COMPSCI 638: Graph Algorithms August 28, 2019

Lecture 1

Lecturer: Debmalya Panigrahi *Scribe: Kevin Sun*

1 Overview

In this lecture, we will define the basics of network flows. In particular, we will look at the maximum flow problem and the Ford-Fulkerson algorithm. Along the way, we will see multiple concepts that are fundamental to the study of network flows.

2 Network Flows

Throughout this lecture, we will assume that $G = (V, E)$ is a directed graph with n vertices, m edges, a "source" vertex s, and a "sink" vertex t. Furthermore, each edge $e = (x, y)$ has a nonnegative capacity denoted by $u(e)$ or $u(x, y)$, and we call such a graph a network. For each vertex $u \in V$, we let $E_{in}(u)$ and $E_{out}(u)$ denote the set of edges entering and leaving u, respectively.

Intuitively, we can think of a flow on G as a description of the quantity of water flowing through each edge, which represents a pipe. This idea leads us to the following definition.

Definition 1. *A* raw flow *on G is a function $r : E \to \mathbb{R}$ that satisfies the following:*

1. *Nonnegativity: $r(e) \geq 0 \quad \forall e \in E$.*
2. *Capacity constraints: $r(e) \leq u(e) \quad \forall e \in E$.*
3. *Conservation at each vertex: $\sum_{e \in E_{in}(v)} r(e) = \sum_{e \in E_{out}(v)} r(e) \quad \forall v \in V \setminus \{s, t\}$.*

The value *of such a flow, denoted by $|r|$, is the net amount of flow leaving the source vertex s. In other words, $|r| = \sum_y r(s, y) - \sum_x r(x, s)$.*

The other way of describing a flow is known as a net flow. The nonnegativity of raw flaws makes them easier to visualize, but mathematically, working with net flows often yields simpler proofs. As we will see, the concept of raw flows and net flows are equivalent.

Definition 2. *A* net flow *is a function $f : E \to \mathbb{R}$ that satisfies the following:*

1. *Skew symmetry: $f(u, v) = -f(v, u) \quad \forall u, v \in V$.*
2. *Capacity constraints: $f(e) \leq u(e) \quad \forall e \in E$.*
3. *Conservation at each vertex: $\sum_{e \in E_{in}(v)} f(e) = 0 \quad \forall v \in V \setminus \{s, t\}$.*

The value *of such a flow, denoted by $|f|$, is the amount of flow leaving the source vertex s. In other words, $|f| = \sum_y f(s, y)$.*

Note that the value of a flow can also be defined with respect to the sink vertex t. More specifically, it is equivalent to define $|r|$ as $\sum_x r(x,t) - \sum_y r(t,y)$ and $|f|$ as $\sum_x f(x,t)$.

We now describe a process that converts a raw flow to a net flow with equal value, and vice versa. Given a raw flow r, consider the flow f defined by $f(x,y) = r(x,y) - r(y,x)$. It is straightforward to verify that such a flow is a net flow, i.e., satisfies the three properties listed in the definition above. The reverse process is simpler: given a net flow f, let r be a flow defined by $r(x,y) = f(x,y)$ if $f(x,y) \geq 0$, and 0 otherwise. It is similarly straightforward to verify that such a flow is a raw flow.

The Maximum Flow Problem: Given a network, find a net flow with maximum flow. This is a fundamental problem in graph algorithms, and there are many algorithms that solve it. Before we present one such algorithm, let us take a deeper look at net flows.

2.1 Flow Decomposition

A fundamental property of flows is that any flow can be decomposed into a set of paths from s to t (known as s-t paths) and cycles. We will use this fact to obtain an upper bound on the value of any flow in a network.

Lemma 1 (Flow decomposition)**.** *Any net flow f can be decomposed into at most m cycles and s-t paths.*

Proof. Let f be a net flow and consider the following algorithm:

1. Let Γ be an s-t path or cycle with positive flow.
2. Let δ denote the minimum flow on any edge of Γ.
3. Reduce the flow on every edge of Γ by δ.
4. Add Γ as an element of the flow decomposition.
5. Repeat from Step 1 until all edges of the network have zero flow.

First, note that Step 1 is always feasible as long as the value of the flow is non-zero. Now notice that each time we enter Step 3, we have reduced the flow on at least one edge to 0. Since there are m edges, this implies we enter step 3 (and consequently, Step 4) at most m times, and each iteration contributes one cycle or s-t path to the decomposition. □

2.2 The Ford-Fulkerson Algorithm

Given a net flow f on a network G, we can define a new network G_f that captures the capacity of G that "remains" after we apply f on G. This residual network forms the foundation of many maximum flow algorithms.

Definition 3. *Let $G = (V, E)$ be a network, and let f be a net flow on G. The* residual network *$G_f = (V, E_f)$ is a network with edge capacities defined as $u_f(x,y) = u(x,y) - f(x,y)$.*

Residual networks are useful because of the following additive property of flows: if f is a net flow on G, and g is a net flow on the residual network G_f, then $f + g$ is a net flow on G with value $|f| + |g|$. This idea is the basis for an algorithm known as the Ford-Fulkerson [FF56] algorithm, the earliest and most well-known algorithm for the maximum flow problem.

Algorithm 1 (Ford-Fulkerson 1956)

Initialize: $f(x,y) \leftarrow 0 \quad \forall (x,y) \in E, \quad G_f \leftarrow G$
while G_f has an s-t path **do**
    $\Gamma \leftarrow s$-t path in G_f
    $\delta \leftarrow \min_{e \in \Gamma} u_f(e)$
    $g \leftarrow$ flow on Γ with value δ
    $f \leftarrow f + g$
end while

Theorem 2. *The Ford-Fulkerson algorithm finds a net flow with maximum value.*

Proof. The algorithm terminates when there is no s-t path in G_f. Upon termination, let S denote the set of vertices that can be reached from s in G_f, and let $T = V \setminus S$. We know that $t \in T$ and there are no edges leaving S in G_f. Now consider any edge e leaving S in G: since $u_f(e) = 0$, we must have $f(e) = u(e)$. Thus, the total flow from S to T is the total capacity of edges from S to T. Since every edge from S to T is fully saturated by the current flow, this flow must be maximum. □

When proving the correctness of the Ford-Fulkerson algorithm, we referred to a concept known as cuts, and we also touched upon the max-flow min-cut theorem.

Definition 4. *In a network G, an s-t* cut *is a subset S of vertices that contains s and does not contain t. The* capacity (or value) *of a cut S is the sum of the capacity of edges leaving S.*

Theorem 3 (Max-flow Min-cut)**.** *Let $G = (V, E)$ be a network with source s and sink t. The maximum flow value from s to t is equal to the minimum s-t cut capacity, which is equal to the maximum number of paths from s to t.*

Notice that if every edge of the network has a capacity of 1, then the s-t paths found by the Ford-Fulkerson algorithm are edge-disjoint. Thus, by the max-flow min-cut theorem, the maximum number of edge-disjoint s-t paths in a graph is equal to the capacity of the minimum s-t cut; this result is known as Menger's theorem.

3 Summary

In this lecture we studied the Ford-Fulkerson algorithm for the maximum flow problem, and also looked at related concepts such as flow decomposition and the max-flow min-cut theorem.

References

[FF56] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.