CS 624
Lecture 14: Maximum Flow

1 The problem

1.1 The kind of graphs we are dealing with

$G = (V, E)$ is a directed graph with two distinguished vertices, s (the "source") and t (the "sink")[1] We also assume that

1. There is at least one path from s to t. In fact, even more is true: for each node u in the graph, there is at least one path from s through u to t.
2. There are no "antiparallel" edges. That is, if $e = (u, v)$ is an edge from vertex u to vertex v, then there is no edge from v to u. (Of course there could be a *path* from v to u. In that case there would be a cycle in the graph, but that is OK.)

Now actually this last item ("no antiparallel edges") is not a big deal. In fact, if we have a graph with edges (u, v) and (v, u), we can just introduce a new vertex w and replace the edge (v, u) by the two edges (v, w) and (w, u). And for everything that we are doing here, that won't change what we are looking for in any respect.

Definition *A* flow network *is a directed graph $G = (V, E)$ with two distinguished vertices s and t such that*

- *for each vertex u there is at least one path from s through u to t, and*
- *there are no antiparallel edges in the graph.*

1.2 Flows

So far, we have a kind of directed graph. We want to look at *flows* in this graph from s to t. So we have to say what this means:

Preliminary Definition *A flow in the directed graph G as above is an assignment of a non-negative real number to each edge of the graph.*

[1] The letter t probably stands for "target", although "sink" is what is generally used to refer to that vertex. I may use "target", however.

This definition will do for the moment, but it will be replaced below by a more sophisticated one which will be more useful.

We denote the assignment by $f : E \to \{r \in \mathbf{R} : r \geq 0\}$. Thus, for each edge e, $f(e)$ is a non-negative real number. If $e = (u, v)$, then we may (by an abuse of notation) also write $f(u, v)$ for $f(e)$—they will mean the same thing.

We think of $f(e)$ as the amount of some substance that is passing over edge e per unit time. So for instance, if the graph is a representation of electrical lines, then $f(e)$ would be the total energy passing through e in some standard unit time. Presumably in that case we would measure $f(e)$ in watts.

We might instead assume that the graph represented a transportation network—roads, say, or railroad lines. And then for each edge e, $f(e)$ would represent the amount of something that was being transported over e in some unit of time[2]

1.2.1 A conservation property, and two ways to express it

Now the idea is that we are transporting some substance from s to t. So it must be that the function f satisfies the following constraint:

Constraint At every node u in the graph other than nodes s and t, the flow into u must exactly equal the flow out of u.

This constraint is really a sort of conservation property—it says that the amount being transported is conserved at each vertex.

We need to be able to express this constraint mathematically. To do this, it is most convenient to extend the function f to all pairs of nodes (u, v)—that is, even pairs for which (u, v) is not an edge in the graph. There are two conventions that can be used to do this:

First convention: extend f so it remains non-negative, We extend the function f so that if (u, v) is not an edge, then we simply set $f(u, v) = 0$. With that notation, we can write our constraint as follows: for each node u different from s and t,

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$$

This is in fact the way our text writes this constraint.

Second convention: extend f so that it can also be negative. We do this as follows: if (u, v) is a (directed) edge in the graph, then we know already that (v, u) is not. We define $f(v, u) = -f(u, v)$. And for all other pairs (x, y) such that neither (x, y) nor (y, x) is an edge, we define $f(x, y) = f(y, x) = 0$.

The idea here is that we can think of a flow along (u, v) as a "negative" flow along the reversed edge (v, u).

[2]This was, in fact, the original motivation for this problem.

With this notation, we can write our constraint more simply: for each node u different from s and t,

$$\sum_{v \in V} f(u, v) = 0$$

It's important to realize that both of these conventions are completely equivalent. The first convention has the advantage that in the beginning, anyway, it can seem somewhat easier to understand what is going on. The second one has the advantage that the mathematics turns out to be much simpler.

We have to pick one. Both are widely used. We are going to use the second one.

1.2.2 The real definition of a flow

So now we can give the precise definition of a flow:

Definition *A flow f on the graph G with source and target vertices s and t is any function mapping pairs of vertices of G to $\mathbf{R}$ such that it satisfies the following three conditions:*

Condition 1: f lives on edges. *If u and v are not joined by an edge (in either direction), then $f(u, v) = 0$.*

Condition 2: f is skew-symmetric. *For all vertices u and v,*

$$f(u, v) = -f(v, u)$$

Condition 3: f satisfies a conservation property. *For all vertices v other than s and t,*

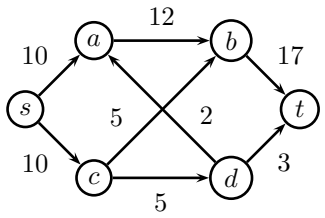
$$\sum_{u \in V} f(u, v) = 0$$

Example Figure 1 shows two flow networks with equivalent flows. By "equivalent", I just mean that in spite of the small difference in notation, actually the same amount of material is passing in the same direction between each corresponding pair of vertices.

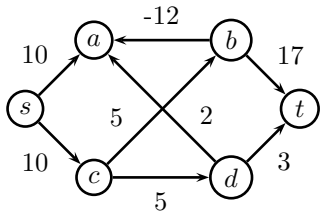
1.1 Exercise *Show that both h and k in Figure 1 are in fact flows. This means, of course, that you have to show that they are flows according to the definition we have just given.*

1.2.3 Adding flows

Flows can be added:



the flow h



the flow k ◇

Figure 1: Two equivalent flows.

Definition *If f and g are two flows, we define their sum $f + g$ in the obvious way:*

$$(f + g)(u, v) = f(u, v) + g(u, v)$$

Example Figure 2 shows a flow network three times. The version on the top has a flow f. (This is the same as the flow h in Figure 1.) The version in the middle has a flow g. The version on the bottom has the flow which is the sum $f + g$.

1.2 Exercise *Prove that f, g, and $f + g$ in Figure 2 are all flows.*

1.3 Exercise *Prove that if f and g are **any** two flows on a flow network, then $f + g$ is also a flow.*

1.4 Exercise *This problem has 4 parts. You need to write something for parts 1, 3, and 4. Part 2 is just for you to read and understand.*

The simplest kind of flow is one that comes from a path in the graph from s to t. And this is also a very useful kind of flow, as we will see below. Let us consider the flow network in Figure 3.

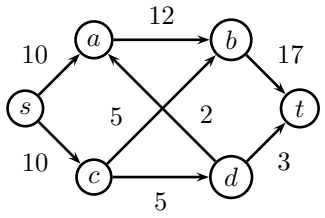
1. *Consider the path $s \to a \to e \to c \to t$ in this network. Suppose that every edge in this path carries a certain amount—the same amount for each edge in the path. For simplicity, we can say this amount is 1.*

   This of course is not a flow, because not every edge has a number associated with it.

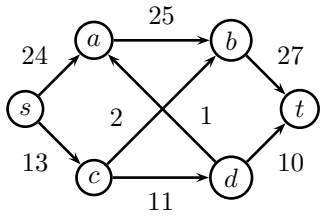
   We can extend what we have to a flow, however, by simply assigning the number 0 to every other edge.

