Mathematical Foundations of Computer Science

CS 499, Shanghai Jiaotong University, Dominik Scheder

Spring 2019

10 Network Flow

- Homework assignment published on Thursday 2019-05-16
- Submit questions and first solution by Wednesday, 2019-05-22, 12:00
- Submit final solution by Wednesday, 2019-05-29.

Exercise 10.1. [From the video lecture] Recall the definition of the value of a flow: $\text{val}(f) = \sum_{v \in V} f(s,v)$. Let $S \subseteq V$ be a set of vertices that contains s but not t. Show that

$$\text{val}(f) = \sum_{u \in S, v \in V \setminus S} f(u,v) \ .$$

That is, the total amount of flow leaving s equals the total amount of flow going from S to $V \setminus S$. **Remark.** It sounds obvious. However, find a formal proof that works with the axiomatic definition of flows.

Proof.

$$\text{val}(f) = \sum_{u \in S, v \in V \setminus S} f(u,v) = \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e)$$

$\sum_{e \text{ out of } v} f(e) = \sum_{e \text{ in to } v} f(e)$, if $v \neq s, t$.
Thus,

$$\begin{aligned}
\text{val}(f) &= \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ in to } s} f(e) \\
&= \sum_{v \in S} \Big(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \Big) \\
&= \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) \\
&= \sum_{u \in S, v \in V \setminus S} f(u, v)
\end{aligned}$$

□

Exercise 10.2. Let $G = (V, E, c)$ be a flow network. Prove that flow is "transitive" in the following sense: If there is a flow from s to r of value k, and a flow from r to t of value k, then there is a flow from s to t of value k. **Hint.** The solution is extremely short. If you are trying something that needs more than 3 lines to write, you are on the wrong track.

Proof. A flow from s to r of value k: $k = \sum_{e \text{ in to } r} f(e)$
A flow from r to t of value k: $k = \sum_{e \text{ out of } r} f(e)$
Then there must be a flow from s to t of value k. □

10.1 An Algorithm for Maximum Flow

Recall the algorithm for Maximum Flow presented in the video. It is usually called the Ford-Fulkerson method.

We proved in the lecture that f is a maximum flow and S is a minimum cut, by showing that upon termination of the while-loop, $\text{val}(f) = \text{cap}(S)$. The problem is that the while-loop might not terminate. In fact, there is an example with capacities in $\mathbb{R}$ for which the while loop does not terminate, and the value of f does not even converge to the value of a maximum flow. As indicated in the video, a little twist fixes this:

> **Edmonds-Karp Algorithm:** Execute the above Ford-Fulkerson Method, but in every iteration choose p to be a shortest s-t-path in G_f. Here, "shortest" means minimum number of edges.

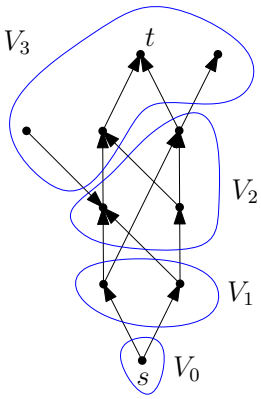
Algorithm 1 Ford-Fulkerson Method

```
1: procedure FF(G = (V, E), s, t, c)
2:     Initialize f to be the all-0-flow.
3:     while there is a path p form s to t in the residual network G_f do
4:         c_min := min{c_f(e) | e ∈ p}
5:         let f_p be the flow in G_f that routes c_min flow along p
6:         f := f + f_p
7:     end while
8:     // now f is a maximum flow
9:     S := {v ∈ V | G_f contains a path from s to v}
10:    // S is a minimum cut
11:    return (f, S)
12: end procedure
```

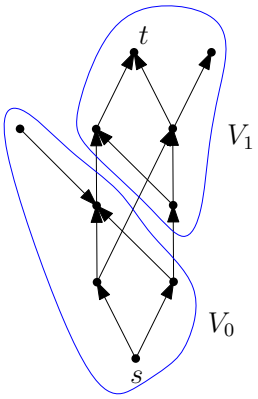
In a series of exercises, you will now show that this algorithm always terminates after at most $n \cdot m$ iterations of the while loop (here $n = |V|$ and $m = |E|$).

