# Notes for Lecture 19

## 1 Network Flows

### 1.1 The problem

Suppose that we are given the network of Figure 1 (top), where the numbers indicate capacities, that is, the amount of flow that can go through the edge in unit time. We wish to find the maximum amount of flow that can go through this network, from $s$ to $t$.

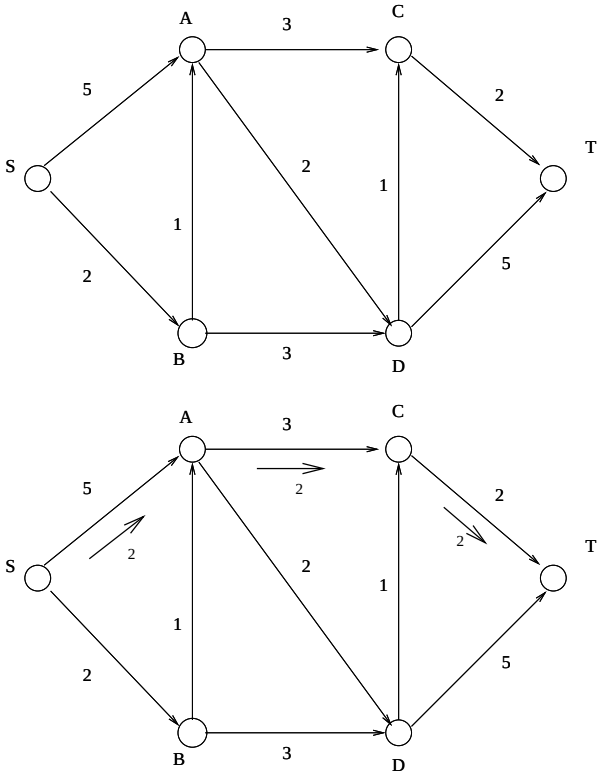



Figure 1: Max flow.

This problem can also be reduced to linear programming. We have a nonnegative variable for each edge, representing the flow through this edge. These variables are denoted $f_{s,a}, f_{s,b}, \ldots$ We have two kinds of constraints:

- *capacity* constraints such as $f_{s,a} \leq 5$ (a total of 9 such constraints, one for each edge), and
- *flow conservation* constraints (one for each node except $s$ and $t$), such as $f_{a,d} + f_{b,d} = f_{d,c} + f_{d,t}$ (a total of 4 such constraints).

We wish to maximize $f_{s,a} + f_{s,b}$, the amount of flow that leaves $s$, subject to these constraints. It is easy to see that this linear program is equivalent to the max-flow problem. The simplex method would correctly solve it.

In general, we are given a "network" that is just a directed graph $G = (V, E)$ with two special vertices $s, t$, such that $s$ has only outgoing edges and $t$ has only incoming edges, and a capacity $c_{u,v}$ associated to each edge $(u, v) \in E$. Then the maximum flow in the network is the solution of the following linear program, having a variable $f_{u,v}$ for every edge.

$$\begin{array}{lrcll} \text{maximize} & \sum_{v:(s,v)\in E} f_{u,v} & & & \\ \text{subject to} & & & & \\ & f_{u,v} & \leq & c_{u,v} & \text{for every } (u,v) \in E \\ & \sum_{u:(u,v)\in E} f_{u,v} - \sum_{w:(v,w)\in E} f_{v,w} & = & 0 & \text{for every } v \in V - \{s,t\} \\ & f_{u,v} & \geq & 0 & \text{for every } (u,v) \in E \end{array}$$

## 1.2 The Ford-Fulkerson Algorithm

If the simplex algorithm is applied to the linear program for the maximum flow problem, it will find the optimal flow in the following way: start from a vertex of the polytope, such as the vertex with $f_{u,v} = 0$ for all edges $(u, v) \in E$, and then proceed to improve the solution (by moving along edges of the polytope from one polytope vertex to another) until we reach the polytope vertex corresponding to the optimal flow.

Let us now try to come up directly with an efficient algorithm for max flow based on the idea of starting from an empty flow and progressively improving it.

