10 Max-Flow Min-Cut

10.1 Flows and Capacitated Graphs

Previously, we considered weighted graphs. In this setting, it was natural to thinking about minimizing the weight of a given path. In fact, we considered algorithms that calculate the minimum weight paths on graphs and we optimized for a variety of parameters (considering all-paths, negative weight edges, etc.). We now are interested in a related problem where our edges have a max capacity and we want to send as much 'flow' from one location to another.

Flow problems are a vary natural set of problems to consider for any graph. Consider the graph of the internet for example; each edge (a connection between two computers, servers, etc.) has a max bandwidth that it allows. We could be interested in calculating what the bottleneck step in sending packets from computer a to b is. For this, we would most likely frame the problem as a flow problem.

Definition 10.1 (Capacitated Network)**.** A capacitated network is $G = (V, C, s, t)$ where V is a finite set of vertices, $s, t \in V$ (colloquially referred to as the source and sink, respectively) and $C : V^2 \to \mathbb{R}^+$, a capacity function defined on all the edges.[1]

Definition 10.2 (s-t flow)**.** A s-t flow is a function $f : V^2 \to \mathbb{R}$ satisfying three conditions:

1. Skew symmetry: for all $u, v \in V$, $f(u, v) = -f(v, u)$
2. Flow conservation: for all interior vertices $u \in V - \{s, t\}$, $\sum_v f(u, v) = 0$
3. Capacity constraints, for all $u, v \in V$, $f(u, v) \leq C(u, v)$.

In layman's terms, the first condition gives the flow a direction per edge. The second condition forces that the only location that flow can be generated from and removed from are the defined source and sink vertices. The third condition forces that none of the edges are over capacity.

The natural question that we want to ask is how much flow is being sent across this network? To answer, this we need to define what a cut is and what the flow across a cut is.

Definition 10.3 (s-t Cut)**.** A s-t cut is a set $S \subseteq V$ such that $s \in S$ and $t \in S^c = V - S$.

Definition 10.4 (s-t Cut Capacity and Flow)**.** The capacity of a cut S is

$$C(S, S^C) = \sum_{u \in S, v \in S^c} C(u, v). \tag{10.1}$$

[1] Here, we wrote C as a function on all vertex pairs. If you wish to think of this in the traditional graph sense then you could choose $E = \{e \in C : C(e) > 0\}$.

And the flow across the cut is

$$f(S, S^C) = \sum_{u \in S, v \in S^c} f(u, v). \tag{10.2}$$

Definition 10.5 (Value of a flow f)**.** The value of a flow f is $\text{Val}(f)$ is

$$\text{Val}(f) = f(\{s\}, \{s\}^c) = \sum_v f(s, v) \tag{10.3}$$

By induction on the set S, one can prove that $\text{Val}(f) = f(S, S^c)$ for any s-t cut S.

10.2 Theorem

Corollary 10.6. *For any flow f and s-t cut S,* $\text{Val}(f) \leq C(S, S^c)$.

Proof. (Sketch.) Apply capacity constraints to the alternate definition of $\text{Val}(f)$ for any cut S. □

We now reach the most interesting statement about flows and cuts; (the previous corollary was the weak duality version). This is the strong duality.

Theorem 10.7 (Max-Flow Min-Cut)**.** *For any capacitated network $G = (V, C, s, t)$,*

$$\max_{\text{flow } f} \text{Val}(f) = \min_{\text{cut } S} C(S, S^c) \tag{10.4}$$

To prove Theorem 10.7, we will need a few additional definitions.

Definition 10.8. For a s-t flow f on a capacitated network, we can define the residual capacity function $C_f : V^2 \to \mathbb{R}^+$ as

$$C_f(u, v) = C(u, v) - f(u, v) \tag{10.5}$$

The residual capacity function is always positive due to capacity constraints on f. It can be thought of as the maximum additional flow that can be pushed across (u, v). We say that f *saturates* (u, v) if $C_f(u, v) = 0$. The residual network is defined as (V, C_f, s, t).

Lemma 10.9. *The following statements hold about flows:*

1. *f_2 is a flow in G_{f_1} iff $f_1 + f_2$ is a flow in G.*
2. $\text{Val}(f_1 + f_2) = \text{Val}(f_1) + \text{Val}(f_2)$.
3. *F_2 is a max flow in G_{f_1} iff $f_1 + f_2$ is a max flow in G.*

4. *If $f^\star$ is a max flow in G and f is any flow in G, then the value of a max flow in G_f is* $\text{Val}(f^\star) - \text{Val}(f)$.