   Show that what we get is actually a flow.

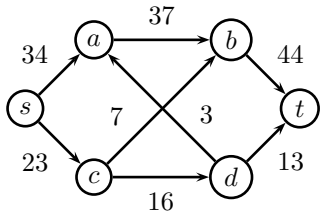
2. *You don't have to write anything for this part of the problem. But I just want to point out that you should be able to see that the flow that you draw has the properties that*



the flow f



the flow g



the flow $f + g$ ◇

Figure 2: Adding two flows.

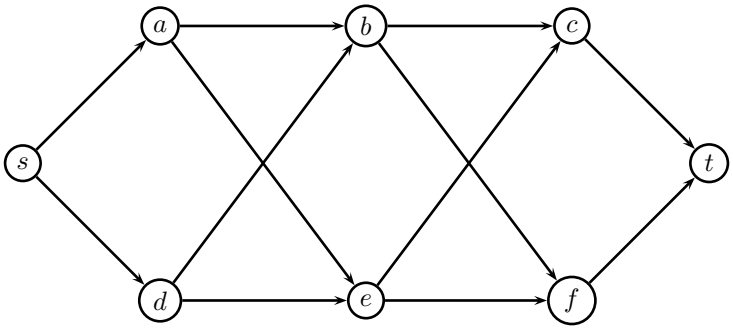


Figure 3: A simple flow network.

(a) *You can produce the flow starting with the path I gave you.*

(b) *If on the other hand you start with the flow, you can figure out what the path was.*

3. *Now suppose that the path remains the same, but that we change the network so that the edge $a \to e$ in the network reverses its direction. You should be able to see that we still have a network.*

   Show that the original path I gave you, containing the original edge from a to e, still corresponds to a flow in this new graph (even though in the new graph the edge goes from e to a). For a hint, just look back to Figure 1.

4. *Finally, let's generalize this result: Suppose that we have a flow network, and suppose that $s = v_0 \to v_1 \to \cdots \to v_n = t$ is a path from s to t, where each edge $v_i \to v_{i+1}$ of the path is either an edge or a reversed edge in the flow network. Suppose that each such edge $v_i \to v_{i+1}$ on the path carries a certain positive amount—say, 1.*

   Show that this corresponds naturally and uniquely to a flow in the flow network.

1.2.4 The value of a flow

Definition *If f is a flow, the **value** $|f|$ of the flow is the flow out of the source. To be precise,*

$$|f| = \sum_{v \in V} f(s, v)$$

Remark *Be careful! This is not an absolute value sign!*

Now you might think that this is a rather unsymmetric definition. Why are we just looking at the flow out of s? Why not look at the flow into t?

1.5 Exercise *Prove that $|f|$ (i.e., the total flow out of s) equals the total flow into t.*

Hint: first show that the following equation must be true:

$$\sum_{\substack{x \in V \\ y \in V}} f(x, y) = \sum_{y \in V} f(s, y) + \sum_{x \in V \setminus \{s,t\}} \sum_{y \in V} f(x, y) + \sum_{y \in V} f(t, y)$$

Then, once you have shown it is true, you should be able to show that

- *The left-hand side is 0. To do this, you use one of the three conditions defining a flow.*
- *The middle term on the right-hand side is 0. To do this, you use a different one of the three conditions defining a flow.*

And then you should be able to finish up almost immediately.

1.6 Exercise *Prove that if f and g are two flows, then*

$$|f + g| = |f| + |g|$$

1.3 Cuts

Definition *A* cut *in G is a partition of the vertex set V into two sets X and $\overline{X}$ such that $s \in X$ and $t \in \overline{X}$.d*

Thus, a flow network in general has many cuts.

If f is a flow and $(X, \overline{X})$ is a cut, the flow across the cut *is defined to be*

$$f(X, \overline{X}) = \sum_{v \in X, w \in \overline{X}} f(v, w)$$

1.7 Lemma *If f is a flow and $(X, \overline{X})$ is a cut, the flow across the cut is equal to the flow value $|f|$.*

PROOF. First, note that[3]

$$\begin{aligned} \sum_{\substack{v \in X \\ w \in V}} f(v, w) &= \sum_{w \in V} f(s, w) + \sum_{\substack{v \in X \setminus \{s\} \\ w \in V}} f(v, w) \\ &= |f| + 0 \\ &= |f| \end{aligned}$$

The proof that the second term on the right-hand side of the first line is really 0 again depends on one of the three conditions defining a flow. To be precise, the last sum in the first line vanishes because v is neither s nor t, and so we can apply the conservation property.

Using this, we see then that

$$\begin{aligned} f(X, \overline{X}) &= \sum_{\substack{v \in X \\ w \in \overline{X}}} f(v, w) \\ &= \sum_{\substack{v \in X \\ w \in V}} f(v, w) - \sum_{\substack{v \in X \\ w \in X}} f(v, w) \\ &= |f| - 0 \\ &= |f| \end{aligned}$$

where here the second sum in the second line vanishes because of skew-symmetry. □

1.4 The capacity of an edge

Now to make our problem realistic, we assume that each edge has a maximum capacity that it can carry. That is, the flow along that edge can be no more than the capacity of the edge.

So there is a **capacity function** $c(e)$ which assigns to each edge a non-negative real number. And we are going to restrict ourselves to considering flows that respect that capacity. That is, we insist that for each edge e,

$$0 \leq f(e) \leq c(e)$$

[3]The phrase "note that" when used in this way is really a sneaky way of saying "prove that". I usually means that the proof is not hard, but you do have to prove it.

If (v, w) is a (directed) edge, we also use the notation $c(v, w)$ for the capacity of that edge. And if (v, w) is not an edge (and in particular, if it is a "reverse edge"), then we set $c(v, w) = 0$[4].

We can also talk about the *capacity of a cut*: if $(X, \overline{X})$ is a cut, we define its capacity $c(X, \overline{X})$ by

$$c(X, \overline{X}) = \sum_{\substack{v \in X \\ w \in \overline{X}}} c(v, w)$$

1.8 Exercise *Prove that if $(X, \overline{X})$ is a cut, then*

$$f(X, \overline{X}) \leq c(X, \overline{X})$$

or equivalently,

$$|f| \leq c(X, \overline{X})$$

1.5 The actual statement of the problem

With this, we are now ready to state exactly what our problem consists of:

> Given a flow network with a capacity function, we want to find the flow respecting that capacity and having the greatest possible value.

We call such a flow a *maximum flow*. We're trying to find a maximum flow.

2 Solving the problem

2.1 Representation of flows and capacities on a flow network

In preparation for what comes later, we are actually going to represent capacities as *pairs* of numbers—one of the numbers will represent the capacity in the direction of the flow, and the other number will represent the capacity in the opposite direction. Of course the way we have set things up, the "capacity in the opposite direction" will be 0. But we will nevertheless find this notation very useful as we g on. Figure 4 shows an example of a flow network with a flow.

