18.310A lecture notes March 28, 2015

Network flows

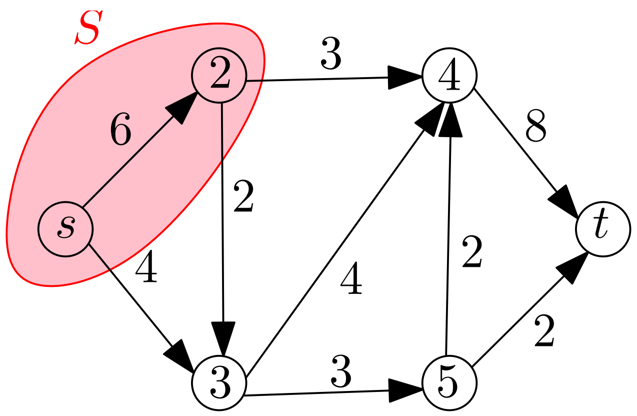
Lecturer: Michel Goemans

---

1 The maximum flow problem

A *network* consists of a set V of vertices (plural of *vertex*...), and set A of arcs (or directed edges) between vertices: an arc from i to j is represented by the ordered pair $(i, j) \in A$. Each arc $(i, j) \in A$ has a *capacity* denoted by $u(i, j)$, representing the maximum rate at which we can send "things" along this arc.

Here is an example network:



In this network we have two special vertices: a *source* vertex s and a *sink* vertex t. Our goal is to send the maximum amount of flow from s to t; flow can only travel along arcs in the right direction, and is constrained by the arc capacities. This "flow" could be many things: imagine sending water along pipes, with the capacity representing the size of the pipe; or traffic, with the capacity being the number of lanes of a road. Or a communications network, where the capacity represents the bandwidth of a particular link in the network, and the flow consists of data packets being sent. Network flows appear in many many settings.

The key property of a flow is *flow conservation*. Consider some vertex i, different from the source or the sink. Then the amount of flow going into vertex i should be the same as the amount of flow going out; no flow appears or disappears at this vertex.

In the above example, there is a flow of value 9 (you should be able to find such a flow pretty easily). But there is no flow of any larger value. How can we see that this is the case? Well, consider the set S shown in red on the figure. Since $s \in S$ and $t \notin S$, all flow from s to t needs to cross the boundary of S. In the example, the total amount of outgoing capacity crossing this boundary is 9; and so there cannot be a flow of larger value.

Let's formalize this notion a bit.

Definition 1. A *s-t-cut* is a subset $S \subset V$ such that $s \in S$, $t \notin S$. The *capacity* of an s-t-cut S is

$$\mathrm{cap}(S) = \sum_{(i,j)\in A : i \in S, j \notin S} u(i,j).$$

Note that we only count *outgoing* arcs towards the capacity of a cut; flow must traverse arcs in the forward direction, so incoming arcs are not useful in "escaping" the cut. The total amount of flow from s to t can't exceed the capacity of any s-t-cut. This is rather intuitive, but will be derived formally in the next section.

We use some shorthand and use "min cut" to refer to the minimum capacity of any s-t-cut. So we have

$$\text{max flow} \quad \leq \quad \text{min cut}.$$

But more is true!

Theorem 1. *max flow $\quad = \quad$ min cut.*

We will show two ways to prove this theorem. One is to express the problem of finding a maximum flow as a linear program, and use strong LP duality. The other one is to look at properties of maximum flows and show that we can derive from any maximum flow a corresponding minimum $(s-t)$ cut giving the same value.

2 Expressing as a linear program

Our decision variables will be

$$\begin{aligned} x_{ij} &= \text{amount of flow on arc } (i,j) \in A \\ z &= \text{total amount of flow sent from } s \text{ to } t. \end{aligned}$$

The trickiest constraint to deal with is flow conservation. Consider any vertex i different from s and t; since flow out is equal to flow in, we have

$$\sum_{j:(i,j)\in A} x_{ij} = \sum_{j:(j,i)\in A} x_{ji};$$

note the order of the indices, it matters! What about if $i = s$? Then the amount of flow going out should be larger than the amount coming in, by an amount z. (You might expect that there should be no flow at all coming in, and you would be right; but we don't need to exclude the possibility at the moment, and it makes the equations easier in fact.) So we have

$$\sum_{j:(s,j)\in A} x_{sj} = \sum_{j:(j,s)\in A} x_{js} + z.$$

Similarly,

$$\sum_{j:(t,j)\in A} x_{tj} = \sum_{j:(j,t)\in A} x_{jt} - z;$$

note the change in sign. We can put all three of these equations together as one, using some notation:

$$\sum_{j:(i,j)\in A} x_{ij} = \sum_{j:(j,i)\in A} x_{ji} \;+\; (\mathbf{1}_{i=s} - \mathbf{1}_{i=t})z.$$

Here, $\mathbf{1}_E$ is just the indicator of the statement E; it is 1 if E is true, and 0 otherwise.

