601.433/633 Introduction to Algorithms **Lecturer:** Michael Dinitz and Jessica Sorrell
Topic: Max-Flow Min-Cut **Date:** 11/6/25

20.1 Introduction

We are given a directed graph $G = (V, E)$ with positive capacities on the edges, $c : E \to \mathbb{R}^+$. This setup is sometimes called a *flow network*. We will slightly abuse notation and let $c(u, v) = 0$ if $(u, v) \notin E$, since this will let us simplify the notation a bit. We will also usually have two special vertices, s and t. For reasons that will become clear, we usually call s the *source* and t the *sink*. We will be concerned today with two different but very related concepts: *flows* and *cuts*. We'll talk a bit about algorithms, but most of the real algorithmic analysis will happen next lecture.

20.2 Flows

Flows are a pretty intuitive concept: a flow from s to t is basically a way of sending "stuff" from s to t. Famous examples include sewer systems, railroad networks, and many others. More formally, an (s, t)-flow is a function $f : E \to \mathbb{R}_{\geq 0}$ such that

$$\sum_{u:(u,v)\in E} f(u, v) = \sum_{u:(v,u)\in E} f(v, u) \tag{20.2.1}$$

for all vertices $v \in V \setminus \{s, t\}$. We might sometimes abuse notation and say that $f(u, v) = 0$ if $(u, v) \notin E$. These equalities are known as *flow conservation*: at every vertex other than the source and the destination, the total flow in is equal to the total flow out.

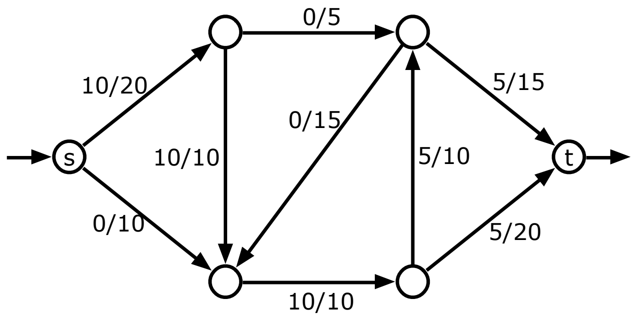
The *value* of the flow, which is sometimes denoted by $|f|$, is the "total amount of stuff" that we're sending from the source to the sink. Since the flow conservation constraints imply that flow in = flow out at all intermediate nodes, the total amount of stuff that we're sending is equal to the total amount of stuff leaving the source, which is equal to the total amount of stuff entering the sink. So

$$|f| = \sum_{u:(s,u)\in E} f(s, u) - \sum_{u:(u,s)\in E} f(u, s) = \sum_{u:(u,t)\in E} f(u, t) - \sum_{u:(t,u)\in E} f(t, u)$$

Of course, our flows have to live within the given capacities. So we also have *capacity constraints* on our flow:

$$f(u, v) \leq c(u, v) \tag{20.2.2}$$

for all $(u, v) \in E$. A flow which satisfies the capacity constraints is sometimes called a *feasible* flow. If $f(u, v) = c(u, v)$ then we say that f *saturates* edge e, and if $f(e) = 0$ we say that f *avoids* edge e. Let's see an example:



An (s,t)-flow with value 10. Each edge is labeled with its flow/capacity.

We are going to be talking about algorithms for computing the *maximum flow*, i.e., a flow f which maximizes $|f|$.

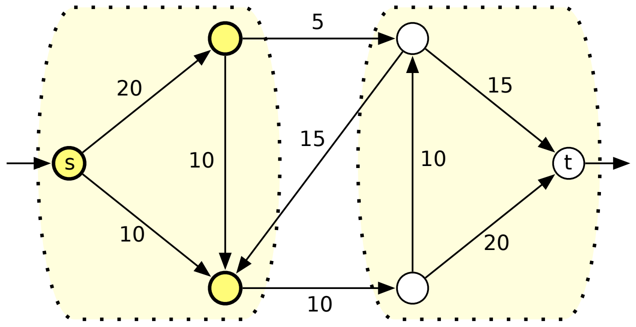
20.3 Cuts

An (s,t)-cut is a partition of the vertices into two sets $(S, \bar{S})$ (recall that $\bar{S} = V \setminus S$) where $s \in S, t \in \bar{S}$. Much of the times we will simply talk about the cut S, since $\bar{S}$ is implied by S. The *capacity* of a cut $(S, \bar{S})$ is the total capacity of the edges from S to $\bar{S}$:

$$cap(S, \bar{S}) = \sum_{(u,v) \in E : u \in S, v \in \bar{S}} c(u,v) = \sum_{u \in S} \sum_{v \in \bar{S}} c(u,v). \tag{20.3.3}$$

Note that the capacity *does not* take into account the capacity of edges from $\bar{S}$ to S. There are other, essentially equivalent definitions of cuts in terms of edge sets, but we're going to mostly stick with this definition.

