Math 3272: Linear Programming[1] Mikhail Lavrov

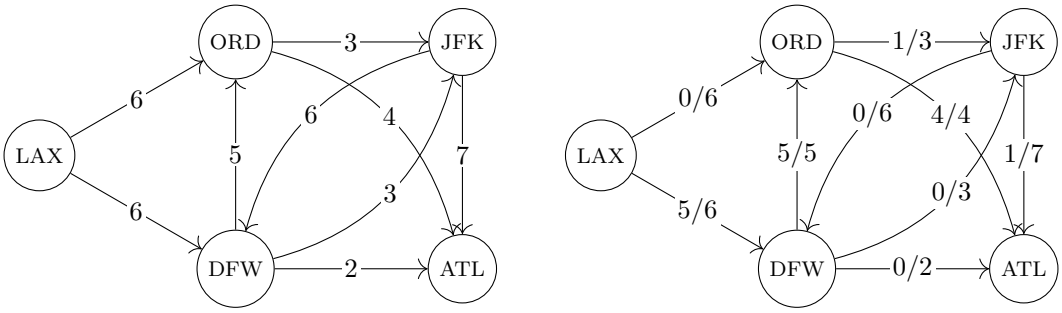
Lecture 20: Network flows and the max-flow problem

October 25, 2022 *Kennesaw State University*

1 Example: a fruit-shipping problem

Suppose you grow oranges in California and want to ship them by airplane to Atlanta. Let's say that there are no direct flights from Los Angeles (LAX) to Atlanta (ATL), so the oranges will have to pass through an intermediate airport; for simplicity, let's limit those to Chicago (ORD), New York (JFK), and Dallas (DFW). On our simplified map of the airports, we will draw arrows representing the possible flights.

Every airplane can carry the same amount of oranges, but the number of flights between the airports is limited. To keep track of that number, we label each arrow from airport to airport with the number of flights we can use to carry oranges going in that direction. This labeling is shown below on the left. On the right, we see one possible (though maybe not very efficient) way that the flights could be used: there are 5 airplanes carrying oranges from LAX to DFW, 5 more carrying those same oranges from DFW to ORD, and then the oranges are split up; 4 airplanes' worth of oranges are taken directly to ATL, and the remaning oranges are shipped through JFK.



We would like to get as many oranges as possible from LAX to ATL per day. How can we formulate this question as a linear program?

Our variables in this problem will have to be the variables we need to specify feasible solutions such as the one we see in the diagram of the right: for every pair of airports connected by flights, we need to know the number of airplanes used to ship oranges from one to the other. For example, we might have a variable $x_{\text{LAX,DFW}}$ to represent how much is being shipped from LAX to DFW; in the diagram, $x_{\text{LAX,DFW}} = 5$. Sometimes, there are flights going both ways, which need different variables; for example, we will have separate variables $x_{\text{DFW,JFK}}$ and $x_{\text{JFK,DFW}}$.

What are the constraints on these variables? The most straightforward ones are the ones coming from the numbers in the diagram: for example, $x_{\text{LAX,DFW}} \leq 6$ because we are told that there are only 6 flights from LAX to DFW. (Also, of course, all these variables should be nonnegative.)

[1] This document comes from an archive of the Math 3272 course webpage: http://misha.fish/archive/3272-fall-2022

But if those were all the constraints, then we'd "solve" the problem by setting each variable to its maximum value—after all, why not?

There are some logical problems with doing that. For example, we'd have 12 flights leaving LAX carrying oranges each day, but 13 flights entering ATL with oranges. Where do the extra oranges come from? The problem is that we forgot to add any constraints saying that oranges can't appear out of nowhere or vanish into nowhere.

The "conservation of oranges" constraints will look at every intermediate airport and say: the number of oranges going in should equal the number of oranges going out. For example, at JFK, this constraint would be

$$x_{\text{ORD,JFK}} + x_{\text{DFW,JFK}} = x_{\text{JFK,DFW}} + x_{\text{JFK,ATL}}.$$

We do not have such a constraint at LAX or ATL. There are no oranges that can be shipped into LAX, but that does not mean that $x_{\text{LAX,ORD}} + x_{\text{LAX,DFW}}$ (the number of oranges shipped out of LAX) should be 0: in fact, we want this quantity to be as large as possible! To solve our problem, we can either maximize $x_{\text{LAX,ORD}} + x_{\text{LAX,DFW}}$ or, equivalently, maximize $x_{\text{ORD,ATL}} + x_{\text{JFK,ATL}} + x_{\text{DFW,ATL}}$: the number of oranges arriving in ATL. With the "conservation of oranges" constraint in play, these should be one and the same.

Now we have all the ingredients we need for this linear program: our first example of a **maximum flow problem**.

2 Maximum flow problems

Let's generalize the problem we have described in the previous section.

In general, our problem will be described by a **network**, which includes two pieces of information:

- A set N of **nodes**. In a maximum-flow problem, there are always two special nodes in N: a **source** s and a **sink** t. Our goal is to transport as much stuff from s to t.
- A set A of **arcs**. Each arc is a pair (i, j), where i and j are nodes, and representing the possibility of going from i to j. Each arc (i, j) has a **capacity** c_{ij}. We say that the arc (i, j) starts at i and ends at j; we assume that there are no arcs that start at t or end at s, because they'd be pointless.

