Lecture 14: Cuts and Capacity Scaling

Objectives of this lecture

- Learn the definition of a *cut* in a directed graph.
- Maxflow/mincut duality.
- Recognize problems involving cuts.
- Basic capacity scaling algorithm for computing maxflow.

1 Philosophy

This lecture will cover two core points in the theory of flows. The first is a concept that is *dual* to flows, called a cut. A cut consists of edges that goes between a subset of the graph and its complement.

Consider an $s \to t$ maximum flow instance. It turns out that the value of the maximum flow exactly equals the smallest cut separating s from t. This is a specific instance of a more general phenomenon called linear programming duality which we will explore later during the course.

Then we will consider flow problems on graphs whose capacities may be very large. The path-packing algorithms you have seen in the prior lectures have runtimes that depend linearly on the largest weight in the graph. In this lecture we will give an algorithm that improves that dependency to logarithmic using a technique called "capacity scaling".

2 Assumed Knowledge

1. Graphs: vertices, edges, weights
2. Paths in graphs, finding paths.
3. Matrix-vector products, linearity.

3 Maxflow-Mincut Duality

3.1 Definition of Cut

Let $G = (V, E)$ be a graph with directed edges. For a subset $S \subseteq V$, the cut induced by S is the set of edges:

$$E(S, V \setminus S) := \{e = (u, v) : u \in S, v \notin S\}.$$

In other words, it consists of edges that start in S and end outside of S.

The size of the cut induced by S is simply the number of edges, i.e., $|E(S, V \setminus S)|$.

For a source vertex $s \in V$ and sink $t \in V$, the minimum cut between s and t is

$$\text{mincut}(s, t) = \min_{S \subseteq V, s \in S, t \notin S} |E(S, V \setminus S)|.$$

3.2 Equality of Maximum Flow and Minimum Cut

Weak duality. Let $\text{flow}(s, t)$ denote the value of the maximum flow from $s \to t$ in the graph G. First we will argue that:

$$\text{flow}(s, t) \leq \text{mincut}(s, t).$$

The proof is simple. Let S be the subset realizing the mincut. Consider deleting the edges in $E(S, V \setminus S)$. After deleting all edges there is no more $s \to t$ path in the graph. Also, each edge deletion can delete at most one flow path. So there must have been at most $\text{mincut}(s, t)$ flow paths in total.

This captures an important fact to convince yourself of: deleting edges of total capacity X reduces the flow by at most X.

Strong duality. The more magical fact is that

$$\text{flow}(s, t) = \text{mincut}(s, t).$$

The proof of this is a bit more involved, so we simply provide a sketch.

Consider a maximum flow f from s to t in G and let $\hat{G}$ be the induced residual graph. Because the flow is maximum, there is no s to t path in $\hat{G}$. Define S to be the set of vertices that s can reach in $\hat{G}$ (say via BFS/DFS). $t \notin S$ because s cannot reach t in $\hat{G}$. Then you can check that

$$|E(S, V \setminus S)| \leq \text{flow}(s, t),$$

by flow conservation.

3.3 Problem: Vertex Connectivity

Problem: Vertex connectivity

Let G be a directed graph and let s and t be so that there is no $s \to t$ edge. What is the minimum number of vertices one must remove from G so that there is no $s \to t$ path? Give a polynomial time algorithm.

We will construct an auxiliary graph as follows. For each vertex $v \neq s, t$ turn it into two vertices v_1 and v_2. Add an edge $v_1 \to v_2$. For every original edge $u \to v$, add the edge $u_2 \to v_1$ (you may assume that s and t have no incoming/outgoing edges respectively). Then you can simply return the minimum cut in this graph.

Here is the proof. It only makes sense to delete edges of the form $v_1 \to v_2$ in your cut. Indeed, if you wanted to delete $u_2 \to v_1$ you could have either simply delete $u_1 \to u_2$ or $v_1 \to v_2$ (at least one of which exists, because you can't have both $u = s$ or $v = t$). So the minimum cut is the minimum number of $v_1 \to v_2$ edges needed to delete to disconnect s from t, which corresponds to a set of vertices to delete.

4 Capacity Scaling

Switching gears for a bit, we now consider computing a maximum flow on a graph that has positive integer capacities at most U and m edges. The maximum flow value is at most $F = mU$ so the Ford-Fulkerson algorithm would take time $O(m \cdot mU) = O(m^2 U)$. Today we will argue how to improve this to $O(m^2 \log(mU))$ using a technique called "capacity scaling". The high-level idea is that if there is an edge with huge capacity U and another will tiny capacity 1, probably we should ignore the tiny capacity edges until we really need to use them later, and at the start we should use the big edges to make big progress.

Here is the formal algorithm description. The algorithm maintains a current flow f at step t, and a guess F which upper bounds the total remaining flow in the graph.

We initialize the flow $f \leftarrow 0$ and $F \leftarrow mU$.

1. Iterate $i = 0, 1, \ldots, O(\log mU)$.
   (a) If $F \leq 10m$, run Ford-Fulkerson which takes time $O(m^2)$ and **terminate**.
   (b) Otherwise, let R be the residual graph of the flow f.
   (c) Let $\hat{R}$ be the rounded residual graph where all capacities are rounded down to the nearest multiple of $\lfloor F/(4m) \rfloor$.
   (d) Run Ford-Fulkerson on $\hat{R}$ and update the flow f.
   (e) Update $F \leftarrow F/2$.

We have to argue two things: that $F \leftarrow F/2$ is valid (the remaining flow is at most $F/2$) and the runtime bound, i.e., each inner loop runs in time $O(m^2)$.

Validity of halving F. This is simple. By rounding each edge down to the nearest $X = \lfloor F/(4m) \rfloor$, each edge throws out at most X units of capacity. Because R has at most $2m$ edges, we throw out at most $2mX \leq F/2$ units of flow.

Runtime of inner loop. The main thing to check is that running Ford-Fulkerson on $\hat{R}$ costs time $O(m^2)$. There are two ways to see this. (1) Consider dividing the capacity of each edge in R

by X. Then, the total flow is at most $F/X \leq O(m)$ by the definition of X. Thus, Ford-Fulkerson runs in time $O(m^2)$.

Total runtime. Because there are $O(\log mU)$ outer steps, the total runtime is $O(m^2 \log(mU))$. Within $O(\log mU)$ steps, the value of F falls below $10m$.