How can we find a small improvement in the flow? We can find a path from $s$ to $t$ (say, by depth-first search), and move flow along this path of total value equal to the *minimum* capacity of an edge on the path (we can obviously do no better). This is the first iteration of our algorithm (see the bottom of Figure 1).

How to continue? We can look for another path from $s$ to $t$. Since this time we already partially (or totally) use some of the edges, we should do depth-first search on the edges that have some *residual capacity*, above and beyond the flow they already carry. Thus, the edge $(c, t)$ would be ignored, as if it were not there. The depth-first search would now find the path $s - a - d - t$, and augment the flow by two more units, as shown in the top of Figure 2.

Next, we would again try to find a path from $s$ to $t$. The path is now $s - b - d - t$ (the edges $c - t$ and $a - d$ are full are are therefore ignored), and we augment the flow as shown in the bottom of Figure 2.

Next we would again try to find a path. But since edges $a - d$, $c - t$, and $s - b$ are full, they must be ignored, and therefore depth-first search would fail to find a path, after marking the nodes $s, a, c$ as reachable from $S$. *We then return the flow shown, of value 6, as maximum.*

There is a complication that we have swept under the rug so far: When we do depth-first search looking for a path, we use not only the edges that are not completely full, but we must also traverse *in the opposite direction* all edges that already have some non-zero flow. This would have the effect of cancelling some flow; cancelling may be necessary to achieve optimality, see Figure 1.2. In this figure the only way to augment the current flow is via the

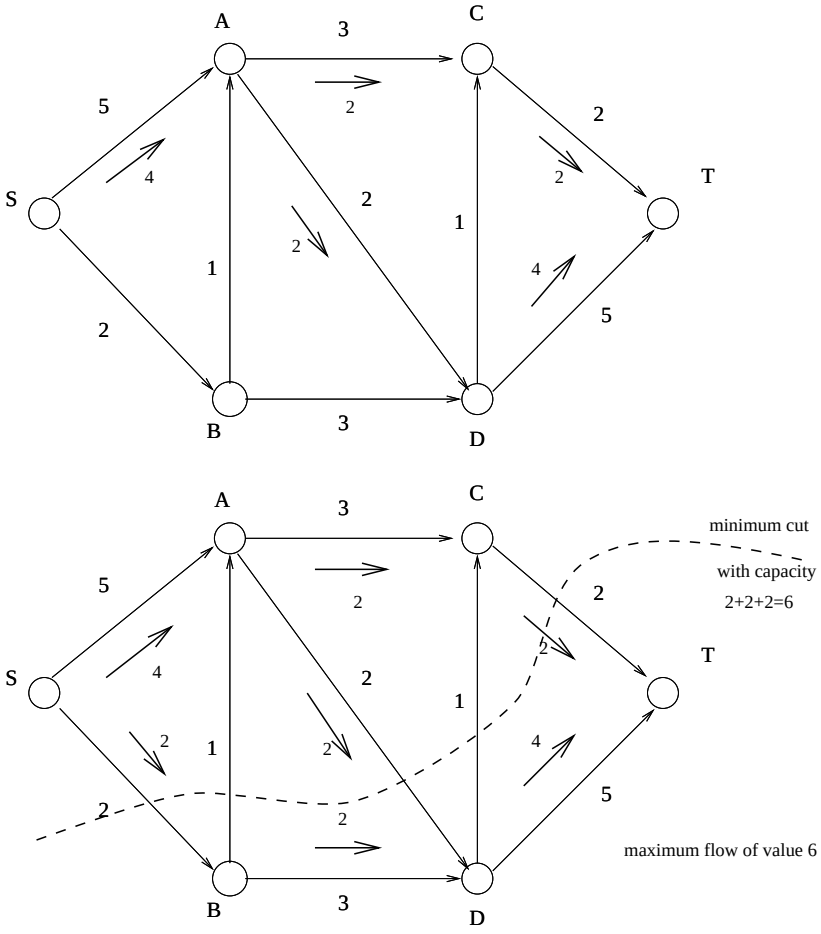
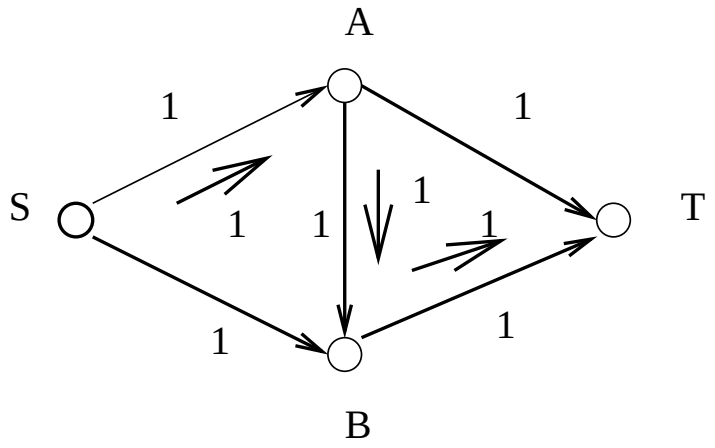