Let's see another quick example:



An (s,t)-cut with capacity 15. Each edge is labeled with its capacity.

We will mostly be concerned with the *minimum cut* problem, where we try to compute the cut of minimum capacity.

20.4 Max-Flow Min-Cut

It's not hard to see that the minimum cut is at least the maximum flow. Slightly more generally, it's not hard to see that the value of *any* (s,t)-flow is at most the capacity of *any* (s,t) cut. Intuitively, this is because if we have a cut of some capacity α, then since any flow has to "cross" the cut, it is only possible to send α flow. Let's prove this a bit more formally.

Lemma 20.4.1 *Let f be a feasible (s,t)-flow, and let $(S,\bar{S})$ be an (s,t)-cut. Then $|f| \leq cap(S,\bar{S})$.*

Proof:

$$\begin{aligned}
|f| &= \sum_v f(s,v) - \sum_v f(v,s) && \text{(definition)}\\
&= \sum_{u\in S}\left(\sum_v f(u,v) - \sum_v f(v,u)\right) && \text{(flow conservation constraints)}\\
&= \sum_{u\in S}\left(\sum_{v\in\bar{S}} f(u,v) - \sum_{v\in\bar{S}} f(v,u)\right) && \text{(remove terms which cancel)}\\
&\leq \sum_{u\in S}\sum_{v\in\bar{S}} f(u,v) && \text{(flow is nonnegative)}\\
&\leq \sum_{u\in S}\sum_{v\in\bar{S}} c(u,v) && \text{(flow is feasible)}\\
&= cap(S,\bar{S})
\end{aligned}$$

■

This proof actually implies something a little stronger, which we'll use later:

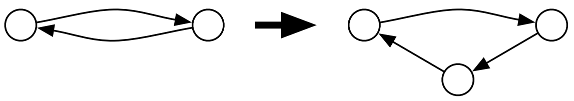
Corollary 20.4.2 *Let f be a feasible (s,t)-flow and let $(S,\bar{S})$ be an (s,t)-cut. If f saturates every edge from S to $\bar{S}$ and avoids every edge from $\bar{S}$ to S, then $|f| = cap(S,\bar{S})$ and f is a maximum flow and $(S,\bar{S})$ is a minimum cut.*

What is not as easy to see is that this upper bound is actually tight: the maximum flow has value *equal* to the capacity of the minimum cut. This is known as the Max-Flow Min-Cut theorem:

Theorem 20.4.3 *In any flow network with source s and sink t, the value of the maximum flow is equal to the capacity of the minimum cut.*

We will spent the rest of the lecture proving this theorem. Note that Lemma 20.4.1 implies that the value of the maximum flow is at most the capacity of the minimum cut.

While it is possible to prove this theorem structurally, we will give a proof which naturally leads to an algorithm (albeit not a very fast algorithm). This approach is due to Ford and Fulkerson, and the resulting algorithm is known as the Ford-Fulkerson algorithm. First, though, we're going to do a simple transformation which will make the notation a little easier. We want to assume that there are no cycles of length 2 in the graph, i.e., that if $(u,v) \in E$ then $(v,u) \notin E$. This is definitely not true in general, but it's not hard to see that we can make it true without losing anything. To do this, we will actually insert a new vertex. If both (u,v) and (v,u) are edges, we will add a new node x and will replace the edge (v, u) with edge (v, x) and (x, u), each of which will have capacity equal to $c(v, u)$. It is not hard to see that this does not change the maximum flow or the minimum cut (good exercise to do at home!).

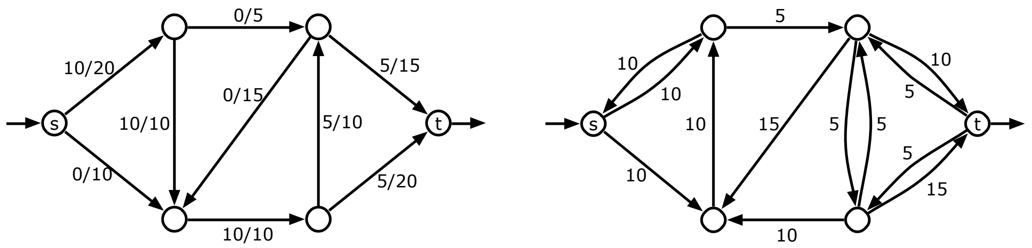


Enforcing the one-direction assumption.