Proof. For (1), by capacity constraints on $G_f(1)$, $f_2(e) \leq C(e) - f_1(e)$. Therefore, $f_1(e) + f_2(e) \leq C(e)$. The other direction follws identically. For (2), notice that $\text{Val}(\cdot)$ is a sum of linear functions. (3) follows from (1) and (2) and (4) follows from (2) and (3). □

The power of this lemma is that flow can be added incrementally until the maximum flow is achieved. This gives us the intuition for *augmenting paths*.

Definition 10.10 (Augmenting Path)**.** Given G, f, an augmenting path for f in G is a path $s \to t$ consisting of edges in the support of G_f (edges with positive capacity in G_f).

Definition 10.11 (Bottleneck Capacity)**.** Given G, f, we define the bottleneck capacity of an s-t path as the least residual capacity of its edges.

Therefore, a path is an augmenting path iff it has non-negative bottleneck capacity. We restate the max-flow min-cut theorem in the following alternate form:

Theorem 10.12 (Alternate Max-Flow Min-Cut Theorem)**.** *The following are equivalent:*

1. *There is an s-t cut S with $C(S, S^c) = \text{Val}(f)$.*
2. *f is a max flow in G.*
3. *There is no augmenting path for f in G.*

Proof. (1) implies (2) is proven by weak duality (Corollary 10.6). To show (2) implies (3), suppose there is an augmenting path p despite f being a max flow. Then we can add flow along the path equal to the bottleneck capacity of p, contradicting the maximality of f. Therefore, no augmenting path exists. To show (3) implies (1), assume there exists no augmenting paths. Define S as the set of vertices reachable by s in G_f. Here reachability is defined in the sense that $E = \{(u, v) : c_f(u, v) > 0\}$. Note that $t \notin S$ because if it were then there is an augmenting path $s \rightsquigarrow t$. Furthermore any edge between a vertex in S and S^c must be at full capacity. Then $C(S, S^c) = f(S, S^c)$, proving maximality. □

10.3 Floyd-Fulkerson Algorithm

This begs the natural question, how do we calculate the max-flow efficiently? Two algorithms are generally used: Ford-Fulkerson and Edmonds-Karp. Naturally, they have different complexities and are therefore useful in different situations. We present both here.

Algorithm 10.13 (Ford-Fulkerson)**.** Given a flow-network $G = (V, C, s, t)$ where we assume $C : V^2 \to \mathbb{Z}^+$ (integer capacities), we initialize a flow $f = 0$. We search for an augmenting path in G_f using a depth-first search. If we find a path, we augment f along the path by the bottleneck capacity and decrease the flow along the 'reverse-path' by the bottleneck capacity. We repeat searching for an augmenting path in G_f until one cannot be found.

Correctness: Correctness follows directly from Lemma 10.9.
Runtime: Each depth-first search costs $O(E)$. Since we assume integer capacities, the bottleneck capacity is at least 1. So at most $\text{Val}(f)$ repetitions of the depth-first search will run. Therefore, the runtime is $O(Ef)$.

It should be noted that Ford-Fulkerson is not guaranteed to terminate if the capacities are irrational.

10.4 Edmonds-Karp Algorithm

The alternative to Ford-Fulkerson is Edmonds-Karp. It is remarkably similar, except instead of performing a depth-first search we perform a breadth-first search and select the shortest path augmenting path. We restate it for consistency.

Algorithm 10.14 (Ford-Fulkerson)**.** Given a flow-network $G = (V, C, s, t)$ where we assume $C : V^2 \to \mathbb{Z}^+$ (integer capacities), we initialize a flow $f = 0$. We search for the shortest augmenting path in G_f using a breadth-first search. If we find a path, we augment f along the path by the bottleneck capacity and decrease the flow along the 'reverse-path' by the bottleneck capacity. We repeat searching for an augmenting path in G_f until one cannot be found.

Correctness: Correctness again follows directly from Lemma 10.9.
Runtime: Each breadth-first search costs $O(E)$. In each augmentation, we saturate at least one of the E edges. Therefore, this can occur at most E times befores the length of the path returned by the breadth-first search must increase. Furthermore, the maximum length of the path returned by the breadth-first search is V. It follows then that at most EV augmentations of the path occur. This yields the runtime $O(VE^2)$.

In general, use Edmonds-Karp over Ford-Fulkerson if $\text{Val}(f) = \Omega(EV)$ or if the capacities are irrational. (If they are rational, you can multiply all of them by a suitable parameter to make them

all integers).