Figure 2: Max flow (continued).

path $s - b - a - t$, which traverses the edge $a - b$ in the reverse direction (a legal traversal, since $a - b$ is carrying non-zero flow).




Flows may have to be cancelled.

The algorithm that we just outlined is the Ford-Fulkerson algorithm for the maximum flow problem. The pseudocode for Ford-Fulkerson is as follows.

1. Given in input $G = (V, E)$, vertices $s$, $t$, and capacities $c_{u,v}$. Initialize $f_{u,v}$ to zero for all edges.
2. Use depth-first search to find a path from $s$ to $t$. If no path is found, return the current flow.
3. Let $c$ be the smallest capacity along the path.
4. For each edge $(u, v)$ in the path, decrease $c_{u,v}$ by $c$, increase $f_{u,v}$ by $c$, and increase $c_{v,u}$ by $c$ (if the edge $(v, u)$ does not exist in $G$, create it). Delete edges with capacity zero.
5. Go to step 2

### 1.3 Analysis of the Ford-Fulkerson Algorithm

How can we be sure that the flow found by the Ford-Fulkerson algorithm is optimal?

Let us say that a set of vertices $S \subseteq V$ is a *cut* in a network if $s \in S$ and $t \notin S$. Let us define the *capacity* of a cut to be $\sum_{u \in S, v \notin S, (u,v) \in E} c_{u,v}$.

We first note that if we fix a flow $f$, and consider a cut $S$ in the network, then the amount of flow that passes "through" the cut, is independent of $S$, and it is, indeed, the cost of the flow.

Lemma 1
*Fix a flow $f$. For every cut $S$ the quantity*

$$\sum_{u \in S, v \notin S, (u,v) \in E} f(u, v) - \sum_{u \in S, v \notin S, (v,u) \in E} f(v, u)$$

*is always the same, independently of $S$.*

As a special case, we have that $\sum_u f(s,u) = \sum_v f(v,t)$, so that the flow coming out of $s$ is exactly the flow getting into $t$.

PROOF: Assume $f_{u,v}$ is defined for every pair $(u,v)$ and $f_{u,v} = 0$ if $(u,v) \notin E$.

$$\begin{aligned}
& \sum_{u\in S, v\notin S} f_{u,v} - \sum_{u\notin S, v\in S} f_{u,v} \\
= & \sum_{u\in S, v\in V} f_{u,v} - \sum_{u\in S, v\in S} f_{u,v} - \sum_{u\notin S, v\in S} f_{u,v} \\
= & \sum_{u\in S, v\in V} f_{u,v} - \sum_{u\in V, v\in S} f_{u,v} \\
= & \sum_{v\in V} f(s,v) + \sum_{u\in S-\{s\}, v\in V} f_{u,v} - \sum_{u\in V, v\in S-\{s\}} f_{u,v} \\
= & \sum_{v\in V} f_{s,v}
\end{aligned}$$

and the last term is independent of $S$. □

A consequence of the previous result is that for every cut $S$ and every flow $f$, the cost of the flow has to be smaller than or equal to the capacity of $S$. This is true because the cost of $f$ is

$$\sum_{u\in S, v\notin S} f_{u,v} - \sum_{v\notin S, u\in S} f_{v,u} \leq \sum_{u\in S, v\notin S} f_{u,v} \leq \sum_{u\in S, v\notin S} c_{u,v}$$

and the right-hand side is exactly the capacity of $S$. Notice how this implies that if we find a cut $S$ and a flow $f$ such that the cost of $f$ equals the capacity of $S$, then it must be the case that $f$ is optimal. This is indeed the way in which we are going to prove the optimality of Ford-Fulkerson.

