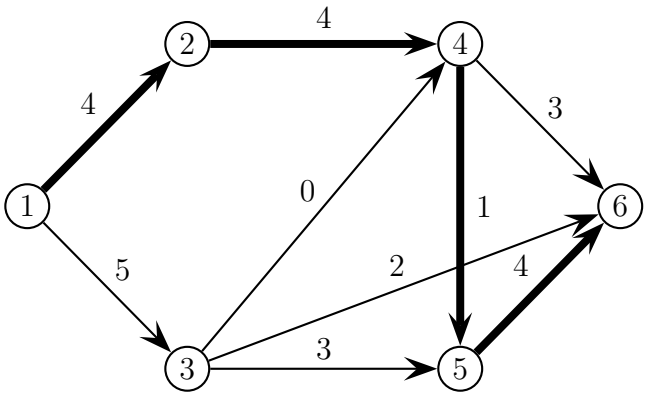
Time for another augmenting path. This time let's take $1 \to 2 \to 4 \to 6$. The excess capacities of these arcs are 5, 4, and 3, respectively, so the most we can send down this way is 3. The updated flow is:



It's getting a little harder to spot augmenting paths now, but there's at least one more: $1 \to 2 \to 4 \to 5 \to 6$. We have

arc	total capacity		current load		excess capacity
$1 \to 2$	5	$-$	3	$=$	2
$2 \to 4$	4	$-$	3	$=$	1
$4 \to 5$	1	$-$	0	$=$	1
$5 \to 6$	9	$-$	3	$=$	6

This shows that we can send one unit of stuff down this path. The updated flow is



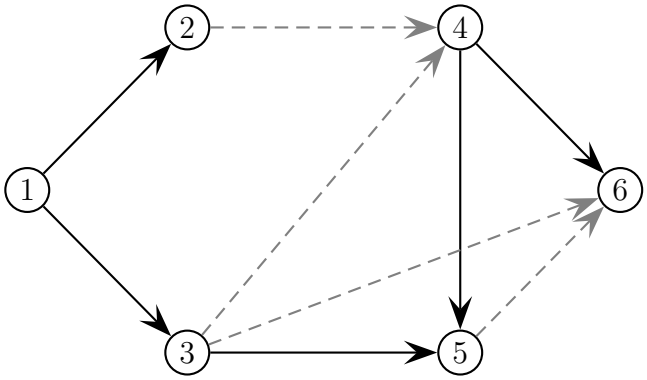
Now we're managing to get 9 units from the source to the sink. But, is the best we can do?

How you know when you're done

Trying to move stuff from the source to the sink depends very much on the paths from the source to the sink. It is depends on the ways in which those paths can be cut off.

Definition 6. *A cut in a network (or just a digraph) is a set of arcs such that if they are removed, there is not path from the source to the sink.*

For example, the dashed arcs in our graph below represent a cut since removing them leaves no paths from the source to the sink. (Note that you are *not* allowed to go $1 \to 3 \to 5 \to 4 \to 6$, because then you are traversing the $4 \to 6$ arc in the wrong direction.)

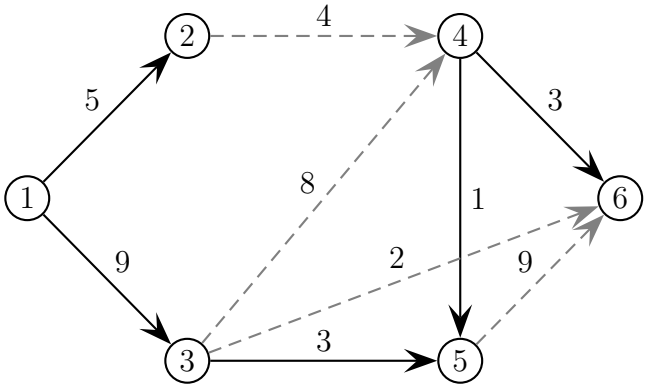


We will need to measure how much stuff could flow through all the arcs in a cut:

Definition 7. *The capacity of a cut is defined to be the sum of the capacities of every arc in the cut.*

So to figure out the capacity of a cut, we want to look at the original capacity graph, *not* the flow graph we may or may not have just made. The example cut above, shown

on the capacity graph below, has capacity $4 + 8 + 2 + 9 = 23$.



Now we are ready for the big theorem, which will tell us when we are done in the Ford-Fulkerson Algorithm.