In that figure, the two numbers in square brackets represent the capacity. The first number is the capacity in the direction *against* the direction of the edge, and the second number is the capacity in the direction *in* the direction of the edge. It's certainly the case that when we start out like this, the first number is ≤ 0 (in fact, it *is* 0), and the second number is ≥ 0. As we will see, as we go on, this always remains true.

The number after the square brackets is the actual flow. Of course it has to be between the two capacities. So in this case it's always positive, which means that the flow is in the direction of the corresponding edge.

[4]So notice that, in contrast to the way we define flows, the function c is always non-negative and is *not* skew-symmetric.

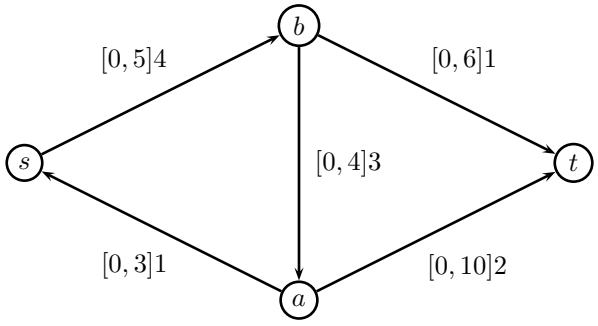


Figure 4: A flow network

Notice also that the flow from node a to node s is "backwards". That is, if we were really interested in transporting material from s to t, we certainly wouldn't be sending back from a to s. But this *is* a legal flow, nonetheless, because

- it is in fact a flow[5], and
- it respects the capacity constraints.

2.2 The residual graph for a flow

Given a flow on a network, as in Figure 4, we can ask ourselves this question: how much could the flow on each edge change, in either direction? The answer for the case of Figure 4 is shown in Figure 5. It is called the *residual graph* of the flow.

Again, in this figure, the two numbers in brackets represent allowable values for the flow *against* and *in the direction of* the edge. Here, though, one of them can actually be negative.

Where do those numbers come from? It's simple: we just take the original flow on that edge and subtract it from the two original capacities of that edge. So for instance, in Figure 4, the edge $s \to b$ has capacities $[0, 5]$ and flow 4. We subtract 4 from 0 and 5 and get $[-4, 1]$. Why do we say this is the range of allowable flows along that edge? Because if we take any value between -4 and 1 and add it to the original flow (which is 4), we get a number between 0 and 5, which are the capacities of that edge.

2.1 Exercise *Show that when we do this, it is still true that the first of the pair of capacities at each edge is* ≤ 0 *and the second is* ≥ 0.

[5] And notice that we really do have a flow here. That is, the conservation condition is satisfied at the two nodes a and b. And you should be able to answer this question: What is the *value* of this flow? (No! I'm not going to give you the answer!)

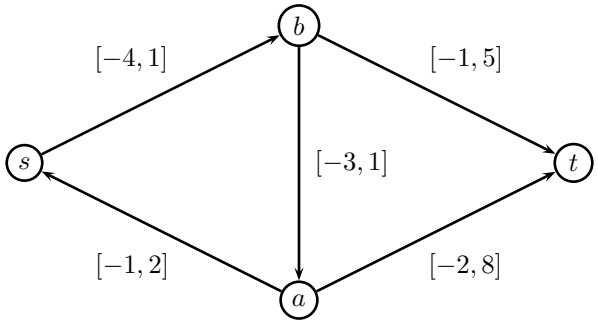


Figure 5: The residual graph for the flow in Figure 4

To allow for this possibility, we are going to say that an edge in the residual graph can go in either direction. The edges that we draw are there to show which of the two numbers we use. So if we introduce an edge from $s \to a$, that edge can carry a flow anywhere from 0 to 1. And if we introduce an edge from $a \to s$, that edge can carry a flow anywhere from 0 to 2.

It's important to realize that this is just something we do in the residual graph. Any edge in the original graph must be one of the edges that was there in the first place.

We say that a flow g in the residual graph is *allowable* if and only if for each edge e, $g(e)$ lies in the interval defined by the pair of numbers in the residual graph corresponding to that edge.

And so it should be clear that if f is the flow in the original graph, and g is any allowable flow in the residual graph, then $f + g$ is a flow which satisfies the original capacity constraints.

So what we need to do is to start with a flow f. Then form the residual graph for that flow. Then find a flow g in the residual graph such that $f+g$ has a greater value than f. And then just continue this process. That's the idea, anyway.

Note, by the way, that we could (in principle, anyway) have a flow in the residual graph whose value on the edge $a \to s$ was -1. That would actually be a good choice to make (if we could), because that corresponds to a flow in the opposite direction $s \to a$, and that's what we're really looking for. We want a flow that takes as much *from* s and transports it *to* t.

2.3 Augmenting paths

So how do we find an allowable flow in the residual graph? Well, the easiest kind of flow to find is a path from s to t where each edge carries the same amount. So let us look for such paths. Here's how we do it:

Suppose $s = v_0 \to v_1 \to \cdots \to v_n = t$ is a path from s to t. Each edge on that path can carry an amount determined by the residual capacity constraints. So for instance, let us consider the path $s \to a \to b \to t$ in our residual graph. The maximum amount on each of these edges is shown in

Figure 6.

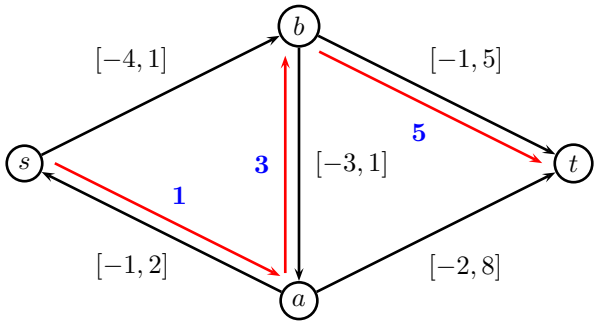


Figure 6: A path in the residual graph from Figure 5. The **blue** number on each edge of the path is the maximum amount possible for the additional flow along that edges, as determined by the residual constraint for that edge.

Note that I'm not asserting that this is necessarily the best path to pick for what we are doing. I'm just using it as an example.

Now for any such path, we let r be the minimum of the maximum possible values along the edges of that path. In this case, the minimum is

$$r = \min\{1, 3, 5\} = 1$$

We call this the *residual capacity* along that path. So if we write that path with its residual capacity, we have Figure 7. We call this path with its associated flow values an *augmenting path.*

2.2 Exercise *Show that an augmenting path, when arrived at in this manner, respects the capacities on each edge.*

We actually need to make one more restriction. It's simple, but it's a key constraint. We define an augmenting path as follows:

Definition *An* augmenting path *is a path from s to t with a positive residual capacity.*

That is, the flow along the path has to be > 0. If the flow along the path is 0, it's not an augmenting path.