LEMMA 2

*Let $G = (V,E)$ be a network with source $s$, sink $t$, and capacities $c_{u,v}$, and let $f$ be the flow found by the Ford-Fulkerson algorithm. Let $S$ be the set of vertices that are reachable from $s$ in the* residual network. *Then the capacity of $S$ equals the cost of $f$, and so $f$ is optimal.*

PROOF: First of all, $S$ is a cut: $s$ belongs to $S$ by definition, and it is impossible that $t \in S$, because otherwise $t$ would be reachable from $s$ in the residual network of $f$, and $f$ could not be the output of Ford-Fulkerson.

Now, the cost of the flow is $\sum_{u\in S, v\notin S} f_{u,v} - \sum_{v\notin S, u\in S} f_{v,u}$, while the capacity of the cut is $\sum_{u\in S, v\notin S} c_{u,v}$.

For every $u \in S$ and $v \notin S$, we must have $f_{u,v} = c_{u,v}$, otherwise $v$ would be reachable from $s$ in the residual network (which would contradict the definition of $S$). Also, for every $v \notin S$ and ever $u \in S$, we must have $f_{v,u} = 0$, otherwise the residual network would have an edge with non-zero capacity going from $u$ to $v$, and then $v$ would be reachable from $s$ which is impossible. So the cost of the flow is the same as the capacity of the cut. □

Notice that, along the way, we have proved the following important theorem.

THEOREM 3 (MAX FLOW / MIN CUT THEOREM)

*In every network, the cost of the maximum flow equals the capacity of the minimum cut.*

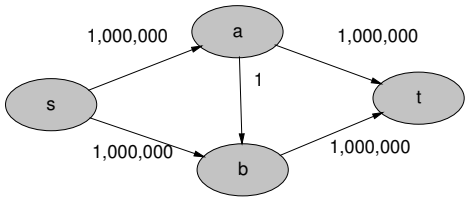



Figure 3: A bad instance for Ford-Fulkerson.

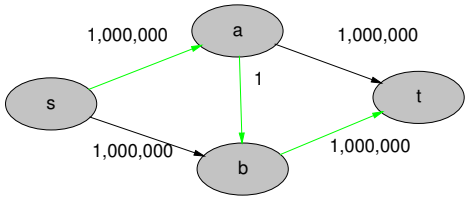



Figure 4: A bad instance for Ford-Fulkerson – Choice of path.

We have established the correctness of the Ford-Fulkerson algorithm. What about the running time? Each step of finding an augmenting path and updating the residual network can be implemented in $O(|V| + |E|)$ time. How many steps there can be in the worst case? This may depend on the values of the capacities, and the actual number of steps may be exponential in the size of the input. Consider the network in Figure 3.

If we choose the augmenting path with the edge of capacity 1, as shown in Figure 4, then after the first step we are left with the residual network of Figure 5. Now you see where this is going.

If step 2 of Ford-Fulkerson is implemented with breadth-first search, then each time we use an augmenting path with a minimal number of edges. This implementation of Ford-Fulkerson is called the Edmonds-Karp algorithm, and Edmond and Karp showed that the number of steps is always at most $O(|V||E|)$, regardless of the values of the capacities, for a total running time of $O(|V||E|(|V| + |E|))$. The expression can be simplified by observing that we are only going to consider the portion of the network that is reachable from $s$, and that contains $O(E)$ vertices, so that an augmenting path can indeed be found in $O(E)$ time, and the total running time can be written as $O(|V||E|^2)$.

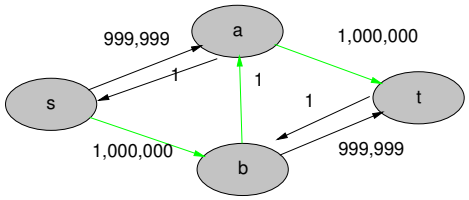



Figure 5: A bad instance for Ford-Fulkerson – Residual network.