Math 3272: Linear Programming[1] Mikhail Lavrov

Lecture 22: The integral flow theorem

November 1, 2022 *Kennesaw State University*

1 Consequences of the Ford–Fulkerson method

In the previous lecture, we saw a method for solving maximum flow problems. Just from the way our method works, there are several things we can deduce about these solutions.

Theorem 1 (Max-flow min-cut theorem)**.** *The maximum value of a flow in any network is equal to the minimum capacity of any cut.*

Proof. When the Ford–Fulkerson method finishes computing the maximum flow in a network, it ends by finding a cut whose capacity is equal to the value of the resulting flow: that's how we know that we've found the optimal solution. □

There are many combinatorial applications of the max-flow min-cut theorem. (Menger's theorem in graph theory is one notable example; it is fairly difficult to prove directly, but can be deduced quickly from this theorem. There are many applications in graph theory, most of which we will not see, just because it would take us too far out of the scope of this class.)

But how do these combinatorial applications work? The flow along an arc is a continuous value, but combinatorial problems often have discrete solutions: for example, we will see some applications where objects are being sorted into categories, and something can't be split halfway between multiple categories. So there is a second theorem that's important to applying maximum-flow problems in such cases:

Theorem 2 (Integral flow theorem)**.** *If all capacities in a network are integers, then the network has an integer maximum flow. (That is, there is a maximum flow* $\mathbf{x}$ *such that for every arc* (i, j)*, the flow* x_{ij} *is an integer.)*

Proof. This result comes from thinking about *how* the Ford–Fulkerson method finds a maximum flow. We repeatedly find an augmenting path, then identify the minimum residual capacity of any of its arcs, and then increase or decrease the flow of those arcs by that residual capacity.

As long as we have an integer flow $\mathbf{x}$, every residual capacity will also be an integer: the residual capacity of any arc in the residual graph is given by either x_{ij} or $c_{ij} - x_{ij}$, and both of those will have integer values. So in the next step, several of the x_{ij}'s will change by an integer value, which means that the next flow will also be an integer flow. That means that when the Ford–Fulkerson method finishes, the values of $\mathbf{x}$ will still all be integers. □

It appears that the linear program for the maximum flow problem is quite special. First, we found out that the dual program has integer optimal solutions (which describe cuts, and not some weird

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

fractional analog of cuts). Now we are seeing that the primal program also has integer optimal solutions!

Although we have already shown that this is true, the first thing we'll do today will be to give another proof of this fact—one that relies on just thinking about the properties of the maximum flow linear program. This is useful to know, because it can guarantee integer solutions to some other problems as well. After that, we will see some applications of maximum flow problems, including ones where the integrality plays a key role.

2 Totally unimodular matrices

Let's back up a bit and consider a general linear program, with constraints $A\mathbf{x} = \mathbf{b}$. We've seen earlier in the semester that a basic solution to this system, where the basic variables are indexed by $\mathcal{B}$ and the nonbasic variables are indexed by $\mathcal{N}$, is given by $\mathbf{x}_\mathcal{B} = (A_\mathcal{B})^{-1}\mathbf{b}$ (with $\mathbf{x}_\mathcal{N} = \mathbf{0}$). What new things can we learn from this?

You might remember the formula for the inverse of a 2×2 matrix:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

The important thing to notice about this formula is that the only number we divide by is $ad - bc$: the determinant of the matrix. This continues for larger matrices; for example, for 3×3 matrices, we have

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}^{-1} = \frac{1}{aei + bfg + cdh - afh - bdi - ceg} \begin{bmatrix} ei - fh & ch - bi & bf - ce \\ fg - di & ai - cg & cd - af \\ dh - eg & bg - ah & ae - bd \end{bmatrix}$$

and the big fraction in front is exactly the determinant of the matrix. In general, the inverse of an $n \times n$ matrix A is equal to $\frac{1}{\det(A)}$ multiplied by another matrix called the **adjugate matrix**, whose entries are polynomial functions of the entries of A.

We do not need to know the details. Just knowing this much gives us a useful corollary:

Theorem 3. *If A is an $n \times n$ matrix with integer entries, then A^{-1} has integer entries if and only if $\det(A) = \pm 1$.*

Proof. If $\det(A) = \pm 1$, then we conclude that A^{-1} has integer entries from formulas like the ones above: we compute the adjugate matrix (which will have integer entries, because we multiply, add, and subtract the entries of A) and divide by $\det(A)$ (which is 1 or -1, so it will not give us any fractional results).