That augmenting path with its associated residual capacity corresponds by Exercise 1.4 to a flow which we can call the *residual flow.* Let's call that residual flow g. If we add this residual flow g to our original flow f, we get the new flow $f + g$ shown in Figure 8

In that figure, the purple amounts are the amounts that have changed. There are two important things to note:

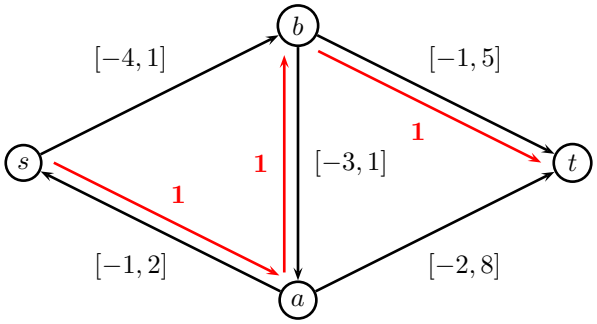


Figure 7: An augmenting path in the residual graph from Figure 5. The **red** number on each edge of the path is additional flow along that edge of the path.

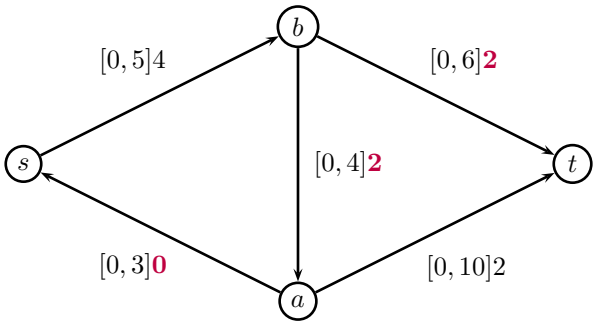


Figure 8: The new augmented flow $f + g$. The **purple** amounts are the ones that have changed.

1. We still have an allowable flow in our graph. That's because the adjustment we made was within the bounds allowed by the "residual" computation. Note that while the augmenting path had two edges "in the wrong direction", the final graph does not.

2. The value of the flow in the new path is greater than the original value. In fact, we have $|f + g| = |f| + |g| = |f| + r > |f|$.

So we have the beginnings of an algorithm here:

- Start with any allowable flow.

- From it, produce the residual graph.
- Use that residual graph to find any augmenting path.
- Add the augmenting path (or more precisely, the flow corresponding to that augmenting path) to the original flow to produce a new flow. The new flow will still be allowable and will have a greater value.
- Repeat this whole process until nothing more can be done.

3 Does the algorithm really work?

Well, that's all very nice, but there are two questions that come up immediately:

1. Is it possible that we might have a non-maximum flow f, but nevertheless there was no augmenting path for f? If there was, then of course the algorithm would fail to yield a maximum flow.
2. Is it possible that the algorithm fails in some other way? For instance, maybe it never ends.

3.1 The max-flow min-cut theorem

The second question actually has a pretty curious answer. But the first question can be answered right away, based on a remarkable theorem due to Ford and Fulkerson, who laid the foundations for this whole theory and wrote the original book on the subject[6].

3.1 Theorem (The max-flow min-cut theorem) *If $G = (V, E)$ is a flow network with capacity function c and source and sink nodes s and t, the following statements are equivalent:*

1. *f is a maximum flow.*
2. *There is no augmenting path for f.*
3. *There is some cut $(X, \overline{X})$ for which*

$$|f| = c(X, \overline{X})$$

PROOF. **(1) $\Longrightarrow$ (2).** If there is an augmenting path p for f, then we can increase the flow value by increasing the flow along the edges of p.

(2) $\Longrightarrow$ (3). Suppose there is no augmenting path for f. Let X be the set of vertices reachable from s on a path on which each edge has positive residual value. Trivially, X includes s, and by our assumption, X does not include t. Thus, $(X, \overline{X})$ is a cut. Further, if (v, w) is an edge in the original graph that crosses the cut (i.e., with $v \in X$ and $w \in \overline{X}$), we must have $f(v, w) = c(v, w)$, since otherwise w would also be in X. Thus, we have

$$|f| = \sum_{\substack{v \in X \\ w \in \overline{X}}} f(v, w) = \sum_{\substack{v \in X \\ w \in \overline{X}}} c(v, w) = c(X, \overline{X})$$

[6] *Flows in Networks*, by L.R. Ford and D.R. Fulkerson. Princeton University Press, 1962.

(3) $\Longrightarrow$ **(1).** We know from Exercise 1.8 that in any case, $|f| \leq c(X, \overline{X})$. The fact that with this cut they are equal means that there is no flow with a greater value than f. □

Note that as a consequence of this theorem, we know that the value of the maximum flow in the network is equal to minimum capacity of any cut. That's where the theorem gets its name.

So this answers the first of our questions: if f is not maximum, we know that there will be at least one augmenting flow, so the algorithm can make progress.

3.2 Termination and correctness

Next, we need to know if the algorithm is guaranteed to terminate. Here's where things get complex.

First of all, let us suppose that all the capacities of the network are integers. In that case, there is something we can say, provided we make two very natural assumptions:

- We assume that we start with a flow that is identically 0. (We'll always assume this in any case; it's the natural way to start.)
- We assume that when we pick an augmenting path, we make the flow along that path as large as possible. That is, we let it be the minimum of the residual values of each edge on the flow.

In this case, the augmenting flow value will of necessity be an integer, and so we will always be increasing the value of the flow by an integer each time we pick an augmenting path. Since the maximum flow value is finite, this process has to stop. This also shows that the maximum flow is an integral flow—the flow along each edge is an integer[7].

Now one might think that this was a bit arbitrary. For instance, there are many flow networks that arise in practice in which the capacities are not necessarily integers. However, this is not really a problem. If they are rational numbers, then we can always reason in exactly the same way by using as our "minimal unit" the fraction which is 1 over the least common denominator of all the capacities. Exactly the same argument then works. And of course computers only really deal with rational numbers in any case.

The problem of irrational numbers is peculiar, though. Ford and Fulkerson actually gave an example in which the capacities were irrational, and they showed that by picking the augmenting paths in that example in a particularly stupid manner, then—even though the flow along these augmenting paths was chosen to be as big as possible—the following would happen:

- The process would not terminate. Now of course the values of the successive flows would keep increasing—that has to be true. But they would always be less than the maximum possible value.
- But it's even worse than that. Because the values of the successive flows keep increasing, they have a limit. But this limit is strictly less than the value of the maximum flow in the network.

I'm not going to bother reproducing that example here, but you can find it in their book. It's amusing, at least.

[7] I don't think this fact is *a priori* obvious.

In any case, we can always assume that our capacities are rational, and so our algorithm—however we pick the augmenting paths—will definitely terminate[8], and give the maximum flow[9].

3.3 Efficiency

However, this doesn't lay the matter to rest. There still is the question of efficiency: how well can we pick the augmenting paths so that we have to repeat the iterations of the algorithm as few times as possible? This question has been considered over many years, and successively better algorithms have been produced.

