COMPSCI 638: Graph Algorithms August 30, 2019

Lecture 2

Lecturer: Debmalya Panigrahi *Scribe: Kevin Sun*

1 Overview

In the last lecture, we introduced the maximum flow problem and the Ford-Fulkerson algorithm. In this lecture, we will first look at a linear programming approach to understanding the max-flow min-cut theorem. Then we will study the Edmonds-Karp algorithm, a modification of the Ford-Fulkerson algorithm.

2 Proving Max-flow Min-cut via LP Duality

Throughout this lecture, we will again assume that $G = (V, E)$ is a directed graph with n vertices and m edges, each edge e has a nonnegative capacity $u(e)$, and there is a designated source vertex s and sink vertex t.

First, let us define a linear program (LP) for the maximum flow problem. Each decision variable f_p represent the amount of flow to send on an s-t path p. Let $P_{s,t}$ denote the set of all s-t paths; note that the size of $P_{s,t}$ is, in general, exponential in the number of vertices. In the maximum flow problem, the total flow along each edge cannot exceed its capacity, and all flows must be positive. This leads us to the following LP, which we call the "primal" linear program (P):

$$\text{(P): } \max \sum_{p \in P_{s,t}} f_p$$

$$\sum_{p: e \in p} f_p \leq u(e) \quad \forall e \in E$$

$$f_p \geq 0 \quad \forall p \in P_{s,t}$$

Now let us consider the dual (D) of this LP. In general, we construct the dual of an LP by associating each constraint with a variable, and each variable with a constraint. So for each edge $e \in E$, we have a variable x_e, and for each path $p \in P_{s,t}$, we will have a constraint for p. We omit the details of constructing a dual, but this process is typically straightforward. For the maximum flow problem, the resulting dual is the following linear program:

$$\text{(D): } \min \sum_{e \in E} u(e) x_e$$

$$\sum_{e \in P} x_e \geq 1 \quad \forall p \in P_{s,t}$$

$$x_e \geq 0 \quad \forall e \in E$$

This linear program is sometimes referred to as a "volume minimization" problem: if we interpret x_e and $u(e)$ as the "length" and "width" of edge e respectively, then the objective of (D) is to minimize the total volume with the constraint that the distance from s to t is at least one.

In general, the constraints of a linear program do not necessarily result in an integral optimal solution. However, when a linear program does have an integral optimal solution, then we call that linear program *integral*.

Theorem 1. *The linear program (D) is integral, that is, there exists an optimal solution in which x_e is an integer for every $e \in E$. In fact, this solution satisfies $x_e \in \{0, 1\}$ for every $e \in E$.*

Before proving this theorem, let's show that it implies the max-flow min-cut theorem. Let x^* be an integral optimal solution of (D), and let $z^* = \sum_{e \in E} u(e) x_e^*$ denote the objective value achieved by x^*. From Theorem 1, x^* exists and $x_e^* \in \{0, 1\}$ for every $e \in E$.

Now consider the following subset of vertices:

$$S^* = \{v : \text{there exists an } s\text{-}v \text{ path on edges with } x_e^* = 0\}.$$

Notice that S^* defines an s-t cut (from our dual constraint), and the capacity of this cut is precisely z^*. This is because every edge leaving S^* has $x_e^* = 1$ (from the definition of S^*), and x^* is optimal, so all other edges have $x_e^* = 0$. By the strong duality theorem, z^* is also the optimal value of (P), i.e., the value of the maximum flow in this network. Thus, the capacity of S^* is equal to the maximum flow value, as desired.

Proof of Theorem 1. Let x^* be an optimal solution of (D), and let $z^* = \sum_{e \in E} u(e) x_e^*$. Also, let $d(v)$ be the shortest path length from s to v, where the length of edge e is x_e^*. Let us define an s-t cut S by choosing a number r uniformly at random in $(0, 1)$ and setting

$$S = \{v : d(v) < r\}.$$

The expected capacity of S is $\sum_{e \in E} u(e) p_e$ where p_e is the probability of e appearing in S. We shall bound this value by analyzing an edge $e = (a, b)$. Consider the following cases:

1. $d(a) \geq d(b)$: The edge (a, b) cannot leave S, so $p_e = 0$.

2. $d(a) < d(b)$: From the definition of S, we have $p_e \leq d(b) - d(a)$. (The inequality can be strict if $d(b) > 1$.) From the triangle inequality, we have $d(b) \leq d(a) + x_e^*$, which implies $p_e \leq x_e^*$.

In either case, we have $p_e \leq x_e^*$, the expected capacity of S is at most $z^* = \sum_{e \in E} u(e) x_e^*$.

Our random cut S was implicitly drawn from a distribution over all s-t cuts, so the minimum s-t cut must have capacity at most z^* as well. This minimum cut yields an integral optimal solution to (D): set $x_e^* = 1$ if e leaves the cut, and 0 otherwise. □

Finally, let us also show that if the capacity of the edges are all integral, then there exists a maximum flow in which every edge carries an integral amount of flow. In other words, if $u(e)$ is an integer for every $e \in E$, then the linear program (P) is also integral. This follows readily from the Ford-Fulkerson algorithm: each augmenting path adds an integral amount of flow, so all modifications made to the residual network are integral. Since the sum of integral flows is integral, the maximum flow returned by Ford-Fulkerson is also integral.