To describe what we are doing with the network, we have a variable x_{ij} for every arc $(i, j) \in A$ representing the amount of stuff being moved along arc (i, j). We call x_{ij} the **flow from** i **to** j, and in general we call the vector $\mathbf{x}$ a **flow**. A feasible flow must satisfy the following constraints:

- Nonnegativity constraints: $x_{ij} \geq 0$ for all $(i, j) \in A$.
- Capacity constraints: $x_{ij} \leq c_{ij}$ for all $(i, j) \in A$.
- **Flow conservation** constraints, which take a bit more effort to write down. For every node $k \in N$ other than s and t, we have

$$\sum_{i:(i,k)\in A} x_{ik} = \sum_{j:(k,j)\in A} x_{kj}.$$

Here, "$i : (i,k) \in A$" means "this sum ranges over all nodes i such that (i,k) is an arc". Similarly, the second sum ranges over all nodes j such that (k,j) is an arc.

The meaning of this constraint is that the total flow going into node k is equal to the total flow going out of node k.

Subject to all three sets of constraints, we maximize the total flow out of the source:

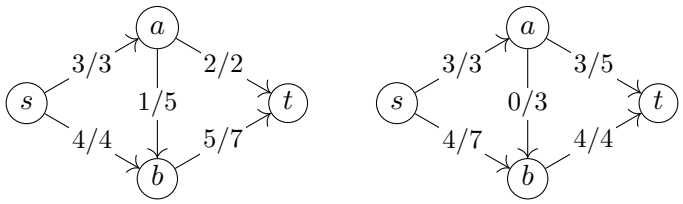
$$\sum_{j:(s,j)\in A} x_{sj}.$$

We call this quantity the **value** of flow $\mathbf{x}$.

3 Cuts and their capacities

We move on to the following question: how can we tell if a feasible flow is optimal?

Consider the first diagram below. Here, just as in the orange shipping problem, we label an arc (i,j) by x_{ij}/c_{ij}: the flow along the arc and the capacity of the arc. Here, the value of the flow 7; we can guarantee that this is best possible, because the total capacity of edges leaving s is 7, so at most 7 flow can leave s.



Now consider the second diagram. Here, things are a bit more complicated. However, we can still see a "bottleneck" in the flow if we think of splitting up the nodes into $\{s,b\}$ and $\{a,t\}$. The total flow going from $\{s,b\}$ to $\{a,t\}$ is 7 (which is still the value of the flow). This cannot be increased, because all edges from s or b to a or t (namely, (s,a) and (b,t)) have the maximum flow possible, and all edges going the other way (namely, (a,b)) have zero flow.

The generalization of this notion is a **cut**. A cut in a network is a partition of the node set N into two sets, S and T, such that $s \in S$ and $t \in T$. (Being a partition requires that $S \cap T = \varnothing$ and $S \cup T = N$: each node is exactly one of the two sets S, T.)

The **capacity** of a cut (S,T) is, informally, the maximum amount of flow that can move from S to T. Formally, it is the sum

$$c(S,T) = \sum_{i\in S}\sum_{j\in T} c_{ij}$$

where we take c_{ij} to be 0 if $(i,j) \notin A$. For example, in the second diagram above, the cut $(\{s,b\},\{a,t\})$ has capacity $c_{sa} + c_{bt} = 7$, because (s,a) and (b,t) are the only two arcs going from $\{s,b\}$ to $\{a,t\}$.

Low-capacity cuts are bottlenecks in the network: if we have a cut (S,T) with capacity $c(S,T)$, then no more than $c(S,T)$ flow can be sent from s to t. This might make intuitive sense, but just in case, let's prove why this happens.

Theorem 1. *If a cut (S,T) has capacity $c(S,T)$, then no more than $c(S,T)$ flow can be sent from s to t in the network.*

Proof. Let $\mathbf{x}$ be a feasible flow in the network, and consider the sum

$$v(\mathbf{x}) := \sum_{k \in S} \left(\sum_{j:(k,j)\in A} x_{kj} - \sum_{i:(i,k)\in A} x_{ik} \right).$$

On one hand, flow conservation tells us that only the $k = s$ term of this sum is allowed to be nonzero. In fact, the $k = s$ term of this sum is equal to the value of $\mathbf{x}$, and therefore the whole sum $v(\mathbf{x})$ simplifies to the value of $\mathbf{x}$.