The other important constraint we must remember is simply that the flow on an arc must not exceed the capacity. Putting this together, we have the following formulation of the maximum flow problem:

$$\begin{array}{lll} \max & z & \\ \text{subject to} & \displaystyle\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} - (\mathbf{1}_{i=s} - \mathbf{1}_{i=t})z = 0 & \forall i \in V \qquad (1) \\ & x_{ij} \le u(i,j) & \forall (i,j) \in A \\ & z, x \ge 0. & \end{array}$$

This already tells us that we can solve the maximum flow problem using the simplex method. There are, however, faster algorithms for solving this problem, and this is described in the next section.

We can formalize the claim that the value of any flow is at most the capacity of any cut.

Lemma 1. *Given a flow x of value z and an $s-t$-cut S, we have $z \le \text{cap}(S)$.*

Proof. Given x and S, we obtain by summing (1) over $i \in S$:

$$z = \sum_{(i,j)\in A: i\in S, j\notin S} x_{ij} - \sum_{(j,i)\in A: j\notin S, i\in S} x_{ji}. \qquad (2)$$

Using the fact that $x_{ij} \le u(i,j)$ for the arcs in the first summation, and $x_{ji} \ge 0$ for the arcs in the second summation, we obtain:

$$z \le \sum_{(i,j)\in A: i\in S, j\notin S} u(i,j),$$

showing the claim as the last term is precisely cap(S). □

3 Augmenting path algorithm

A natural algorithm for the maximum flow problem is to start with the zero flow (no flow on any arc) and to repeatedly *push* more flow from the source to the sink until we are unable to push more flow. The notion of *pushing* flow needs, of course, to be formalized.

Suppose we have a flow x. If there exists an arc (s, v_1) with $x_{sv_1} < u(s, v_1)$ then we can increase the flow along this arc say by $\epsilon > 0$ where $\epsilon \le u(s, v_1) - x_{sv_1}$, thereby increasing the net flow leaving s by ϵ. However, flow conservation at v_1 is not satisfied any more and to repair it, we could increase the flow by ϵ on some arc $(v_1, v_2) \in A$ with $x_{v_1v_2} < u(v_1, v_2)$ provided ϵ is small enough. Another option to reestablish flow conservation at v_1 would be to *decrease* the flow on an arc $(v_2, v_1) \in A$ with $x_{v_2v_1} > 0$ by ϵ. We have now flow conservation back at v_1, but we have again destroyed flow conservation at v_2. We can thus proceed similarly and find either an arc $(v_2, v_3) \in A$ on which we can increase flow by ϵ or an arc $(v_3, v_2) \in P$ on which we can decrease flow by ϵ. This operation of pushing flow would be successful if we can eventually reach vertex t, as we do not need to maintain flow conservation at the sink.

To formalize this operation of flow augmentation, it is convenient to define an auxiliary directed graph G_x, called the *residual network*, and which depends on the current flow x. The graph G_x has the same vertex set as G, and its edge set is

$$A_x = \{(i,j) : (i,j) \in A, x_{ij} < u_{ij}\} \cup \{(i,j) : (j,i) \in A, x_{ji} > 0\}.$$

The edges in the first set in this definition are sometimes called the forward arcs and those in the second set the backard arcs. This means that for any $(i,j) \in A_x$, we can either increase the flow on (i,j) or decrease the flow on (j,i). In both cases, the net flow out of i increases while the net flow out of j correspondingly decreases. For each arc (i,j) in A_x, we define a residual capacity $u_x(i,j)$ by

$$u_x(i,j) = \begin{cases} u(i,j) - x_{ij} & (i,j) \in A : x_{ij} < u(i,j) \\ x_{ji} & (j,i) \in A : x_{ji} > 0. \end{cases}$$

An *augmenting path* P with respect to a flow x is defined as a directed path from s to t in the residual graph G_x. Given an augmenting path P we can modify the flow along P, by increasing the flow by $\epsilon(P)$ on forward edges and decreasing it by $\epsilon(P)$ on backward where $\epsilon(P) = \min_{(i,j)\in P} u_x(i,j)$. This choice of ϵ corresponds to the largest value such that the resulting flow satisfies the capacity and nonnegativity constraints.