Now let f be a feasible (s, t) flow. We can define what is called the *residual capacities* as follow:

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise} \end{cases}$$

Let's think about these residual capacities for a moment. First, note that since $f(u, v) \leq c(u, v)$ and $f(u, v)$ is nonnegative, all residual capacities are nonnegative. Second, note that there can be a nonzero residual capacity even if there isn't an edge (the second case of the equation). So we will actually define the *residual graph* $G_f = (V, E_f)$, where $(u, v) \in E_f$ if $c_f(u, v) > 0$. Note that in this graph there might actually be cycles of length 2. For example:



A flow f in a weighted graph G and the corresponding residual graph G_f.

We can now prove Theorem 20.4.3 by using this residual graph and Corollary 20.4.2. We break into two cases, and show that the theorem is true in the first case while the second case cannot happen.

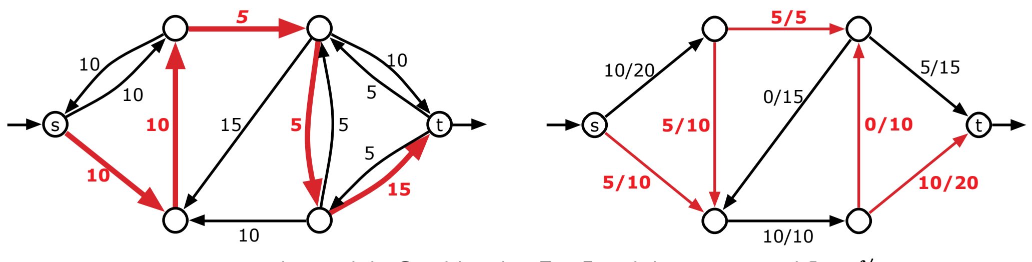
First, suppose that there is no path from s to t in the residual graph G_f. Let S be the vertices reachable from s in G_f. Note that $s \in S$ and $t \notin S$, so $(S, \bar{S})$ is an (s, t)-cut. For every $u \in S$ and $v \in \bar{S}$ we know that $c_f(u, v) = 0$ (or else v would be in S). If $(u, v) \in E$ then this implies that $c_f(u, v) = c(u, v) - f(u, v) = 0$, and hence $f(u, v) = c(u, v)$, so f saturates the edge (u, v). On the other hand, if $(v, u) \in E$, then this implies that $f(v, u) = 0$, and hence f avoids the edge (v, u). So f saturates every edge from S to $\bar{S}$ and avoids every edge from $\bar{S}$ to S. Thus by Corollary 20.4.2 we know that $|f| = cap(S, \bar{S})$ and f is a maximum flow and $(S, \bar{S})$ is a minimum cut. Hence the maximum flow is equal to the minimum cut.

Now suppose that there *is* a path from s to t in G_f. We will try to derive a contradiction to show that this cannot happen. Let this path be $s = v_0, v_1, v_2, \ldots, v_r = t$, and call this path P. Without loss of generality, we may assume that there are no cycles on this path (if there are cycles then we can just find a shorter path without cycles). Such a path (from s to t in the residual graph) is called an *augmenting path*. Let $F = \min_{i=0}^{r-1} c_f(v_i, v_{i+1})$. We are going to claim that we can "push" F more flow from s to t, so f is not actually a maximum flow. This will prove Theorem 20.4.3.

Consider the new flow $f' : E \to \mathbb{R}^{\geq 0}$ defined as follows:

$$f'(u,v) = \begin{cases} f(u,v) + F & \text{if } (u,v) \text{ in } P \\ f(u,v) - F & \text{if } (v,u) \text{ in } P \\ f(u,v) & \text{otherwise} \end{cases}$$

Let's see a quick example:



An augmenting path in G_f with value $F = 5$ and the augmented flow f'.

Note that P leaves the source s once and never comes back (since there are no cycles in P). Thus $|f'| = |f| + F > |f|$. So if we can show that f' is a feasible flow under the original capacities, we will have a contradiction with our assumption that f is a maximum (s,t)-flow.

Let's first show that flow conservation holds in f' at all nodes other than s and t. Consider some node v. If P does not enter or leave v then $f'(u,v) = f(u,v)$ and $f'(v,u) = f(v,u)$ for all u, so since flow conservation held under f, it also holds under f'. Now consider some $v_i \in P$ with $v_i \neq s,t$. Intuitively, since we pushed F new flow through P, whatever change happened to the incoming flow to v_i also happened to the outgoing flow. To prove this more formally, note that the only edges with changed flow are the edge between v_i and v_{i-1}, and the edge between v_i and v_{i+1}. So we just need to prove that the change in flow in these edges balances out. We'll break into four cases depending on the directions of the underlying edges.