Definition 10.3. *Let (G, s, t, c) be a flow network and $k \in \mathbb{N}_0$. A* k-layering *is a partition of $V = V_0 \cup \cdots \cup V_k$ such that (1) $s \in V_0$, (2) $t \in V_k$, (3) for every edge $(u, v) \in E$ the following holds: suppose $u \in V_i$ and $v \in V_j$. Then $j \leq i + 1$. In words, point (3) states that every edge moves at most one level forward.*

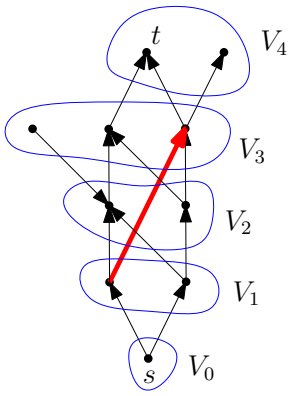
The figure below illustrates this concept: for one network we show two possible layerings and something that looks like a layering but is not:



A 3-layering.



A 1-layering.



Not a layering: the fat red edge "jumps ahead".

Exercise 10.4. Suppose the network (G, s, t, c) has a k-layering. Show that $\text{dist}(s, t) \geq k$. That is, every s-t-path in G has at least k edges.

Proof. **Claim:** With i edges, a path starting from s can reach at farthest a vertex in V_i in $k-$layering $(1 \leq i \leq k)$.

Proof of claim: Suppose the claim does not hold. That is, there is some path starting from s with i edges that can reach a vertex in V_j, $i < j \leq k$. And we know that S is in V_0. Then there must exist some edge (u, v), $u \in V_m, v \in V_n$ such that $n - m > 1$ in the path, which violates the definition of layering. □

Let $i \leq k-1$ and we get that with less than k edges, a path starting from s can reach at farthest a vertex in V_{k-1}, but can never reach a vertex in V_k. To reach $t \in V_k$, the path must include at least k edges. □

Exercise 10.5. Conversely, suppose $\text{dist}(s, t) = k$. Show that (G, s, t, c) has a k-layering.

Proof. **Claim:** Algorithm 2 outputs a $k-$layering for $\text{dist}(s, t)=k$.

Algorithm 2 Algorithm to output a $k-$layering for $\text{dist}(s, t)=k$

```
1: procedure k-LAYERING(G = (V, E), s, t, c,dist(s, t)=k)
2:     grab a shortest path p = (V', E') from s to t
3:     sort V' according to the sequence of visiting on the path
4:     for all vertex v_i ∈ V' do
5:         V_i ← {v_i}
6:     end for
7:     for i ← k to 1 do
8:         for all edge (u, v) ∉ E' with v ∈ V_i do
9:             add u to V_{i-1}
10:        end for
11:    end for
12:    return V_0, V_1, ··· , V_k
13: end procedure
```

Proof of claim: Clearly $s \in V_0$ and $t \in V_k$ according to this algorithm. We just need to prove that for every edge $(u, v) \in E$, $u \in V_i$, $v \in V_j$, $j \leq i+1$.

First for every edge $(u, v) \in E'$ in p, according to the algorithm, we have $j = i + 1$ exactly, which satisfies the requirement. Then for edges not in p,

suppose some edge has that $j > i+1$. Then we can delete at least two edges in the shortest path: one from V_i to V_{i+1} and another from V_{i+1} to V_j, replace them with the very special edge, and thus we get a shorter path than the shortest path p. Therefore all edges in E satisfy $j \leq i+1$. □

Since Algorithm 2 returns a k-layering for the network (G, s, t, c), (G, s, t, c) has a k-layering. □

Let (G, s, t, c) be a flow network and $V_0, \ldots, V_k$ a k-layering. We call this layering *optimal* if $\text{dist}_G(s, t) = k$. Here, $\text{dist}_G(u, v)$ is the shortest-path distance from s to t (measured by number of edges). If there is no path from s to t, we set $\text{dist}_G(s, t) = \infty$. In this case, no layering is optimal. For example, the 3-layering in the above figure is optimal, but the 1-layering in the middle of the above figure is not. Let us explore how layerings and the Ford-Fulkerson Method interact.

Exercise 10.6. Let (G, s, t, c) be a flow network and $V_0, V_1, \ldots, V_k$ be an optimal layering (that is, $k = \text{dist}_G(s, t)$. Let p be a path from s to t of length k. Suppose we route some flow f along p (of some value $c_{\min} > 0$) and let (G_f, s, t, c_f) be the residual network. Show that $V_0, V_1, \ldots, V_k$ is a layering of (G_f, s, t, c_f), too. Obviously, condition (1) and (2) in the definition of k-layerings still hold, so you only have to check condition (3).