The *augmenting path* algorithm for the maximum flow problem is very simple to state. Start with the zero flow $x = 0$. While there exists an augmenting path P in the residual graph G_x corresponding to the current flow x, push flow along x. In the case that several augmenting paths exist, we have not specified which one to select. However, independently of which augmenting path is selected, we claim that this algorithm terminates and outputs a maximum flow, and this is stated in the following theorem. We should emphasize, however, that this assumes that the capacities are all integers (or rational numbers). In the case of potentially irrational capacities, an unfortunate choice of augmenting path may lead to the algorithm never terminating, and the flow value converging in the limit to a non-optimal value. When a flow has integer values for every arc, we say that the flow is *integral*.

Theorem 2. *Given a maximum flow problem with integer capacities, the augmenting path algorithm terminates with a flow x^* that is both maximal and integral. Furthermore, there exists a cut S separating s and t whose capacity equals the value of the maximum flow x^*.*

Proof. When the capacities are all integral, we first claim that the flow x will always be integral during the execution of the algorithm. This is shown by induction. Indeed, the algorithm starts with an integral flow ($x_a = 0$ for every $a \in A$), and so the base case is satisfied. Now, assuming that the current flow x is integral, we observe that all residual capacities are integral, and therefore $\epsilon(P)$ is also an integer. Therefore, the flow obtained after pushing $\epsilon(P)$ units along P remains integral.

This implies that the algorithm terminates. Indeed, at every iteration $\epsilon(P)$ is a non-zero integer, and therefore is at least 1. This means that the net flow out of s increases by at least one unit at every iteration, and since the flow value of any flow is bounded by the capacity of any cut, say the cut induced by $\{s\}$, the number of iterations before termination is finite. Let x^* be the integral flow that the algorithm returns.

To show that this flow is a maximum flow, we exhibit a cut separating s from t whose capacity equals the flow value. For this purpose, consider the residual graph G_{x^*} corresponding to this final flow x^*. Define S by

$$S := \{v \in V : \text{there exists a directed path from } s \text{ to } v \text{ in } G_{x^*}\}.$$

Obviously $s \in S$ and, since there are no augmenting paths in G_{x^*} (because of termination of the algorithm), t is not reachable from s and thus $t \notin S$. Thus S induces a cut separating s from t. By definition of S, there are no arcs between S and $V \setminus S$ in the residual graph G_{A^*}. This means that $x^*_{ij} = u(i,j)$ for $(i,j) \in A$, $i \in S$, $j \notin S$, and $x^*_{ji} = 0$ for $(j,i) \in A$, $j \notin S$, $i \in S$. This observation together with (2) imply that

$$z^* = \sum_{(i,j)\in A: i\in S, j\notin S} x^*_{ij} - \sum_{(j,i)\in A: j\notin S, i\in S} x^*_{ji} = \sum_{(i,j)\in A: i\in S, j\notin S} u_{ij} = \text{cap}(S).$$

This means that both x^* is a maximum flow and S gives a mimum $s - t$ cut. □

4 Dual

In this section, we provide an alternate proof of the maximum flow minimum cut theorem given in Theorem 1 using linear programming duality.

The first step step is to write the dual. One way to do this is to write the LP in matrix form, replacing equality constraints by pairs of inequalities (since the constraint $a^T x = b$ is the same as the pair of constraints $a^T x \le b$, $a^T x \ge b$). This is certainly doable, but a little painful. If you work with linear programs enough, you get quite good at taking the dual; there are some discussion about shortcuts in your linear programming notes. If you consult those, you should be able to determine that the following is indeed the dual:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} u(i,j)\alpha_{ij} \\ \text{subject to} \quad & \beta_t - \beta_s \ge 1 \\ & \alpha_{ij} + \beta_i - \beta_j \ge 0 \qquad \forall (i,j) \in A \\ & \alpha_{ij} \ge 0 \qquad \forall (i,j) \in A \\ & \beta_i \in \mathbb{R} \qquad \forall i \in V. \end{aligned}$$

So far, we see no sign of cuts; somehow we need to massage our dual if we're going to get our max-flow min-cut theorem.

