Flows

Albert Ye

March 29, 2023

1 The Problem

We have a directed graph representing a network of pipelines to send stuff through, and we want to send stuff from a **source** s to a **sink** t. Each pipeline has a **capacity**, and there is nowhere to store the stuff en route from s to t. This means that the maximum amount you can ship depends entirely on the capacities of the pipelines.

A **flow** is essentially a copy of the network where the edge weights represent the amount of stuff being passed through this edge. Some units of stuff are being passed from s to t such that everything fits in the pipelines described. Our goal for this problem is to maximize the flow.

The **max flow** is a flow containing the most possible units.

2 Linear Programming

There are a couple of rules to follow for flows:

- $0 \leq f_e \leq c_e, \forall e \in E$.
- Moreover, flow is conserved. $\sum_{(w,u)\in E} f_{wu} = \sum_{(u,v)\in E} f_{uv}$.

The **size** of a flow is the total quantity sent from s to t and, by the conservation principle, is equal to the quantity leaving s: $\text{size}(f) = \sum_{(s,u)\in E} f_{su}$.

We want to satisfy some linear constraints (on what f_{uv} can be for each edge (u, v)) in order to maximize a sum of linear variables. Therefore, the max-flow problem reduces to a linear program.

Therefore, we can base an algorithm off of simplex, which is known as the **Ford-Fulkerson Algorithm.**

Definition 2.1 (Ford-Fulkerson)**.** We start with a zero flow, where nothing is being pushed through any edge. We repeatedly choose a path from s to t that is not saturated and increase flow along that path as much as possible.

2.1 Residual Graph

Sometimes, our algorithm will fail, as it could give us a suboptimal path. The example with vertices s, a, b, t that I'm too lazy to code up shows this.

However, our algorithm is OK if we are allowed to cancel any existing flow from one vertex to another. This is where the **residual graph** comes in.

Definition 2.2 (Residual Graph)**.** Given a graph G and flow f on G, the **residual graph** G_f replaces each edge $w(u,v) = c_{uv}$ (where c_{uv} is the capacity) with two edges, such that $w(u,v) = c_{uv} - f_{uv}$ and $w(v,u) = f_{uv}$.

We fix Ford-Fulkerson over these cases by performing all actions **on the residual graph**.

Definition 2.3 (Ford-Fulkerson, Reworded)**.** We start with a zero flow, where nothing is being pushed through any edge. We repeatedly choose a path from s to t **on the residual graph** that is not saturated and increase flow along that path as much as possible.

3 Other Variations

We can also use this problem to look at min-cut and bipartite matching.

3.1 Min-Cut

Definition 3.1 (s-t cut)**.** An s-t cut is a partition $V = L \cup R$ of the vertices such that $s \in L, t \in R$.

The **capacity** of the cut is $\text{capacity}(L,R) = \sum_{u \to v, u \in L, v \in R} c_{uv}$.

We have something akin to weak duality between the min-cut and max-flow through this theorem:

Theorem 3.2 (Min-Cut Max-Flow)**.** For any flow and any cut (L,R), we see that $\text{size}(f) = \text{capacity}(L,R)$.

Proof. We can split this into $\leq$ and $\geq$ sides. The $\leq$ is done by definition of capacity, as the maximum flow that can be passed down must be at most the capacity of any cut. Otherwise, it would burst some edge, which is not good.

The $\geq$ can be proven by running Ford-Fulkerson on G. If done correctly, there should be no $s \to t$ path in the residual graph G_f. We can set L to be all vertices that can be reached from S in the residual graph, and then R to be all other vertices. Then, we note that there is a cut of edges that the max flow must pass through by definition. Therefore, $\text{size}(f) = \text{capacity}(L,R) \geq \min - \text{cut}$.

Or, just use strong duality! ■

Note that we can also frame the min-cut problem as a linear program in its own right.

3.2 Bipartite Graph Matching

Our input is a bipartite graph $G = (L, R, E)$, where $|L| = |R| = n$.

Our output is a **perfect matching** from L to R, where each node in L is matched to a node in R.

Define all edges from L to R to be directed from L to R with a capacity of 1. We can hook each node in L to a vertex s via an edge with capacity 1, and each node in R to a vertex t via an edge with capacity 1.

Now, we have a maximum possible max-flow from s to t of capacity n. Running Ford-Fulkerson can give us this max-flow, which corresponds to a perfect matching if and only if the amount of flow is n.