First, let's see the kind of bad algorithmic behavior that can happen. Suppose we start with the network in Figure 9.

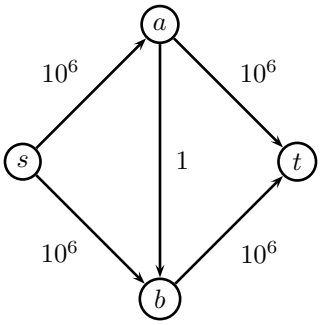


Figure 9: A network with a possible sequence of very bad augmenting paths. The number on each edge is the capacity of that edge.

3.2 Exercise *Show that there is a sequence of augmenting paths for the graph in Figure 9 that reaches the maximum possible flow very slowly—in fact, it takes* $2 \cdot 10^6$ *steps (each step being a successive augmenting path) to get to the maximum flow.*

In fact, this is a famous example, and occurs in many places, including in our text. So I'm not assigning it. But you should try it yourself in any case, and check by what is shown in the text.

The point here is that in general, if the maximum flow is denoted by f^*, then it might take $|f^*|$ iterations (in this case, actually 2 times that many) of the algorithm to get to the maximum flow, if we choose the augmenting paths in a particularly stupid manner.

It turns out, however, that we can do much better. In fact, there are algorithms by which we can choose the augmenting paths so that the number of paths does not depend at all on $|f^*|$, but only on the underlying graph G.

There is an extensive theory of successively better ways to do this. We won't cover much of it—in fact, we'll only cover the first result that was discovered. It's interesting in itself, however, and

[8] You would want to make sure you use integer arithmetic in your procedure, in order not to confuse the matter with roundoff errors.

[9] And there are even algorithms that will pick augmenting paths in such a way that even if the capacities are irrational, we get a terminating procedure.

shows the kind of reasoning that we can expect to use in these kinds of problems. The result is due to Edmonds and Karp[10]. (And there is an equivalent algorithm, due to Dinic[11], that was discovered independently at about the same time, and actually published slightly earlier.)

The Edmonds-Karp algorithm is simply a way to specify which augmenting path to pick next at each step. The specification is simple: pick the shortest augmenting path from s to t. (And by "shortest", we mean simply the path with the least number of edges—that is, we completely ignore the residual capacities.) How do we find the shortest path? That's easy—we can use breadth-first search for that. Just start at s, and stop in the breadth-first search algorithm when we first get to t. Then walk backwards in the tree to get the path from s to t. It will be a path of minimal length.

So now let us show that the Edmonds-Karp algorithm really works pretty well. We will use the following notation:

If f is a flow in the graph G, then by G_f we denote the residual graph corresponding to f. Further, if u and v are nodes in G, then by $\delta_f(u, v)$ we denote the shortest-path distance from u to v in G_v. That means, in particular, that we consider only paths in G each edge of which can carry a positive flow value in the residual graph G_f.

3.3 Lemma *If the Edmonds-Karp algorithm is run on a flow network $G = (V, E)$ with source s and sink t, then for all vertices $v \in V \setminus \{s, t\}$, the shortest-path distance $\delta_f(s, v)$ in the residual graph G_f increases with each flow augmentation.*

Remark *By "increases", we mean "weakly increases". That is, if g is the flow corresponding to the next step in the process, then*

$$\delta_f(s, v) \leq \delta_g(s, v)$$

Proof. We will prove this by contradiction.

1. Suppose that for some vertex $w \in V \setminus \{s, t\}$ there is a flow augmentation that causes the shortest-path distance from s to w to decrease. Let f be the flow just before the first augmentation that decreases the shortest-path distance from s to w, and let g be the flow just afterward. That is, we assume that

   $$\delta_g(s, w) < \delta_f(s, w)$$

   We will show this leads to a contradiction.

   Let v be the vertex with minimum $\delta_g(s, v)$ whose distance was decreased by the augmentation, so that $\delta_g(s, v) < \delta_f(s, v)$. (So it might be that $v = w$, but it also might be that v is some vertex "closer to s in G_g".)

   Let p be the shortest path from s to v in G_g, and let u be the predecessor of v on that path. So (u, v) is an edge in G_g and (by the usual cut-and-paste argument)

   $$\delta_g(s, u) = \delta_g(s, v) - 1 \tag{1}$$

[10] Jack Edmonds and Richard M. Karp. Theoretical improvements in the algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

[11] E.A. Dinic. Algorithm for solution of maximum flow in a network with power estimation. *Soviet Mathematics Doklady*, 11(5):1277–1280, 1970.

Because of the particular way we chose v, we know that the distance of u from s did not decrease. That is, we know that

$$\delta_g(s,u) \geq \delta_f(s,u) \tag{2}$$

2. We will now show that the edge (u,v) in the graph G is not in G_f. Here is the reason: If (u,v) were in G_f, then $\delta_f(u,v) = 1$, and so we would have the following:

$$\begin{aligned}
\delta_f(s,v) &\leq \delta_f(s,u) + \delta_f(u,v) \\
&= \delta_f(s,u) + 1 \\
&\leq \delta_g(s,u) + 1 && \text{by (2)} \\
&= \delta_g(s,v) && \text{by (1)}
\end{aligned}$$

which contradicts our assumption that $\delta_g(s,v) < \delta_f(s,v)$. So it must be that the edge (u,v) in the graph G is not in G_f.

3. Thus we have $(u,v) \notin G_f$ but $(u,v) \in G_g$. The only way this could happen is if the augmentation increased the flow from v to u.[12] Now our algorithm always augments flow along shortest paths, and therefore the shortest path from s to u in G_f must have (v,u) as its last edge.[13] Therefore,

$$\begin{aligned}
\delta_f(s,v) &= \delta_f(s,u) - 1 \\
&\leq \delta_g(s,u) - 1 && \text{by (2)} \\
&= \delta_g(s,v) - 2 && \text{by (1)}
\end{aligned}$$

But this contradicts our assumption that $\delta_g(s,v) < \delta_f(s,v)$. So the assumption must have been incorrect, and the lemma is proved. □

3.4 Theorem *If the Edmonds-Karp algorithm is run on a flow network $G = (V,E)$ with source s and sink t, then the total number of flow augmentations performed by the algorithm is $O(VE)$.*

Remark *Note that we really should write $O(|V||E|)$. What we have written is a common "abuse of notation" which is harmless and almost always makes things easier to read.*

PROOF. 1. Suppose p is an augmenting path in a residual graph G_f. We say that an edge (u,v) of p is *critical* iff the residual capacity of p is the residual capacity of (u,v)—that is, if $c_f(p) = c_f(u,v)$. At least one edge on any augmenting path must be critical.

Further, after we have augmented a flow using an augmenting path, any critical edge on the path disappears from the residual network.

We will show that each of the $|E|$ edges in G can become critical at most $|V|/2$ times.

