9
Network Flows

In this chapter we talk about the network flow problem, how to find maximum flows in networks, and how to use network flows to model and solve many different kinds of discrete optimization problems, including those in airline scheduling, baseball elimination, and project selection.

9.1 The Model

Consider a directed graph $G = (V, E)$, where each edge has a capacity $u_e \geq 0$. As always, the graph has n vertices and m edges. We have two special vertices: the source s and the sink t.

An ***s-t flow*** f is a function that assigns a value to each edge in the network such that

1. The flow "respects" edge-capacities: i.e., $0 \leq f(e) \leq u_e$ for every edge.
2. The flow is conserved at each internal node, so that the amount entering equals the amount leaving: formally,
$$\sum_{uv \in E} f(uv) = \sum_{vw \in E} f(vw).$$
3. No flow enters the source s or leaves the sink t (so that for every edge us and every edge tv we have $f(us) = 0$ and $f(tv) = 0$.

The *value* $\text{val}(f)$ of the flow is the total amount of flow on the edges leaving the source, i.e., $\text{val}(F) := \sum_{su} f(su)$. The goal of the MaxFlow problem is to find an s-t flow of maximum value. If the source and sink are clear from context, we drop the s-t words, and just refer to f as a *flow*, and a flow with maximum value as a *maximum flow*.

In this chapter, we have two goals: (a) we show how to solve the MaxFlow problem, and (b) to solve other optimization problems by reducing them to MaxFlow.

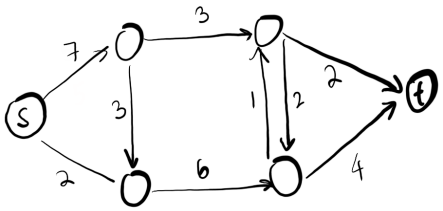

Figure 9.1: A flow network with edge capacities in black.

Imagine sending some incompressive fluid (water/oil) from s to t along these edges (where each edge you can think of as a pipe, whose cross-sectional area is its capacity). The question we ask us: what is the maximum rate at which you can send flow in steady-state?

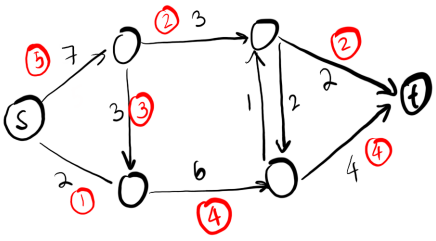

Figure 9.2: Flow values denoted in red. This is a maximum flow in this network.
Show that this value also equals the total flow entering the sink, i.e., $\sum_{vt} f(vt)$.

9.1.1 The Results: MaxFlow

One of the most important results concerns maximum flows in networks with integer capacities.

Theorem 9.1 (Integral Max-Flow). *Given an instance of network flow where the edge capacities are integers, there exist a maximum flow that is integral, that is, where all values $f(e)$ are also integers.*

We do not claim that *each and every* maximum flow in such a network takes on integer values, just that there *exist* maximum flows with this integrality.

How can find such a maximum flow? We will see algorithms later, but let us state the result here.

Theorem 9.2 (Ford-Fulkerson). *Given an instance of network flow where the edge capacities are integers, the Ford-Fulkerson augmenting paths algorithm finds a maximum (integral) flow in time $O(mF^*)$, where F^* is the value of such a maximum flow.*

In some cases, this algorithm may be too slow. In particular, the flow algorithm does not run in time polynomial in the input length. (That's not the only problem—if the edge capacities are not integers, the Ford-Fulkerson algorithm may not terminate.) Thankfully, there are other algorithms.

Theorem 9.3 (Better Algorithms). *Given an instance of network flow where the edge capacities are integers, and the maximum flow value is F, the fattest augmenting path algorithm finds a maximum (integral) flow in time $O(m^2 \log F)$. Given any instance (even with non-integer capacities) the shortest augmenting path algorithm runs in time $O(mn^2)$.*

These are not the fastest algorithms currently known: we now know how to solve maximum flows in time $O(mn^{1/3})$, but we will not be covering these in this course.

9.1.2 Flow Decompositions

One useful concept is the idea of a flow-decomposition: given an s-t flow, a *flow-decomposition* is a collection of directed s-t paths $P_1, P_2, \ldots, P_s$ along with non-negative flow values $\phi_1, \phi_2, \ldots, \phi_s$, such that for each edge uv

$$f(uv) = \sum_{i : uv \in P_i} \phi_i.$$

Moreover, the total flow should be the network is $\text{val}(f) = \sum_i \phi_i$. (See the examples on the right.) Visually, the decomposition shows how the flow f can be represented as the *sum of flows along single paths*.