1. (v_{i-1}, v_i) and (v_i, v_{i+1}). In this case the edges used by P are edges in G (not just in G_f). Thus when using f' instead of f the total flow into v_i goes up by F, but the total flow out of v_i also goes up by F, and hence flow conservation is maintained.

2. (v_{i-1}, v_i) and (v_{i+1}, v_i). In this case the edge leaving v_i used by P is in G_f but not in G (the reverse edge is in G). Thus $f'(v_{i-1}, v_i) = f(v_{i-1}, v_i) + F$, and $f'(v_{i+1}, v_i) = f(v_{i+1}, v_i) - F$. So the total flow entering and leaving v_i remains the same, so flow conservation is preserved.

3. (v_i, v_{i-1}) and (v_i, v_{i+1}). In this case the edge entering v_i used by P is in G_f but not in G (the reverse edge is in G). Thus, symmetric to the above case, when we push F flow along P to get f' we are actually decreasing by F the amount of flow from v_i to v_{i-1} and increasing by F the amount of flow from v_i to v_{i+1}, so again flow conservation is preserved.

4. (v_i, v_{i-1}) and(v_{i+1}, v_i). In this case both edges used by P are actually reversed edges from G. So when we switched from f to f' we *decreased* the amount of flow entering v_i by F, and also decreased the amount of flow leaving v_i by F.

Now let's show that $0 \leq f'(u,v) \leq c(u,v)$ for all edges $(u,v) \in E$. Consider an edge $(u,v) \in E$. If neither (u,v) nor (v,u) is in the augmenting path P, then $0 \leq f'(u,v) = f(u,v) \leq c(u,v)$ (since f is a feasible flow). Suppose that (u,v) is in P. Then $f'(u,v) = f(u,v) + F \geq f(u,v) \geq 0$, and

$$f'(u,v) = f(u,v) + F \leq f(u,v) + c_f(u,v) = f(u,v) + c(u,v) - f(u,v) = c(u,v).$$

On the other hand, suppose that (v,u) is in P. Then $f'(u,v) = f(u,v) - F \leq f(u,v) \leq c(u,v)$, and

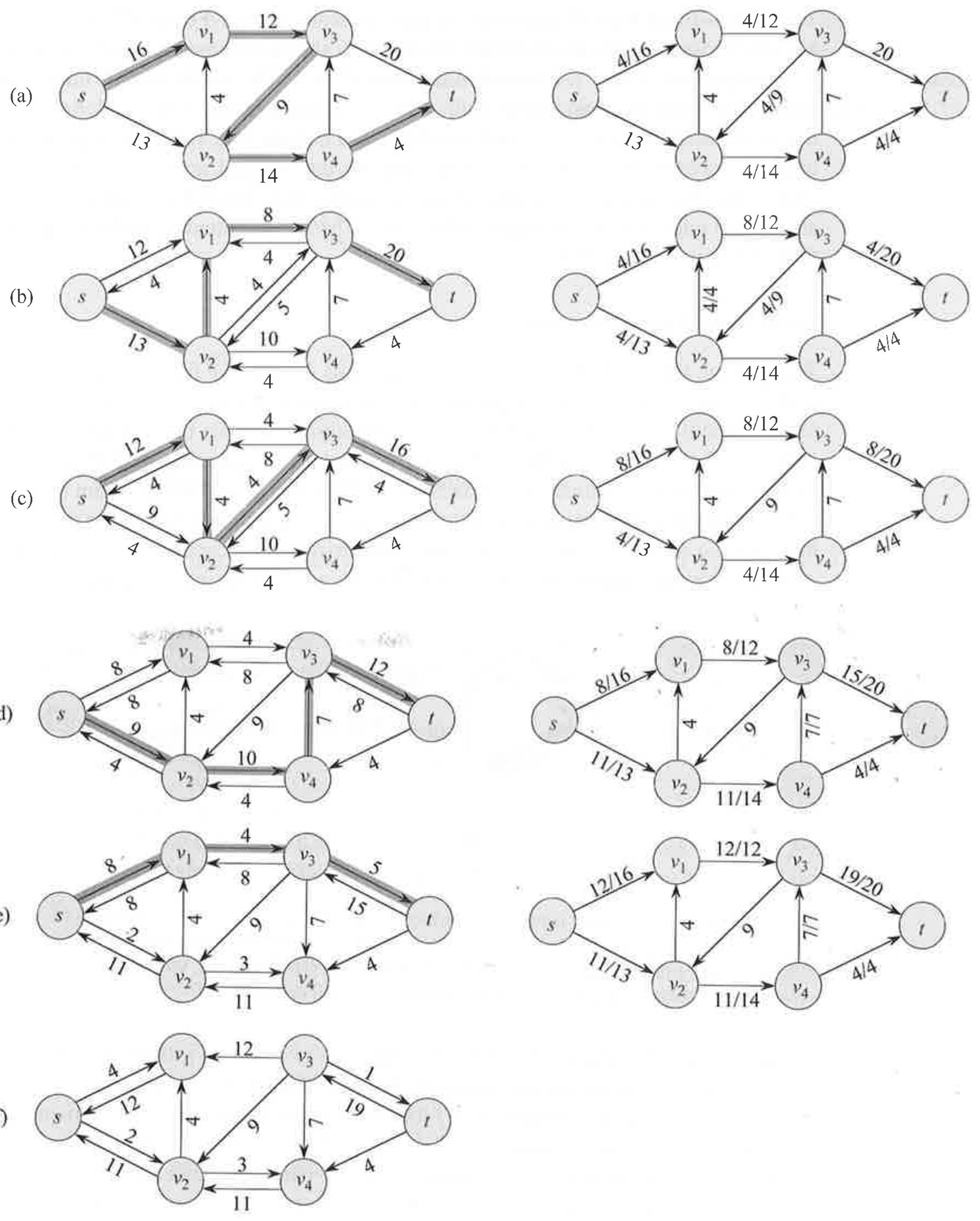
$$f'(u,v) = f(u,v) - F \geq f(u,v) - c_f(v,u) = f(u,v) - f(u,v) = 0$$

