• DISCLAIMER: These notes are not necessarily an accurate representation of what I said during the class. They are mostly what I intend to say, and have not been carefully edited.

• Main topics:
  – Maximum Flows
  – Linear program for maxflow
  – Path decomposition of flows

• Last time:
  – Transform combinatorial formulations into LPs.
  – Why $x_u + x_v \geq 1$ for vertex cover.
  – Subscripts too compact.

Our starting point is the shortest path linear program from last time:

\[
\begin{align*}
\text{minimize:} & \quad \sum_e c_e x_e \\
- \sum_{u \rightarrow v} x_{u \rightarrow v} + \sum_{u \rightarrow v} x_{u \rightarrow v} &= \begin{cases} 
1 & \text{if } u = s \\
-1 & \text{if } u = t \\
0 & \text{otherwise}
\end{cases} \\
x_e & \geq 0
\end{align*}
\]

Notice that if we remove the objective, this still tells us something quite interesting: the linear program has a feasible solution if and only if there is a directed $s \rightarrow t$ path.

Instead of asking whether there exists a single path from $s$ to $t$, we can also ask for the existence of multiple paths. In fact, if we limit the edges so that each edge can participate in at most one path, the problem becomes maximizing the number of disjoint paths.

When written as a linear program, we can simply drop the requirement for in/out in $s$ and $t$, and move the amount of flow leaving $s$ into the objective.
maximize:

\[ \sum_{s \to u} x_{s \to u} - \sum_{w \to s} x_{w \to s} \]  \hspace{1cm} (4)

\[ - \sum_{w \to u} x_{w \to u} + \sum_{u \to v} x_{u \to v} = 0 \quad \forall u \neq s, t \]  \hspace{1cm} (5)

\[ x_e \leq 1 \]  \hspace{1cm} (6)

\[ x_e \geq 0 \]  \hspace{1cm} (7)

This leads to the maximum flow problem, which combinatorially can be interpreted as routing the maximum amount of traffic from source \( s \) to sink \( t \).

This problem has a variety of applications other than routing/shipping goods. One particularly important one is the assignment problem, or bipartite matching.

If we have a set of paths, then setting the solution \( x_e = 1 \) if and only if \( e \) is used in one of these paths gives a solution to the linear program. We will do a change of variables and use \( f \) to denote these flows.

This leads to the following natural greedy algorithm for solving this problem:

**GreedyFlow (incorrect)**

1. While there exists path from \( s \) to \( t \)
   
   (a) Route path from \( s \) to \( t \).
   
   (b) Remove path.

2. Return collection of paths routed.

This doesn’t work! Consider the graph with edges

\[ s \to a, a \to t, a \to b, s \to b, b \to t, \]

if we initially pick the path \( s \to a \to b \to t \), it will block off both paths. The issue is that we used the edge \( a \to b \) when we shouldn’t have used it. Note however that ‘undoing’ the flow on it behaves exactly like having an edge in the reverse direction. This observation leads to the Ford-Fulkerson algorithm.

**FordFulkerson**

1. While there exists path from \( s \) to \( t \)
   
   (a) Route path from \( s \) to \( t \).
   
   (b) **Reverse directions of all edges on this path.**

2. Return collection of paths routed.
To show this is correct, we show that once a path does not exist, there exists a cut whose size is exactly the value of the flow leaving \( s \). This cut serves as a ‘bottleneck’, and upper bounds the value of the maximum flow.

The graph with the edges reversed is called the **residual graph**, which we denote with \( G_f \). If there is no \( s \rightarrow t \) path in the residual graph, then there exists a cut \( S \) of size 0 in \( G_f \) such that \( s \in S \) and \( t \notin S \). That is, every edge \( u \rightarrow v \) with \( u \in S \) and \( v \in \overline{S} \) is used in the flow.

The size of the cut in \( G \) is:

\[
\left| \{ u \rightarrow v : u \in S, v \in \overline{S} \} \right| = \sum_{u \in S, v \in \overline{S}} f_{u \rightarrow v}
\]

Also, we must have \( f_{w \rightarrow u} = 0 \) for all \( w \in \overline{S} \) and \( u \in S \), since otherwise \( u \rightarrow w \) must be in the residual graph.

On the other hand, we can sum up the residuals of all vertices in \( S \)

\[
F = \sum_{u \in S} \left( \sum_{u \rightarrow v} f_{u \rightarrow v} - \sum_{w \rightarrow u} f_{w \rightarrow u} \right)
\]

\[
= \sum_{u \in S, v \in \overline{S}} f_{u \rightarrow v} - \sum_{w \in \overline{S}, u \in S} f_{w \rightarrow u} = |\partial_G(S)|.
\]

Here we use \( \partial_G(S) \) to denote the size of the cut formed by \( S \). This proves that maximum flow equals to minimum cut. Minimum cut is also extremely useful in algorithm design: it has a variety of applications in clustering and segmentation.

**PROBABLY WON’T GET TO THIS POINT**

We now go back to the path based interpretation of maximum flow again, and show it formally. To complete the connection to the linear program we need to show that any solution to the linear program can lead to a set of paths.

If we have an integer solution, we can do the following to remove a path: start from source, while there is an edge leaving, follow that edge. Each vertex except \( s \) or \( t \) has in-degree equaling to out-degree. So this path leads back to either \( s \) or \( t \).

If we got to \( t \), we have a path, can remove it and repeat. If we got to \( s \), we just found a cycle. Removing this cycle decreases the amount of flow leaving \( s \) by 1, so this will eventually terminate as well.

So it remains to deal with fractional solutions.

First, note that since the capacities are integral, any cycle formed by edges with fractional flows can be ‘toggled’ to one with integer flow by pushing further flow on it.

This means we can repeat this until there are no cycles in the undirected graph formed by the edges with fractional flow.

Then if there is a fractional edge, we can trace it until we reach \( s \) and \( t \): at any intermediate node there must be another fractional edge since the total net amount entering/leaving is 0 at nodes \( u \neq s, t \). This means there is still a path from \( s \) to \( t \) using edges with fractional flow values. Pushing more flow on it would contradict the assumption of the flow being maximum.