Network Flow and Ford-Fulkerson

Suhas Arehalli

COMP221 - Spring 2024

1 Network Flow Problems

- A different kind of graph problem!
- Rather than have weights represent the *cost* of an edge, weights represent *capacities* across edges.
- The metaphor is that the graphs represent a some kind of flow along edges from one vertex to another (liquid through pipes, packets through a network of computers, people through public transit, etc.), and we want to measure the possible *flow* through the graph. Capacities are limits on the amount of a thing that can flow through any particular edge.
- Let's formalize: Consider a graph $G = (V, E)$ with capacities $c : E \to \mathbb{R}_{\geq 0}$.
- How much can flow from s to t?
  - Along any particular edge e, the maximum flow is equivalent to the edge's capacity $c(e)$.
  - Along a path P, our flow can get *bottle-necked*! We're limited by the smallest capacity along that path, $\min_{e \in P} c(e)$
  - Across a full graph, things are more complicated — think of all of the different ways that flow can travel! We'll need to develop some theory in order to make sense of this problem!

2 Network Flow Theory

2.1 Networks and Flow

Def. A **(flow) network** is a directed graph $G = (V, E)$ with a *capacity function* $c : V^2 \to \mathbb{R}_{\geq 0}$ and *source and sink vertices* $s, t \in V$.

Our intuition is that each edge has a flow capacity determined by c, and that flow will travel from our source vertex along the edges toward our sink vertices. The fact that the graph is directed allows us to specify that the capacity can depend on direction — the capacity from vertex u to v doesn't necessarily need to be the capacity from v to u.

Note that c also has V^2 (rather than E) as it's domain. This will just make our notation easier! We just stipulate that $c(u, v) = 0$ if $(u, v) \notin E$. This is similar to how we stipulate $c(u, v) = \infty$ if

$(u, v) \notin E$ in our shortest path algorithms — it's a default value that specifies that two vertices are not adjacent, with the value differing because of the different interpretations of the two kinds of edge weights (cost vs capacity).

Now what are flows in a flow network? Formally:

Def. A **flow** over a flow network $G = (V, E)$ with $c : V^2 \to \mathbb{R}_{\geq 0}$ is a map $f : V^2 \to \mathbb{R}$ such that:

- $f(u, v) \leq c(u, v)$ for all $u, v \in V$ (Capacity Constraint)
- $f(u, v) = -f(v, u)$ for all $u, v \in V$ (Skew Symmetry)
- $\sum_{u \in V} f(u, v)$ for all $v \in V \setminus \{s, t\}$ (Conservation of Flow)

The 3 properties should capture some of you intuitions about what flow through a network should look like! We have a capacity constraint because c sets a limit on flow through each edge. We enforce skew symmetry because we think about flow having direction: If vertex v is getting 3 units of flow from u, vertex u is *losing* 3 units of flow to v — equivalent to getting -3 units of flow! Finally, conservation of flow tells us that the net flow through vertices should be 0 (what comes in goes out, and what goes out must have come in!). The exceptions are the source (which can generate flow freely) and the sink (which can absorb flow freely).

Def. The **value** of a flow f is

$$|f| = \sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t).$$

Note that since the source and sink do not have to conserve flow, they can generate/absorb flow! I will also claim that the amount they generate/absorb must be equal (you can prove this via conservation!). As a result, we can use this value to represent the value of our flow.

Note that here we've talked about a kind of graph (a flow network), a kind of object we can construct over that graph (a flow), and a way to measure how good that object is (a flow's value). Thus we have the set-up for another graph optimization problem — Max Flow!

Def. The **Maximum Flow Problem**

INPUT: A flow network $G = (V, E)$, capacity function $c : V^2 \to \mathbb{R}_{\geq 0}$, and vertices s, t

OUTPUT: A flow f such that $|f|$ is maximized.

2.2 Residual Graphs & Augmenting Paths

Our strategy for solving the Maximum-Flow Problem will involve incrementally improving a sub-optimal flow by finding *Augmenting Paths* — informally, paths from s to t along which we can increase the flow. This idea is the *Ford-Fulkerson method*, which we'll talk about formally a bit later. First, we'll develop some theory about Augmenting Paths (APs) and a structure we'll use to help find APs – a Residual Graph!