On the other hand, if A and A^{-1} both have integer entries, then $\det(A)$ and $\det(A^{-1})$ must both be integers. But it's always true that $\det(A^{-1}) = \frac{1}{\det(A)}$, because $\det(A) \cdot \det(A^{-1}) = \det(AA^{-1}) = \det(I) = 1$. The only way that both $\det(A)$ and $\frac{1}{\det(A)}$ can be integers is if $\det(A) = \pm 1$. □

We say that an $m \times n$ integer matrix A is **unimodular** if every $m \times m$ submatrix $A_\mathcal{B}$ has $\det(A_\mathcal{B}) \in \{-1, 0, 1\}$. This means that every time we have a basic solution to $A\mathbf{x} = \mathbf{b}$, the inverse matrix

$(A_\mathcal{B})^{-1}$ will have integer entries, by Theorem 3. As a result, when A is totally unimodular and $\mathbf{b}$ is an integer vector, $A\mathbf{x} = \mathbf{b}$ will only have integer basic solutions. (We allow a determinant of 0 to make a more general claim: this corresponds to $m \times m$ submatrices which can never give us basic solutions anyway, since $A_\mathcal{B}$ will not be invertible.)

An integer matrix A is **totally unimodular** if every square submatrix, of any size, has a determinant of -1, 0, or 1. (Note that submatrices don't have to pick consecutive rows or consecutive columns. A valid 3×3 submatrix of a large matrix A might pick the 1^{st}, 3^{rd}, and 6^{th} rows with the 2^{nd}, 9^{th}, and 10^{th} colums.

This definition allows us to generalize to basic solutions of systems like $A\mathbf{x} \leq \mathbf{b}$ (which are really $A\mathbf{x} + I\mathbf{s} = \mathbf{b}$ for some slack variables $\mathbf{s}$). When we take an $m \times m$ submatrix of such a system, if some of the slack variables are basic, then the determinant might be equal to the determinant of a smaller, $k \times k$ submatrix of A.

This is the explanation for integer solutions to network flow problems! It turns out that:

Theorem 4. *In any network, the matrix of flow conservation constraints is totally unimodular.*

Proof. The key to this is that each variable x_{ij} in a flow appears in at most two conservation constraints: once in flow conservation at i, and once in flow conservation at j. (When i or j is s or t, there might be just one constraint.) Moreover, these have opposite coefficients: 1 and -1.

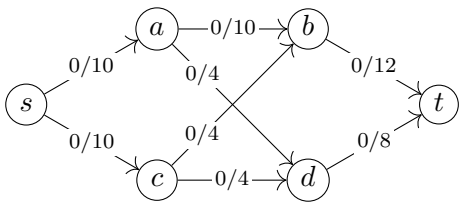
If we take a $k \times k$ submatrix of the flow conservation matrix, one of the following happens happens:

1. We picked the column for a variable x_{ij}, but didn't pick *any* of the rows where x_{ij} has a positive coefficient. Then our submatrix has a column of all zeroes, and the determinant is 0.
2. We picked the column for a variable x_{ij}, but only picked *one* of the rows where x_{ij} has a nonzero coefficient. Then we can do an expansion by minors along x_{ij}'s column, and get a determinant equal to ± 1 times a $(k-1) \times (k-1)$ determinant; repeat this argument for that determinant instead.

   (If k is already 1, we get a determinant of ± 1 equal to our single nonzero entry.)
3. If case 1 and 2 don't occur for any column, then every column has both a 1 and a -1 inside it. Then the rows of our submatrix add up to 0, because the 1 and -1 in every column cancel. This is a linear dependency between the rows, so the determinant is 0.

Since all cases result in a determinant of -1, 0, or 1, the matrix is totally unimodular. □

To help with visualization, here is an example network and the matrix for its flow conservation constraints:



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

The rows correspond to nodes a, b, c, d; the columns, to variables $x_{sa}, x_{sc}, x_{ab}, x_{ad}, x_{bt}, x_{cb}, x_{cd}, x_{dt}$.

Theorem 4 implies the integral flow theorem (we have to consider the capacity constraints as well, but it turns out these do not change much). It also explains why the dual program always has an integer optimal solution (representing a cut).

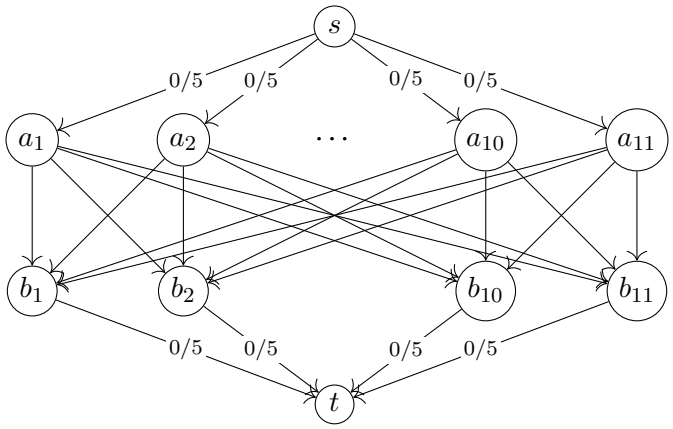
3 Some applications

3.1 Graph factorization

Problem 1. *A chocolate factory has* 11 *employees; conveniently; there are* 11 *industrial processes that they need to be trained in. Not everyone needs to be trained in every process; is it possible for every employee to learn* 5 *processes, such that every process has* 5 *employees that are trained in it?*