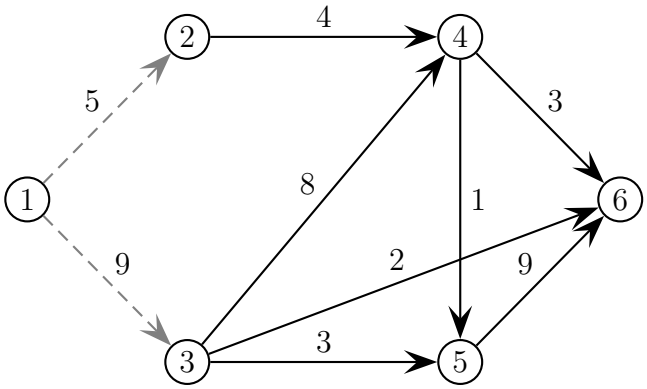
Theorem 8 (Max-Flow Min-Cut Theorem). *In every network, the maximum flow equals the minimum capacity of a cut.*

This theorem was proved in 1956 independently by Ford and Fulkerson and by Feinstein and Shannon. The proof by Ford and Fulkerson uses their algorithm is a very straight-forward way. The theorem can also be proved by apply the Duality Theorem from Linear Programming.

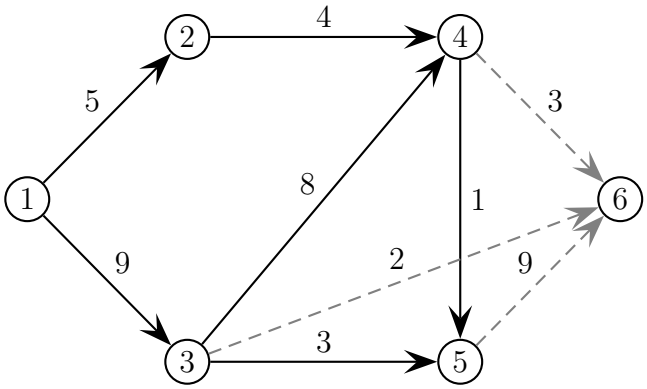
The only problem with the Max-Flow Min-Cut Theorem is that the two quantities it says are equal are both hard to get a handle on. But, it is nice to know that if you can find a flow and a cut with the same value, you are done. That is, you have found the best flow possible for that network.

What about our example? Did we find the best flow possible? Well, we were able to pump 9 units from the source to the sink. However, the cut we found had capacity 23, so there must either be a better flow or a smaller cut. Let's keep playing around with the cuts for a while, and see if we can make a better one.

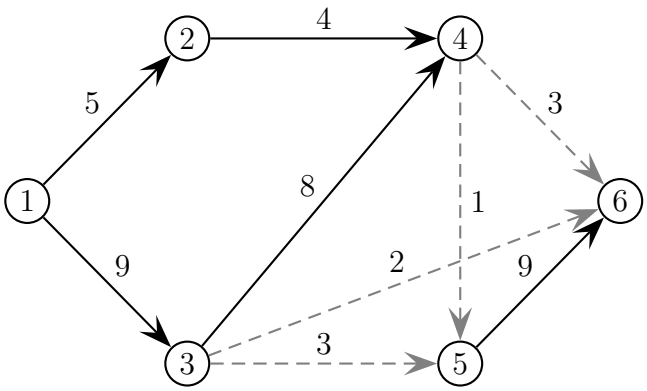
Here's a cut with capacity $5 + 9 = 14$:



And here's another cut with capacity $3 + 2 + 9 = 14$:



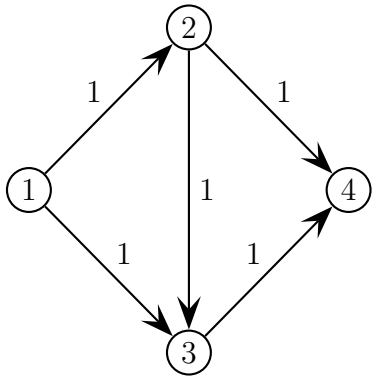
Finally, here's a tricky cut with capacity $2 + 1 + 3 + 3 = 9$:



So, we found a flow that transported 9 units and a cut that had capacity 9. By the Max-Flow Min-Cut Theorem we have found the best flow possible, and we can stop.