Now rearrange this sum slightly differently. First, split it up:

$$v(\mathbf{x}) = \sum_{k \in S} \sum_{j:(k,j)\in A} x_{kj} - \sum_{k \in S} \sum_{i:(i,k)\in A} x_{ik}.$$

Now split up each of these sums further: in each sum ranging over all i or all j, consider $i, j \in S$ separately from $i, j \in T$. (To simplify notation, we'll drop the requirement that $(k,j) \in A$ or $(i,k) \in A$: with the convention that the capacity of arcs not in A is 0, this doesn't make a difference.) We get:

$$v(\mathbf{x}) = \left(\sum_{k \in S} \sum_{j \in S} x_{kj} + \sum_{k \in S} \sum_{j \in T} x_{kj} \right) - \left(\sum_{k \in S} \sum_{i \in S} x_{ik} + \sum_{k \in S} \sum_{i \in T} x_{ik} \right).$$

The double sum over $k \in S, j \in S$ cancels with the double sum over $k \in S, i \in S$, because those two sums include the exact same terms, so we have

$$v(\mathbf{x}) = \sum_{k \in S} \sum_{j \in T} x_{kj} - \sum_{k \in S} \sum_{i \in T} x_{ik}.$$

Now we're going to put an upper bound on $v(\mathbf{x})$. For the first sum, we have $x_{kj} \le c_{kj}$ in each term, and so replacing x_{kj} by c_{kj} can only increase the result. For the second sum, we have $x_{ik} \ge 0$ in each term, and we're subtracting all of these terms, so replacing x_{ik} by 0 can also only increase the result. Therefore

$$v(\mathbf{x}) \le \sum_{k \in S} \sum_{j \in T} c_{kj} - \sum_{k \in S} \sum_{i \in T} 0.$$

But this is precisely the definition (with slightly different summation variables) of the capacity $c(S,T)$. Therefore $v(\mathbf{x}) \le c(S,T)$, which is the inequality we wanted. □

4 Cuts and linear programming duality

Cuts in a network are a way to put an upper bound on the value of a feasible flow. We already have one way to do that for any linear program, not just this one: the dual program. Can we learn to think of cuts in terms of a linear programming dual of the maximum flow program?

The answer is that the two are *almost* the same; we'll deal with the differences over the next few lectures. Here is a pair of dual linear programs we will look at more closely:

$$\textbf{(P)} \quad \begin{cases} \underset{v,\mathbf{x}}{\text{maximize}} & v & & & \\ \text{subject to} & v - \sum_{j:(s,j)\in A} x_{sj} = 0 & & & (u_s) \\ & \sum_{i:(i,k)\in A} x_{ik} - \sum_{j:(k,j)\in A} x_{kj} = 0 & \text{for } k \in N, k \neq s,t & & (u_k) \\ & \sum_{i:(i,t)\in A} x_{it} - v' = 0 & & & (u_t) \\ & x_{ij} \leq c_{ij} & \text{for } (i,j) \in A & & (y_{ij}) \\ & \mathbf{x} \geq \mathbf{0} & & & \end{cases}$$

$$\textbf{(D)} \quad \begin{cases} \underset{\mathbf{u},\mathbf{y}}{\text{minimize}} & \sum_{(i,j)\in A} c_{ij} y_{ij} & & \\ \text{subject to} & -u_i + u_j + y_{ij} \geq 0 & \text{for } (i,j) \in A & (x_{ij}) \\ & u_s = 1 & & (v) \\ & u_t = 0 & & (v') \\ & \mathbf{y} \geq \mathbf{0} & & \end{cases}$$

Here, **(P)** is essentially the same as our maximum-flow linear program. To make the dual look nicer, we've added a dummy variable v that stands in for the objective function, and a dummy variable v' that does nothing (but should be equal to v in any feasible solution). This means that we have constraints for s and for t that look a bit like the flow conservation constraint.

Meanwhile, if we stare at **(D)** long enough, we will recognize a minimum-cut-like idea to it. There is only one constraint involving each dual variable y_{ij}, which can be rewritten as $y_{ij} \geq u_i - u_j$; however, y_{ij} also must be nonnegative. So we could make the $\mathbf{y}$-variables disappear if we rewrote **(D)** as a minimax program, where we minimize

$$\sum_{(i,j)\in A} c_{ij} \max\{0, u_i - u_j\}.$$

Now imagine that every u_i variable is either 0 or 1. Then $\max\{0, u_i - u_j\}$ is equal to 1 whenever $u_i = 1$ and $u_j = 0$, but it is equal to 0 in all other cases. So the objective function just adds up the capacity c_{ij} for every arc (i,j) such that $u_i = 1$ and $u_j = 0$.

We can interpret such a binary vector $\mathbf{u}$ as a cut (S,T), where $S = \{i \in N : u_i = 1\}$ and $T = \{i \in N : u_i = 0\}$. Helpfully, the two remaining constraints in **(D)** tell us that by this rule, $s \in S$ and $t \in T$. The minimax form of the dual objective function becomes the capacity $c(S,T)$: we *do* get cuts back out of linear programming duality!

Of course, not everything is entirely clear yet. Why should $\mathbf{u}$ necessarily be a binary vector? We can maybe make an argument from optimality tht $0 \leq u_i \leq 1$ for all $i \in N$, since making sure all the u-variables are between u_s and u_t tends to make $\max\{0, u_i - u_j\}$ smaller. I am not spelling this argument out in detail, because it doesn't resolve the bigger mystery: why should every u_i be an integer? More on this later.