Def. The **Residual Graph** of a flow network with $G = (V, E)$ and capacity $c : V^2 \to \mathbb{R}_{\geq 0}$ with respect to a flow f is

$$G_f = (V, E_f)$$

with **residual capacity** $c_f : V^2 \to \mathbb{R}$

$$c_f(u,v) = c(u,v) - f(u,v)$$

where

$$E_f = \{(u,v) \in V^2 \mid c_f(u,v) > 0\}$$

The existence of an AP implies that we are underutilizing the capacity along that path. To find one, it might be helpful to have some idea of how much of the capacity along each edge is being unutilized — this is our intuition for building the residual graph!

Now we can talk about APs formally:

Def. An **Augmenting Path** P over a flow network $G = (V, E)$ with cost $c : V^2 \to \mathbb{R}_{\geq 0}$ with respect to flow f is a path from s to t in G_f with residual flows c_f.

The key here is matching this definition to our intuition: First note that c_f is non-negative because we have a capacity constraint on our flow. Thus, if we have a path from s to t in our residual graph, this means that we have a path from s to t along which every edge has positive unutilized capacity! Thus, we can improve our flow function by increasing flow along the corresponding edges in G by the maximum flow through the AP!

Now, here's the Ford-Fulkerson Method:

1. Start with a trivial flow function $f(u,v) = 0$.
2. Build G_f and find an augmenting path P. If we can't, we're done!
3. If we can, find $k = \min_{(u,v)\in P} c_f(u,v)$ — the bottleneck for flow through the augmenting path.
4. Build $f' = \begin{cases} f(u,v) + k, & (u,v) \in P \\ f(u,v), & \textit{otherwise} \end{cases}$. This is a better flow!
5. Repeat from step 2 with $f \leftarrow f'$

2.3 Runtime Complexity

How fast does Ford-Fulkerson run? In practice, it depends on how you find an augmenting path. More interesting that comparing different graph traversal methods (i.e., Edmonds-Karp using BFS is best!) is considering why selecting a *good* AP is critical. Alternatively, let's talk about why selecting a *bad* AP can be tragic! Consider the graph in Fig. **??**

- Observe that if we pick *good* augmenting paths, we can find the maximum flow in two iterations of Ford-Fulkerson: Find the paths $P_1 = s, v_1, t$ and $P_2 = s, v_2, t$ to get the maximum flow $2k$!
- But what if we use an algorithm that picks really bad augmenting paths?
  - Consider first an augmenting path $P = s, v_1, v_2, t$. This path can carry a maximum flow of 1 (bottlenecked by (v_1, v_2))!
  - The residual graph will then look like Fig. **??**. Then let's run another step of Ford-Fulkerson with the augmenting path in red, which will have maximum flow 2!

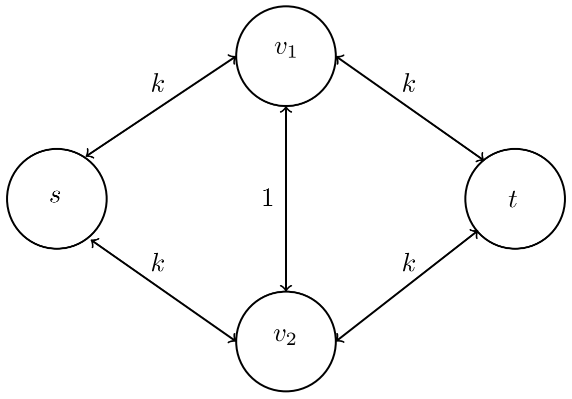

Figure 1: A network G with labels representing capacities

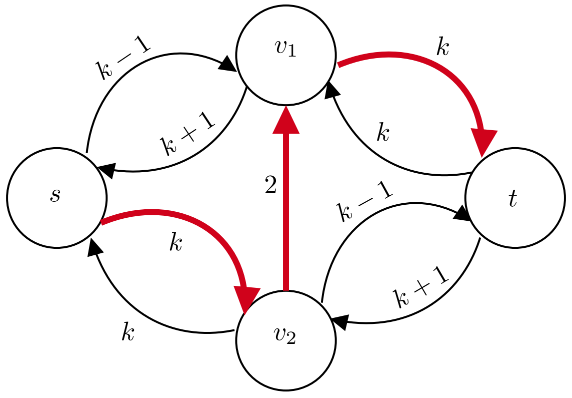

Figure 2: A residual graph G_f after one step of Ford-Fulkerson on the graph in Fig. **??** with augmenting path $P = s, v_1, v_2, t$. Consider the next (bad) augmenting path in red.