Thus we have proved Theorem 20.4.3, the Max-Flow Min-Cut theorem.

20.5 Consequences: Algorithm and Integrality

There are a few interesting consequences of both Theorem 20.4.3 and of our proof of the Theorem. Most importantly, it immediately implies an algorithm for computing a max flow and a min cut. We start with zero flow (so $G_f = G$), find an augmenting path in G_f, and push as much flow as we can along it. This flow now gives a new residual graph, and we just repeat this until there is no longer a path from s to t in G_f. Once this happens, we know from Corollary 20.4.2 that we have found a maximum flow, and from the proof of Theorem 20.4.3 we know that is we let S be the nodes reachable from s in the final G_f, then $(S, \bar{S})$ is a minimum cut. So even though we didn't phrase it algorithmically, our proof of Theorem 20.4.3 actually gives us an algorithm! This is known as the *Ford-Fulkerson algorithm.*

Let's see an example of running this algorithm. This is from the book, but there are an enormous number of examples online – if you don't understand the algorithm, I'd encourage you to look at them.



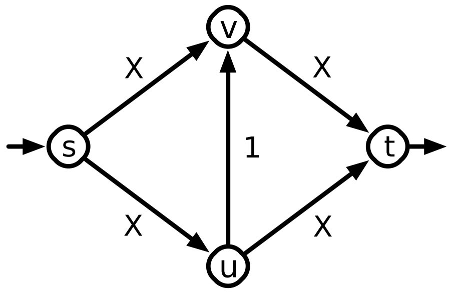
We'll spend most of next lecture actually analyzing the running time of this algorithm (and of some variants), but before we do that, I want to point out a structural feature of this algorithm which actually gives a very weak running time bound. Suppose that all of the capacities in our network are integers. Then it is easy to see by induction that everything stays integral throughout the running of the algorithm. So we actually have the following *integrality theorem*:

Theorem 20.5.1 *If all capacities in a flow network are integers, then there is a maximum flow such that the flow through every edge is an integer.*

Moreover, in this case we also have an upper bound on the running time. If the maximum flow possible is F and all capacities are integers, then in every iteration we push at least 1 unit of flow. So there are at most F iterations. Each iteration simply requires finding a path from s to t in the residual graph, which takes only $O(m+n)$ time by using DFS or BFS. Thus we get the following:

Theorem 20.5.2 *If all capacities in a flow network are integers and the maximum flow value is F, then the running time of Ford-Fulkerson is at most $O(F(m+n))$.*

Unfortunately this can be a large running time – it is not even polynomial in the size of the input. And it can actually occur if we are not smart about how we pick augmenting paths. For example, consider the following flow network, where X is some very large integer:



A bad example for the Ford-Fulkerson algorithm.

If we first push 1 unit of flow along the path $s-u-v-t$, and then push one unit of flow along the augmenting path $s-v-u-t$, etc., then in every iteration we will push exactly one unit of flow so the number of iterations will be $2X$. But the size of the input is at most $O(1)+O(\log X)$, so the running time is exponential in the size of the input.