3 The Edmonds-Karp Algorithm

Let us recall the Ford-Fulkerson algorithm from Lecture 1: while there exists an s-t path in the residual network G_f, send flow along this path and update G_f. Each s-t path that increases flow is known as an *augmenting path*, and the augmenting paths found in Ford-Fulkerson are arbitrary.

Each path takes $O(m)$ to find and the number of paths is f, the value of the maximum flow, so the runtime of Ford-Fulkerson is $O(mf)$. This might look polynomial but is actually "pseudo-polynomial" because of the f factor: it only requires $\log U$ bits in the input to represent a capacity value U, so $O(mf)$ is not polynomial in the *size* of the input.

The Edmonds-Karp [EK72] algorithm applies a small modification to the Ford-Fulkerson algorithm: instead of increasing flow along an *arbitrary* s-t path, we increase flow along a *shortest* s-t path. (The length of a path is measured by the number of edges it comprises.)

Algorithm 1 (Edmonds-Karp 1972)

```
1: Initialize: $f(x,y) \leftarrow 0 \quad \forall (x,y) \in E, \quad G_f \leftarrow G$
2: while $G_f$ has an $s$-$t$ path do
3:     $\Gamma \leftarrow$ a shortest $s$-$t$ path in $G_f$
4:     $\delta \leftarrow \min_{e \in \Gamma} u_f(e)$
5:     $g \leftarrow$ flow on $\Gamma$ with value $\delta$
6:     $f \leftarrow f + g$
7: end while
```

Theorem 2. *The Edmonds-Karp algorithm runs in $O(m^2 n)$ time.*

To prove this theorem, we let $d(x,y)$ denote the shortest path distance from x to y in G_f (again, measured by the number of edges). Also, we say an edge e is *saturated* by an augmenting path if its residual capacity before the augmentation is positive and zero after the augmentation.

Lemma 3. *In the Edmonds-Karp algorithm, $d(s,v)$ and $d(v,t)$ never decrease for any vertex v.*

Proof. For contradiction, let Y be the set of vertices whose distance from s decreases after augmenting on a path Γ, and let $y \in Y$ be the closest vertex from s after the augmentation. Also, let G_f and G'_f denote the residual networks before and after this augmentation, respectively, and let $d(\cdot,\cdot)$ and $d'(\cdot,\cdot)$ denote the respective distance functions in these graphs.

Let x be the predecessor of y on a shortest s-y path in G'_f and consider the edge $e = (x,y)$. From the definitions, we have

$$d(s,y) > d'(s,y) = d'(s,x) + 1 = d(s,x) + 1. \tag{1}$$

Now we claim that e does *not* exist in G_f. If e is in G_f, then

$$d(s,y) \le d(s,x) + 1$$

because one s-y path is to follow a shortest s-x path and then the edge $e = (x,y)$. But this contradicts (1), so we can conclude e is not in G_f.

Thus, $e = (x,y)$ must have been created when we augmented on Γ, which implies that Γ sends flow from y to x. But from (1), we have $d(s,y) > d(s,x) + 1$, so (y,x) cannot be part of a shortest s-t path, contradicting the definition of Γ. The proof for $d(v,t)$ is nearly identical. □

Lemma 4. *In the Edmonds-Karp algorithm, no edge gets saturated more than $\frac{n-1}{2}$ times.*

Proof. Whenever an edge $e = (x, y)$ is saturated, we have $d(s, x) < d(s, y)$ before the saturation. For this edge to appear again, we must first send flow from y to x to "recover" some residual capacity of (x, y). On this recovery path, we must have $d(s, y) < d(s, x)$; in other words, x and y must alternate as "the one closer from s" whenever (x, y) gets saturated.

From Lemma 3, we know that $d(s, y)$ cannot decrease, which implies $d(s, x)$ increases by at least two between any two saturations of (x, y). The maximum value of $d(s, x)$ is $n - 1$, so the number of times (x, y) gets saturated is at most $(n - 1)/2$. □

Proof of Theorem 1. Each iteration of Edmonds-Karp saturates at least one edge, so from Lemma 4, the total number of iterations is at most $m(n - 1)/2$. Each iteration takes $O(m)$ time, so the overall runtime is $O(m^2 n)$. □

We conclude by noting that although the Edmonds-Karp algorithm runs in polynomial time, it still proceeds in a path-by-path manner, and this results in one of the $O(m)$ factors in the runtime. In the next lecture, we will learn about *blocking flows*, flows that saturate *every* shortest *s-t* path. As we will see, this additional modification to the Ford-Fulkerson algorithm further reduces its overall runtime.

4 Summary

In this lecture, we used linear program duality to prove the max-flow min-cut theorem. We also saw that a small modification to the Ford-Fulkerson algorithm—using a *shortest* rather than *any s-t* path—reduces its runtime from pseudo-polynomial to polynomial.

References

[EK72] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.