2. Suppose (u,v) is an edge in E. Since augmenting paths are (in the Edmonds-Karp algorithm) shortest paths, when (u,v) is critical for the first time (say in the residual network G_f), we must have

$$\delta_f(s,v) = \delta_f(s,u) + 1$$

[12] Be careful—make sure you really understand what this is saying.
[13] Again, be careful: why the *last* edge?

Once the flow is augmented, the edge (u, v) disappears from the residual network. It cannot reappear later on another augmenting path until after the flow from u to v is decreased, which occurs only if (v, u) appears on an augmenting path. If g is the flow in G when this event occurs, then we have

$$\delta_g(s, u) = \delta_g(s, v) + 1$$

Since $\delta_f(s, v) \leq \delta_g(s, v)$ by Lemma 3.3, we have

$$\begin{aligned} \delta_g(s, u) &= \delta_g(s, v) + 1 \\ &\geq \delta_f(s, v) + 1 \\ &= \delta_f(s, u) + 2 \end{aligned}$$

Now certainly for any flow f, $1 \leq \delta_f(u, v) \leq |V|$. So by what we have just shown, (u, v) can become critical at most $|V|/2$ times. Since there are $O(E)$ pairs of vertices that can have an edge between them in a residual network,the total number of critical edges during the entire execution of the Edmonds-Karp algorithm is $O(VE)$. Each augmenting path has at least one critical edge, and so the number of augmenting paths in the course of the algorithm is at most $O(VE)$, and the theorem is proved. □

3.5 Theorem *The running time of the Edmonds-Karp algorithm is $O(VE^2)$.*

PROOF. The number of augmenting paths we need to produce is $O(VE)$. The cost of producing each augmenting path is the cost of the breadth-first search from s to t. This is ostensibly $O(V + E)$. However, since we are only concerned with vertices that are reachable from s, each such vertex corresponds to an edge, and so $|V| = O(E)$. Thus the cost of producing each augmenting path is $O(E)$, and the total cost of the algorithm then is $O(VE^2)$. □

4 An application: the "marriage problem"

This is actually a general problem in graph theory, but we'll first describe it in the way that everyone thinks about it, since it's so easy to remember that way.

Suppose we have two finite sets G ("girls") and B ("boys"). Each girl likes one or more boys, and each boy likes one or more girls. Let's assume that if a girl and a boy both like each other, they would be happy to marry each other.

The problem then is: is there a way of marrying all the girls and all the boys to someone they each like? If so, we say that the marriage problem is *solvable*.

Of course this could only be done if there were an equal number of girls and boys. Let's assume that there are n girls and n boys.

Even so however, it's not so simple to decide the question. For instance, suppose all the girls liked one boy (and he liked all the girls). And suppose all the boys liked one girl (and she liked all the boys). There will be a lot of unhappy people in this case.

To make the wording in what follows less cumbersome, let us agree that when we say that "a girl g likes a boy b", we mean without specifically having to say so that also the boy b likes the girl g. (If

we are modeling a real problem and a girl likes a boy, say, but the boy doesn't like the girl, we just ignore that, and say nothing about either one liking the other.)

In 1935, the British mathematician Philip Hall proved a theorem that gives a necessary and sufficient condition for the problem to be solvable:

4.1 Theorem (Hall's "marriage theorem") *The marriage problem is solvable iff for each r (with $1 \leq r \leq n$), every set of r girls likes at least r boys*[14].

The necessary and sufficient condition provided by this theorem seems one-sided: presumably one could instead assume that each set of r boys likes at least r girls. It turns out that they are equivalent, and we state that as a lemma now, because we will need this fact in the proof of the theorem below.

4.2 Lemma *If for each r (with $1 \leq r \leq n$), every set of r girls likes at least r boys, then also every set of r boys likes at least r girls.*

Proof of the lemma. It will help to refer to Figure 10. Suppose B_0 is a set of r boys. Let G_0 be the set of girls that those boys like. (Equivalently, G_0 is the set of girls that like any of those boys.) We need to show that G_0 contains at least r girls.

Let us set

$$B_1 = B \setminus B_0$$
$$G_1 = G \setminus G_0$$

And just for notational convenience, let us define g_0 to be the size of the set G_0, and similarly for g_1, b_0, and b_1. Now the set of girls G_1 likes only boys in B_1. (That's by definition; any girl who likes a boy in B_0 is automatically in G_0.)

Therefore by the assumption of the lemma, we must have $g_1 \leq b_1$. (It might be that g_1 is strictly less than b_1, but that's OK also.)

Therefore it follows that $g_0 \geq b_0$, and that's what we needed to prove. □

Proof of the theorem.

1. It's clear that if the problem is solvable, then it's certainly true that every group of r girls likes at least r boys, since each one likes at least the one that she will marry, and possibly others as well, and none of those boys is going to marry more than one of the r girls, so there are at least r boys liked by the r girls.

2. Thus, we have to prove the other direction: if each group of r girls likes at least r boys, then the problem is solvable.

   We can model this as a graph problem: the nodes in our graph will be the union of the sets G and B. There is an (undirected) edge between a node $g \in G$ and $b \in B$ iff they like each other. And that's all the edges there are.

   This is an example of what is called a bipartite graph. We want to know if there is a map m from G to B such that the map is 1-1 and onto and such that $m(g) = b$ only if there is an edge between g and b.

   We will turn this problem into an equivalent maximum flow problem. Here's how:

[14] By this we simply mean that each one of the r boys is liked by (and likes) *at least* one of the r girls.

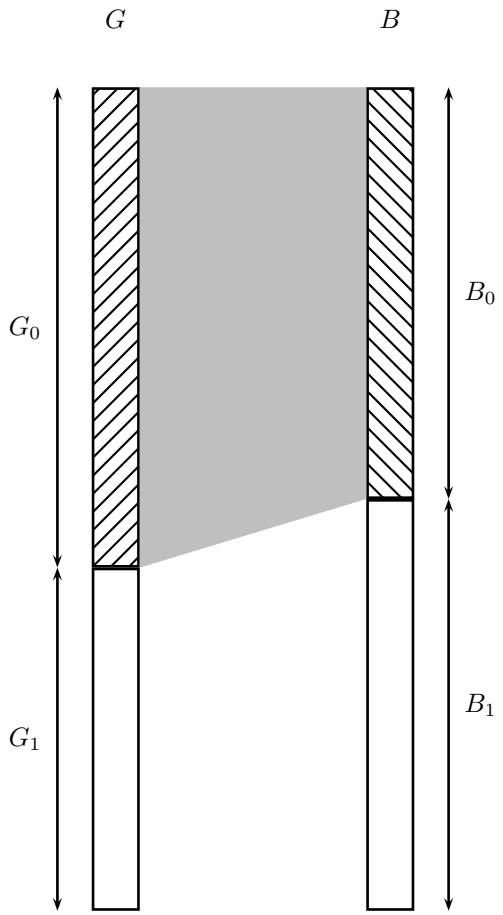


Figure 10: Illustration for the proof of Lemma 4.2.