Theorem 9.4 (Flow Decomposition Theorem). *Any s-t-flow f admits a flow decomposition using at most m paths. Moreover this decomposition can be found in polynomial time. (Finally, if the flow f is integral, then the flow decomposition also assigns integral ϕ_i values for each path.)*

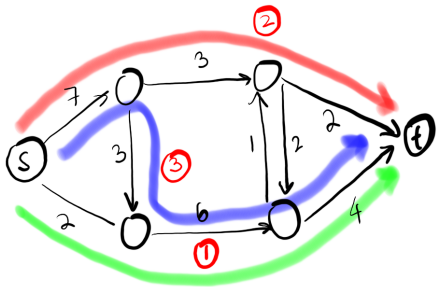

Figure 9.3: A flow decomposition of the above flow. Note that the decomposition sends integer amounts of flow on each path.

Proof. Drop all edges that carry zero flow f. Find any s-t path P in this graph. Define ϕ for this path to be $\min_{e \in P} f(e)$, the smallest flow value on any edge of this path. Now define a new flow f' from f by subtracting ϕ on each edge of P: that is, for any edge $e \in E$,

$$f'(e) := f(e) - \phi \cdot \mathbb{1}_{(e \in P)}.$$

(Please check that f' is a flow.) Note that f' is non-zero only on the edges that had non-zero flow in f. Moreover it is zero on the edge on P that had least flow in f. So the number of edges with non-zero flow in f' is less than in f, and we induct on f'. Since there are at most m edges to start off, and finding each path takes $O(m)$ time (using DFS or BFS), we get a runtime of $O(m^2)$. □

9.1.3 *Flows and Cuts*

A concept that is "dual" to an s-t-flow is an ***s-t-cut***. Such a cut is a subset C of edges which intersects all the s-t paths; that is, deleting these edges means there are no remaining s-t paths. The *value/capacity* of this cut C is $\text{val}(C) := \sum_{e \in C} u_e$, the sum of capacities of edges in C. Note that the entire set of edges E is clearly a cut, and the object of interest will be a cut of smallest capacity.

Claim 9.5. Given any flow f and any cut C,

$$\text{val}(f) \leq \text{val}(C).$$

Proof. Consider the flow f and its flow decomposition, say using paths $P_1, P_2, \ldots, P_s$. For each path P_i there is at least one edge $e_i \in C$ that belongs to P_i. Since the flow ϕ_i on the path P_i is at most the capacity of any edge on it, $\phi_i \leq u_{e_i}$. Hence,

$$\text{val}(f) = \sum_i \phi_i \leq \sum_i u_{e_i} \leq \text{val}(C).$$ □

This means, every cut gives an upper bound on the possible max-flow value. The next (famous) theorem says that there always exists a cut whose value *equals* the value of the max-flow.

Theorem 9.6 (MaxFlow/MinCut Theorem)**.** *Given any network flow instance, there exist an s-t flow f^* and an s-t cut C^*, such that*

$$\text{val}(f^*) = \text{val}(C^*).$$

Moreover, if all the edge capacities are integers, we can find f^* that is an integral flow.

Hence, if f^* is purported to be a max-flow then we can "prove" it is a max-flow by showing a cut whose value matches that of the

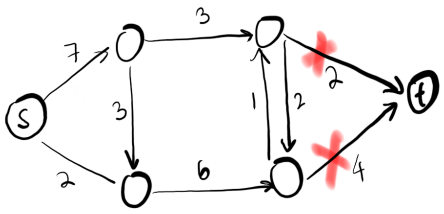

Figure 9.4: A minimum cut in our example.

As a thought experiment, suppose someone gave you a tree and claimed it was an MST. How would you quickly prove it? Or the same question, but with a path P that was claimed to be a shortest path?

flow. And in a dual way, we can show a purported cut C^* is indeed a minimum cut by showing that there is a flow whose value matches that of the cut. In a very strong sense, the objects *s-t* flows and *s-t* cut complement each other: we say they are *dual objects*. As we will see later, the notion of duality will be central in linear programming. (And indeed the above theorem will be a special case of the notion of *strong duality* of linear programs.

9.2 *Applications*