Suppose that the first part of the problem is solved. Later, the factory would like 3 *of the employees trained in a process to get a official certification in it. Is it possible for this to happen so that every employee gets certifications in* 3 *of the* 5 *processes they are trained in?*

For the first part of this problem, there are some mathematical constructions with modular arithmetic that show how it can be done; however, we can also set this up as a network flow problem. Draw the following network (where unlabeled arcs (a_i, b_j) should have capacity 1):



Suppose we can find an integer flow in this network with value $5 \cdot 11 = 55$. Then every node a_i receives 5 flow from s, which it must send to 5 different nodes among $b_1, b_2, \ldots, b_{11}$. Meanwhile, each of those nodes must receive a total of 5 flow, which it sends on to t. Now, if we interpret a flow of 1 from a_i to b_j as "Employee i is trained in process j" then we satisfy the conditions in the problem exactly.

How do we know that an integer flow like this exists? The key is that a *fractional* flow like this can be found without any work. As before, send 5 flow from s to every node a_i, and have each node a_i send $\frac{5}{11}$ units of flow to each node $b_1, b_2, \ldots, b_{11}$. As a result, every node b_j receives $\frac{5}{11}$ units of flow from 11 different sources, for a total of 5, which it passes on to t. This is a maximum flow, since we cannot send any more flow out of s, so by the integral flow theorem, there is an integer maximum flow with the same value.

For the second part of the problem, assuming we have solve the first any way we like, redraw the network: keep only the arcs (a_i, b_j) which were used in the integer solution to the first part of the problem. Then, change the capacity on the arcs out of s and into t to 3.

Again, we can find a fractional flow with value $3 \cdot 11$ (the maximum possible) quickly. Node s sends 3 flow to each of $a_1, a_2, \ldots, a_{11}$. Each of them sends $\frac{3}{5}$ flow along each of the 5 arcs leaving it; then each of the nodes $b_1, b_2, \ldots, b_{11}$ receives $\frac{3}{5}$ flow along 5 incoming arcs, for a total of 3, and sends on 3 flow to t. By the integral flow theorem, there is also an integer flow with value $3 \cdot 11$; picking out only the arcs (a_i, b_j) with flow 1, and interpreting them as "Employee i gets a certification in process j", we satisfy the requirements of the problem.

3.2 Consistent matrix rounding

Problem 2. *The students, faculty, and staff at a university vote on whether they prefer coffee or tea, and we get the following percentages among those who voted:*

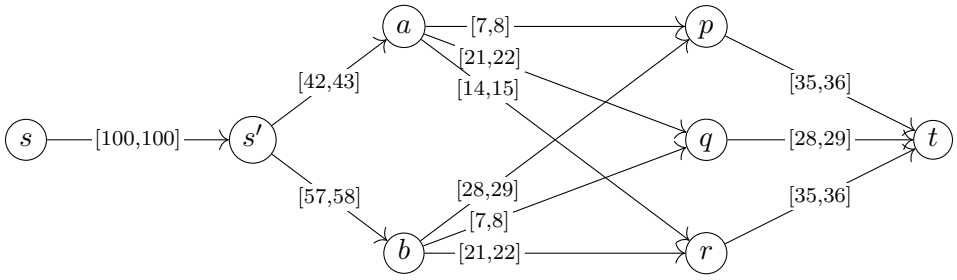
	Students (p)	*Faculty (q)*	*Staff (r)*	*Total*
Coffee (a)	7.143%	21.43%	14.29%	42.86%
Tea (b)	28.57%	7.143%	21.43%	57.14%
Total	35.71%	28.57%	35.71%	100%

Can we round all percentages to integer values so that the row and column totals still make sense?

There are more than just aesthetic reasons why we might want to round data like this. In a scientific report, we want to avoid revealing specific numbers, but as it is, a determined investigator might notice that all of these percentages are approximate multiples of $\frac{1}{14}$. This strongly suggests that there were 14 people total, that exactly one student preferred coffee, and so forth.

It turns out that we achieve the rounding we want if we allow ourselves a tiny bit of flexibility: we can round each percentage in either direction, not just to the closest integer. Moreover, this can be described as a variant of the network flow problem!

The network that represents the problem is given below. Arcs (s', a) and (s', b) correspond to the row sums; arcs (p, t), (q, t), and (r, t) correspond to the row sums; the six intermediate arcs correspond to individual entries of the table. With this setup, flow conservation constraints are exactly the constraints that tell us that row and column sums do what they're supposed to do.



We have not talked about how to solve a network flow problem with lower and upper bounds. It turns out that there are ways to convert this problem to a problem with only capacities, and if we had more time in the semester, we'd absolutely talk about how. For now, let's just prove that this network *must* have an integer flow satisfying all the constraints (and with value exactly 100).

First, there is a feasible flow $\mathbf{x}$ using the exact values that generated our percentages: for example, x_{sa} in this feasible flow is exactly $\frac{6}{14} \cdot 100\%$, the value that created our approximate value of 42.86%. This flow is optimal, so an optimal integer flow exists as well.