- First, we make the graph a directed graph. Every edge is between a girl and a boy. We make the edge a directed edge from the girl to the boy.
- We introduce two new nodes, s and t.
- We introduce an edge from s to each girl, and we introduce an edge from each boy to t.

Then we introduce capacities on these edges to turn this into a network flow problem. This is very simple: the capacity of each flow is 1.

The question is then to find the maximum flow from s to t. It is clear that the marriage problem is solvable iff the maximum flow has value n. Certainly the maximum flow can't be greater than n. So to show that the marriage problem is solvable, it is enough to show that the maximum flow has value $\geq n$.

Now the max-flow min-cut theorem tells us that the maximum flow will have value n iff the

cut of minimum capacity has capacity n. Thus it is enough to show that the capacity of every cut is $\geq n$. And we are assuming that every group of r girls likes at least r boys.

So say $(X, \overline{X})$ is a cut. We could handle this all at once[15], but just to make it easier to see what is going on, let us divide this into a number of different cases:

Case 1: $X = \{s\}$. In this case, $c(X, \overline{X})$ is just the sum of the capacities of the n edges from s to G. And this is n.

Case 2: $\overline{X} = \{t\}$. In this case $c(X, \overline{X})$ is just the sum of the capacities of the n edges from B to t. And this is also n.

Case 3: $X \subseteq \{s\} \cup G$. (So in particular, $\overline{X}$ includes all of B.) Let $G_0 = X \cap G$. (G_0 might be all of G, but doesn't have to be.)

Say the number of elements of G_0 is r. $c(X, \overline{X})$ counts the following edges:

- The edges leaving G_0. By our assumption, the number of these edges is $\geq r$
- The edges entering $G \setminus G_0$. The number of these edges is just $n - r$.

So $c(X, \overline{X}) \geq n$.

Case 4: $\overline{X} \subseteq B \cup \{t\}$. (So in particular, X includes all of G.) Let $\overline{X}_0 = \overline{X} \cap B$. ($\overline{X}_0$ might be all of B, but it doesn't have to be.

Say the number of elements of $\overline{X}_0$ is r. $c(X, \overline{X})$ counts the following edges:

- The edges entering $\overline{X}_0$—and by our assumption and the result of the lemma, the number of these edges is $\geq r$.
- The edges leaving $B \setminus \overline{X}$. The number of these edges is just $n - r$.

So $c(X, \overline{X}) \geq n$.

Case 5: $\overline{X} \cap G \neq \varnothing$ and $\overline{X} \cap B \neq \varnothing$. Set

$$\overline{X}_0 = \overline{X} \cap G$$
$$\overline{X}_1 = \overline{X} \cap B$$

It will help to look at the picture in Figure 11 to understand what is going on here.

Say the number of elements of $\overline{X}_1$ is r.

Let A be the set of elements of G which have edges leaving them and arriving at elements of $\overline{X}_1$. We know by the lemma that there are at least r elements of A.

Let us set

$$A_0 = A \cap X$$
$$A_1 = A \cap \overline{X} = A \setminus A_0$$

Let the number of elements of A_0 be a_0, and the number of elements of A_1 be a_1. So we know that $a_0 + a_1 \geq r$.

$c(X, \overline{X})$ counts the following edges:

- The edges from $B \setminus \overline{X}_1$ to t. The number of these edges is $n - r$.

[15] Case 5 of the set of cases that immediately follows really would handle all the cases at once.

- The edges entering $\overline{X}_1$ that come from A_0. Since each element of A_0 has at least one edge leaving it and arriving at $\overline{X_1}$, there are at least a_0 such edges.
- The edges leaving s and entering A_1. There are exactly a_1 such edges.

Thus $c(X, \overline{X}) \geq (n - r) + a_1 + a_0 = (n - r) + r = n$

And so we are done. □

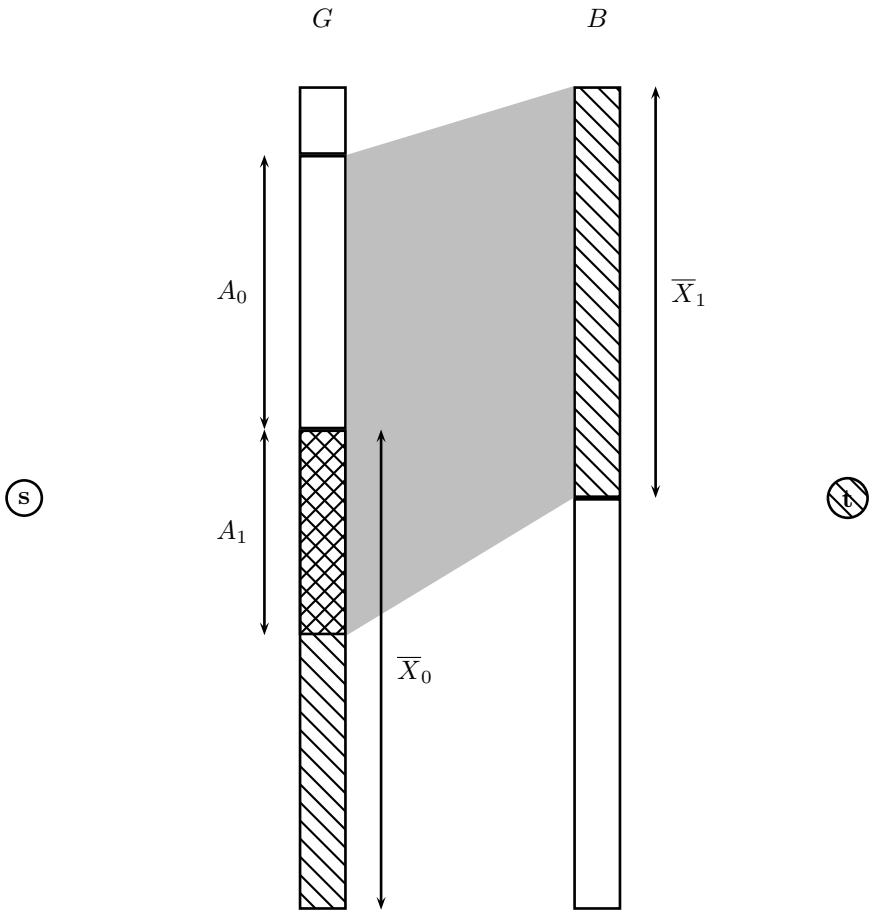


Figure 11: Illustration for Case 5 of the proof of Hall's marriage theorem

5 Application: baseball elimination

This really applies to any kind of league sports, but (in this country, anyway), it gets discussed mainly in the context of baseball. There are a number of teams in a league. They all play the same number of games. At the end of the season, the team with the greatest fraction of games won is the winner. (This is slightly simplified, but the basic idea is correct.) We assume that each game is either won or lost (that is, there are no ties), and that a team can be a winner even if it ties for first place.

Now it often happens that well before the end of the season one can predict that a team could not possibly wind up in first place. In such a case, that team is said to be "mathematically eliminated". This is not always so easy to see, however, and sports reporters have frequently made mistakes in this matter.