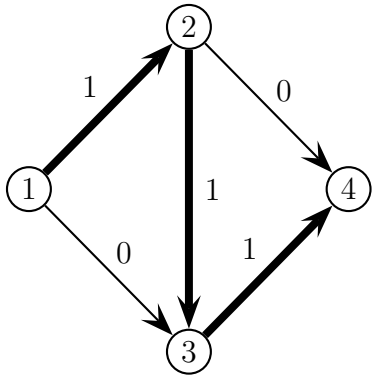
Wrinkles

Let's suppose now that the capacity graph is



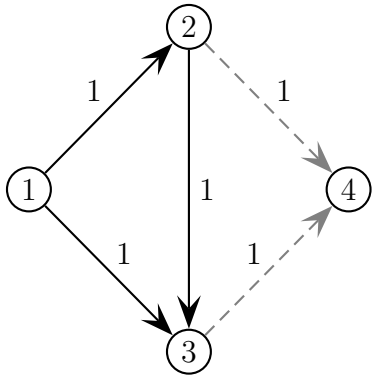
And let's suppose that we start the Ford-Fulkerson Algorithm by adding the augmenting path $1 \to 2 \to 3 \to 4$, which we can ship 1 unit down. Then we get the following

flow



Stop for a moment and answer these questions: Is this the maximum flow possible? If not, can you spot an augmenting path that would make it maximum?

To answer the first question you should think of the Max-Flow Min-Cut Theorem. If we could find a cut of capacity 1, then this would indeed be the maximum possible flow. Sadly, there is no cut of capacity 1. Then minimum cut we can make has capacity 2:



The Max-Flow Min-Cut Theorem then tells us that there must be a better flow. Of course, in this example, it's easy to find a better flow, but don't think about that. The concern right now is: how can we improve the flow we already have?

The answer is that we can add an augmenting path that goes the wrong way down a pipe! So, we can add the path that goes $1 \to 3 \to 2 \to 4$. But how much can we take through this route?

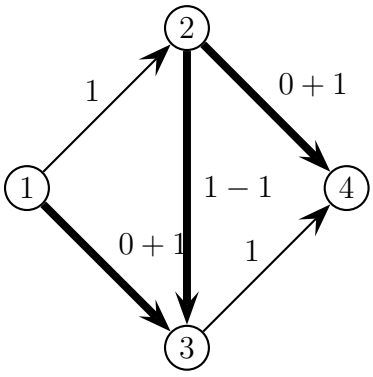
We compute excess capacity like normal for the arcs that we are going the right way on:

arc	total capacity		current load		excess capacity
$1 \to 3$	1	$-$	0	$=$	1
$2 \to 4$	1	$-$	0	$=$	1

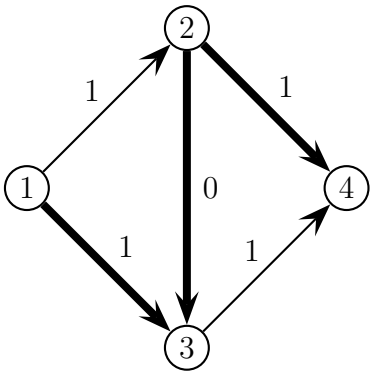
For the $3 \to 2$ arc that we are going the wrong way down, the excess capacity is however much is currently flowing down it in the right direction. So, since we are currently sending 1 unit down the $2 \to 3$ arc in the right direction (that is, from 2 to 3), we can send 1 unit up this arc in the wrong direction.

Then the other change is that when we update the flow, we add the new amount to the arcs that we went down in the right direction like before, but we *subtract* the new amount from the arcs that we went down the wrong way.

In this case our new flow is



or, after simplifying:



This has now increased our total flow to 2, and since we already found a cut of capacity 2, the Max-Flow Min-Cut Theorem tells us that we are done.

There are two ways to think about why we are allowed to send stuff the wrong way down a pipe. First, we might just observe that in adding the augmenting path in the manner in which we just did, we did break any of the rules for flows - that is, our new flow still satisfied (1), (2), and (3).

The other way to think about it is that instead of "sending stuff the wrong way," we were really just dropping off stuff on one side of the arc and then telling the previous path to leave it's stuff on the other side of the arc. In our example, in our initial flow we were pumping 1 unit from 1 to 2 to 3 to 4. When we went to add the new path, what we said to the old path was "we'll bring one unit from 1 to 3, so you can take that to 4, and hey, would you mind leaving a unit at 2 so that we can pick it up and take it to 4?"