Solution. For the residual network, we assume it violates condition (3), i.e., there exist an edge moves at least 2 levels forward. However, this edge doesn't exist in the original network. By the definition of residual network, we add an edge only if there exist an edge between the corresponding two vertices in the original graph. That is a contradiction. Then we claim there is no edge moves at more than 1 level forward in the residual network.

Exercise 10.7. Show that every network (G, s, t, c) has an optimal layering, provided there is a path from s to t.

Solution. Consider vertices in G, there are three types of vertices. For the first type $v \in V$, there exist both a path $s \to v$ and a path $v \to t$. Let the set of such vertices be V_{reduce}.

For the second type $v \in V$, there exist a path $s \to v$, but no path $v \nrightarrow t$. Let the set of such vertices be V_{out}.

For the second type $v \in V$, there exist a path $v \to t$, but no path $s \nrightarrow v$. Let the set of such vertices be V_{in}.

$$V = V_{reduce} \cup V_{out} \cup V_{in}$$

For $v \in V_{reduce}$, by definition, it's reachable for s. Let $c = dist_G(s, v)$ (the shortest distance from s to v). Then we assign v to c^{th} level V_c. In this way, no edge moves at more than 1 level forward. Otherwise, if we let $(v_1, v_2) \in E$, $c = dist_G(s, v_1)$, $c + 2 = dist_G(s, v_2)$, it's a contradiction because there exist a path $s \to v_1 \to v_2$ whose length is $c + 1$.

For each $v' \in V_{in}$, there exist at least one vertex in V_{reduce} that v' can first reach to. Among those vertices we choose the vertex in the highest layer. Then we assign v' to that layer.

For each $v' \in V_{out}$, there exist at least one vertex in V_{reduce} that v' can first connect to v'. Among those vertices we choose the vertex in the lowest layer. Then we assign v' to that layer.

Therefore, every network (G, s, t, c) has an optimal layering, provided there is a path from s to t.

Exercise 10.8. Imagine we are in some iteration of the while-loop of the Edmonds-Karp algorithm. Let $V_0, \ldots, V_k$ be an optimal layering of (G, s, t, c). Show that after at most m iterations of the while-loop, $V_0, \ldots, V_k$ ceases to be an optimal layering. **Remark.** Note that it is the *network* that changes from iteration to iteration of the while-loop, not the partition $V_0, \ldots, V_k$. We consider the partition $V_0, \ldots, V_k$ to be fixed in this exercise.

Proof. From the previous definition of the layering, we know that any flow f cannot pass the partition forwarding. This means that flow cannot go from V_i to V_j $(j > i + 1)$ directly. For $m = |E|$, there is at most m edges from V_i to V_{i+1}. Therefore, after at most m iterations, if more flows wat to pass the partition, they need to go back from V_{i+1} to V_j $(j \leq i + 1)$. This makes $dist(s, t) > k$ and $V_0, \cdots, V_k$ no long an optimal layering, which leads to a contradiction.

□

Exercise 10.9. Show that the Edmonds-Karp algorithm terminates after $n \cdot m$ iterations of the while-loop. **Hint.** Initially, compute an optimal k-layering (which?). Then keep this layering as long as its optimal. Once it ceases to be optimal, compute a new optimal layering. Note that the Edmonds-Karp algorithm does not actually need to compute any layering. It's us who compute it to show that $n \cdot m$ bound on the number of iterations.

Proof. Since Edmonds-Karp Algorithm is a special case of Ford-Fulkerson Algorithm, from Exercise 8 we know that after at most m iterations of the

while-loop, $V_0, \cdots, V_k$ will cease to be an optimal layering. That is, $dist(s,t)$ is no longer k.

As shown above, Edmonds-Karp Algorithm always choose p to be the shortest path from s to t. Then $dist(s,t) = k'$ is now at least $k+1$. We have $1 \leq k \leq n$, we can find a new optimal layering at most n times.

Hence the Edmonds-Karp Algorithm terminates after $n \cdot m$ iterations of while-loop. □

Exercise 10.10. Show that every network has a maximum flow f. That is, a flow f such that $\text{val}(f) \geq \text{val}(f')$ for every flow f'. **Remark.** This sounds obvious but it is not. In fact, there might be an infinite sequence of flows $f_1, f_2, f_3, \ldots$ of increasing value that does not reach any maximum. Use the previous exercises!

Proof. Suppose that some network don't have a maximum. Then, there must exist one augmenting path from s to t. Then we can use Edmonds-Karp algorithm to deal with this situation. From previous exercise, we have proved that this algorithm terminates after limited number of iterations. When it terminates, we successfully get the maximum flow f. □