Here is a famous example (which is used over and over again in this area). It shows the standings in the American League East on August 30, 1996.

Team	Won-Lost	Left	NYY	BAL	BOS	TOR	DET
New York Yankees	75-79	28		3	8	7	3
Baltimore Orioles	71-63	28	3		2	7	4
Boston Red Sox	69-66	27	8	2		0	0
Toronto Blue Jays	63-72	27	7	7	0		0
Detroit Tigers	49-86	27	3	4	0	0	

The question is: Can Detroit possibly wind up in first place (or tied for first place), or is it at this point mathematically eliminated?

At first, it appears that Detroit might still be able to get to the top: it has 27 games more to play, and if it wins all of them it will have won 76 games. If the Yankees lose all of their remaining games (or even win just one of them), then Detroit might still wind up in first place.

However, it turns out that this is impossible, and the reason is that the other teams have to play other games with each other as well, and if the Yankees lose that many games, then some other team will wind up ahead. Here's how it might have been explained[16]

> By winning all of their remaining games, Detroit can finish the season with a record of 76 and 86. If the Yankees win just 2 more games, then they will finish the season with a 77 and 85 record which would put them ahead of Detroit. So, let's suppose the Tigers go undefeated for the rest of the season and the Yankees fail to win another game.
>
> The problem with this scenario is that New York still has 8 games left with Boston. If the Red Sox win all of these games, they will end the season with at least 77 wins putting them ahead of the Tigers. Thus, the only way for Detroit to even have a chance of finishing in first place, is for New York to win exactly one of the 8 games with Boston

[16] I am copying this explanation and the accompanying flow network analysis from `http://riot.ieor.berkeley.edu/~baseball/detroit.html` and also from some notes by Jeff Erickson at `http://www.cs.illinois.edu/~jeffe/teaching/algorithms/`.

and lose all their other games. Meanwhile, the Sox must lose all the games they play against teams other than New York. This puts them in a 3-way tie for first place as shown in the table below.

Team	Won	Lost
Detroit Tigers	76	86
Boston Red Sox	76	86
New York Yankees	76	86
Baltimore Orioles	??	??
Toronto Blue Jays	??	??

Now let's look at what happens to the Orioles and Blue Jays in our scenario. Baltimore has 2 games left with with Boston and 3 with New York. So, if everything happens as described above, the Orioles will finish with at least 76 wins. So, Detroit can catch Baltimore only if the Orioles lose all their games to teams other than New York and Boston. In particular, this means that Baltimore must lose all 7 of its remaining games with Toronto. The Blue Jays also have 7 games left with the Yankees and we have already seen that for Detroit to finish in first place, Toronto must will all of these games. But if that happens, the Blue Jays will win at least 14 more games giving them at final record of 77 and 85 or better which means they will finish **ahead** of the Tigers. So, no matter what happens from this point in the season on, Detroit can not finish in first place in the American League East.

Well that's a convincing explanation, but it's a pretty painful one, and I wouldn't trust myself to be able to do that on a regular basis without making a lot of mistakes. Is there a better way?

Yes, there is. It turns out that this problem can be modeled as a network flow problem. Let's just do it for the case at hand. We will create a bipartite graph as follows: the two "parts" of the graph will be called G (for "games")and T (for "teams"). We are trying to figure out whether or not Detroit is mathematically eliminated, and we will do this by making nodes involving everything *except* Detroit. Then we will add start and sink nodes s and t, as usual, and our resulting graph is shown in Figure 12.

It's pretty clear how the graph is constructed:

- The numbers represent edge capacities.
- The capacity of each edge from s to a game node is simply the number of such games that have yet to be played. For instance, the New York Yankees still have 3 games to play with the Baltimore Orioles.
- The capacity of each edge from a team node to t is the only slightly tricky one: it is the most number of games that team could win and not beat the Detroit Tigers, assuming that the Detroit Tigers win all their games. So if the Detroit Tigers win all their remaining games, they will have won 76 games. The New York Yankees can be allowed to win at most 1 more game in order not to surpass that figure. And so on.
- The capacity of every other edge—i.e., the edges from the game nodes to the team nodes—is ∞ (or actually, just some very big number).

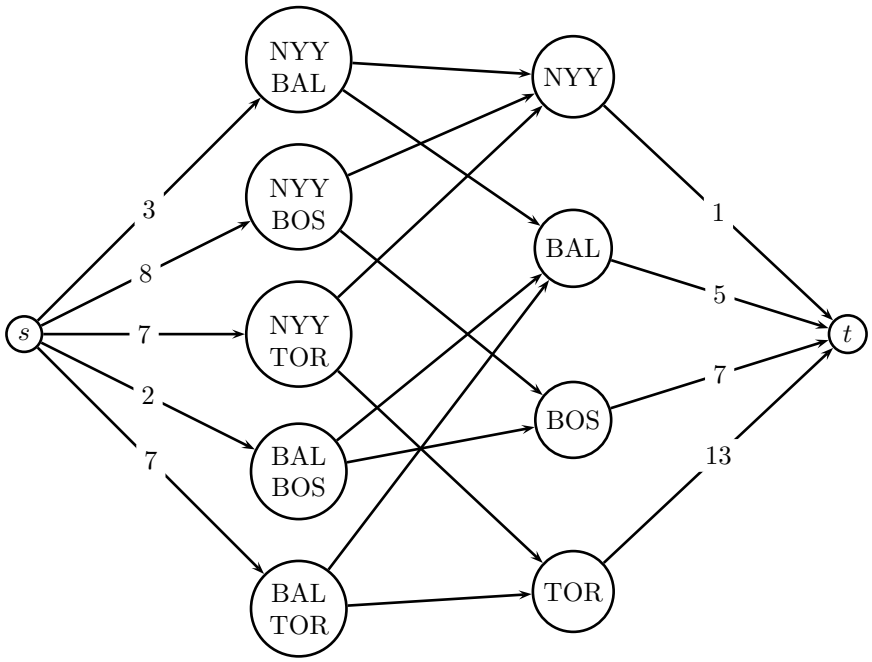


Figure 12: A flow graph for the American League East standings on August 30, 1996.

"Game" nodes are ones given the names of two teams, like the top one on the left, which represents all the games played between the New York Yankees and the Baltimore Orioles.

"Team" nodes are the ones given the names of just one team, like the top one on the right, which represents the New York Yankees.

The labels on the edges represent capacities. Unlabeled edges have infinite capacity.

It should be clear that the Detroit Tigers can win if there is a flow satisfying these constraints in which every game is played.

5.1 Exercise *Show why this is true. In particular, state clearly what flow is intended to pass over the "inner" edges (between the game nodes and the team nodes). Then explain why the sentence just before this exercise has to be true.*

And it's easy to see that this is impossible: The number of remaining games is the maximum possible flow from s, which is $3 + 8 + 7 + 2 + 7 = 27$. But the maximum flow that could arrive at t is only $1 + 5 + 7 + 13 = 26$. So Detroit can't win.

Isn't that neat?