  - Now if you draw the residual graph that results, you'll see that the middle edge between v_1 and v_2 runs in one direction with residual capacity 2. We can repeatedly construct a bad augmenting path through that edge that has a maximum flow of 2.
  - This means that a particularly bad (i.e., *worst-case*) implementation of Ford-Fulkerson would take $Theta(k)$ iterations of the while loop in order to find the maximum flow on this graph!

Graphs like this demonstrate that, worst-case, with no requirements on how the APs are chosen, the time complexity of Ford-Fulkerson depends on strange things — the edge capacities and the value of the maximum flow! In fact, if we assume that capacities are integers, we can show that Ford-Fulkerson will find a maximum flow in $O(f \cdot g(n))$ time, where f is the maximum flow and we know we can find an AP in $O(g(n))$ time (typically something like $O(|V| + |E|)$, since traversal algorithms like BFS/DFS run in graph-linear time).

- Why? Consider: What is the minimum increase in flow from each augmenting path?

Edmonds-Karp (i.e., using BFS for find augmenting paths) is the preferred Ford-Fulkerson algorithm because it can be shown that we tend to pick good paths: We need, at worst, $|V||E|$ augmenting paths to find the maximum flow). This gives us a time complexity of $\Theta(|V||E|(|V| + |E|))$, which is often just written as $\Theta(|V||E|^2)$ assuming that $|E| \gg |V|$ (i.e., the graph is dense).

2.4 Correctness

We're not going to *formally* prove this algorithm correct, since much of the interesting stuff is out-of-scope for this class, and the stuff is that in-scope is routine rather than enlightening. Regardless, it's worth summarizing the relevant results and the proof structure, because that may help you understand why this technique works!

First, we need to prove (with a loop invariant) that the algorithm computes a valid flow at each step (all flow properties are preserved!). This should be fairly straightforward!

Then we must show that once no augmenting paths exist, we have a maximum flow. This uses a result called the **Max-Flow Min-Cut Theorem**, which relates the maximum flow to another kind of object called an *s–t cut*.

Def. An *s–t* **cut** of a Graph $G = (V, E)$ with vertices $s, t \in V$ is a partition of V into V_s, V_t such that $s \in V_s$ and $t \in V_t$.

Given a cost function $c : V^2 \to \mathbb{R}$, the **cost** of an *s–t* cut V_s, V_t is

$$c(V_s, V_t) = \sum_{\substack{(u,v)\in E,\\ u\in V_s,\\ v\in V_t}} c(u, v)$$

The idea is that we split the graph into two pieces, one with s and the other with t, and we assign a cost to this split equal to the cost of every edge that goes from s to t — those edges that *cross the cut*, in graph parlance. Remember that this graph is directed!

The part of the Max-Flow Min-Cut Theorem we care about states the following: The cost of the minimum cut of a flow network is equal to the value of the maximum flow! With this, we have the following argument:

Suppose no AP exists on the residual graph G_f. This means that s is disconnected from t over G_f, and thus we can construct the following s–t cut: Let $V_s = \{v \in V \mid s \text{ is connected to } v\}$ and $V_t = V \setminus V_s$. We can then do a little algebra to show that via our definitions, we know $|f| = c(V_s, V_t)$. Since f is a real flow, V_s, V_t is a real s–t cut, and the maximum flow has value equal to that of the minimum cut, we must conclude that f is a maximum flow and s–t is a minimum cut!

If taking the Max-Flow Min-Cut Theorem on authority is unconvincing, lets think about the relationship between a cut and flow and build an intuition:

- If $s \in V_1$ and $t \in V_2$, for any flow to get from s to t, it *must* cross the cut.
- Since this is true for *any* cut, it must be true for the *cheapest* cut.
- This means that the cheapest cut acts as a bottleneck on flow! Since all flow must cross through the cheapest cut, the maximum possible flow is the capacity/cost of that cut!

If you want a fully formal unpacking of all of this, Chapter 24.2 of CLRS (the alternative textbook) has a fairly good exposition. However, be warned that they use a slightly different formalization of flows and residual networks than we do, but perhaps that will build an appreciation for the simplifications we make!