First, notice that for a fixed $(i,j) \in A$, the only constraints on α_{ij} are that $\alpha_{ij} \ge \beta_j - \beta_i$, and $\alpha_{ij} \ge 0$. Moreover α_{ij} appears with a nonnegative coefficient in the minimization objective. So the only reason that we would pick $\alpha_{ij} > \beta_j - \beta_i$ in an optimal solution is if $\beta_j - \beta_i$. More precisely, we may take

$$\alpha_{ij} = \max(\beta_j - \beta_i, 0).$$

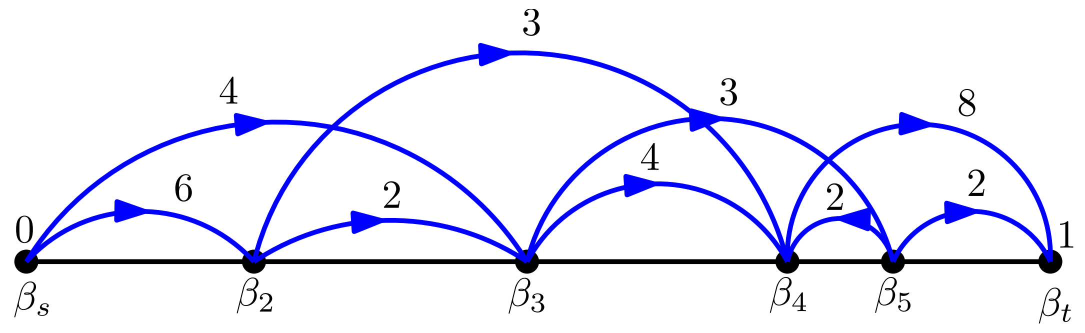
Let's use the convenient notation $(Z)^+ := \max(Z, 0)$. So we can rewrite α_{ij} as $(\beta_j - \beta_i)^+$.

Next, observe that just by shifting all the β_i's by the same amount, we can obtain an equivalent solution with $\beta_s = 0$, so let's assume this from now on. So $\beta_t \geq 1$; but in fact we can assume that $\beta_i \leq 1$ for all $i \in V$ (meaning that $\beta_t = 1$); for consider the replacement

$$\beta'_i = \min(\beta_i, 1) \qquad \forall i \in V;$$

then for any i, j such that $\beta_j > \beta_i$, we have $\beta'_j - \beta'_i \leq \beta_j - \beta_i$. Thus the objective value of the solution β' is not larger than the objective value of the solution β.

So we can place all the β_i's on the interval $[0, 1]$, with $\beta_s = 0$ and $\beta_t = 1$. Each β_i is associated with a vertex $i \in V$. So we can imagine drawing the graph, in such a way that vertex i is placed at position β_i:



Now we can interpret the objective nicely in this picture. Any arc that is going from right to left contributes nothing to the dual objective. An arc (i, j) going from left to right contributes an amount equal to the product of its capacity $u(i, j)$ and its length $\beta_j - \beta_i$. Thus, independently of whether the arc (i, j) goes to the right or to the left, it contributes $u(i, j)\alpha_{ij} = u(i, j)(\beta_j - \beta_i)^+$ to the dual objective function.

Observe that any $s - t$-cut S corresponds to a solution to the dual in which β_i is 0 for $i \in S$ and 1 for $i \notin S$. But the dual contains many solutions β that do not correspond to cuts. Consider any optimum solution β^* in the dual. The β_i^* values might not be all 0 or 1, but we show below that we can extract from it an $s - t$-cut that has the same dual objective function value.

Given an optimum dual solution β^*, define, for each $q \in [0, 1)$, the s-t-cut

$$S(q) := \{i \in V \mid \beta_i^* \leq q\}.$$

(This is certainly an s-t-cut, since $\beta_s^* = 0$ and $\beta_t^* = 1$.) Suppose we generate $q \in [0, 1)$ uniformly at random. We have only seen discrete probability, but this is analogous (one could imagine that we finely discretize the interval $[0, 1)$ and choose any of these values uniformly at random). What is the probability that an arc (i, j) belongs to the cut generated by $S(q)$? Well, it is precisely $\alpha_{ij}^* = (\beta_j^* - \beta_i^*)^+$, and therefore by linearity of expectations, we have that

$$\mathbb{E}(cap(S_q)) = \sum_{(i,j) \in A} u(i, j)\alpha_{ij}^*. \tag{3}$$

Each S_q leads to a feasible solution in the dual and therefore $cap(S_q) \geq \sum_{(i,j)\in A} u(i, j)\alpha_{ij}^*$ by optimality of β^*. But, because of (3), we have that any S_q must have capacity equal to $\mathbb{E}(cap(S_q))$ (since no cut can have capacity less than the optimum dual value). Thus we have shown that any

S_q generated this way has a capacity equal to the optimum dual value, which by strong duality of linear programming, is equal to the maximum flow value. We have thus given another proof of Theorem 1.