Mathematical Foundations of Computer Science

CS 499, Shanghai Jiaotong University, Dominik Scheder

10 Network Flow

- Homework assignment published on Monday 2018-05-07
- Submit questions and first solution by Sunday, 2018-05-13, 12:00
- Submit final solution by Sunday, 2018-05-20.

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

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

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.

First, we use e to denote an edge, and it is obvious that for each $v \in V - s, t$:

$$\sum_{e\, into\, v} f(e) = \sum_{e\, out\, of\, v} f(e)$$

We call it flow conservation, and with the help of it we can know that:

$$\text{val}(f) = \sum_{e\,out\,of\,s} f(e) = \sum_{v \in S}(\sum_{e\,out\,of\,u} f(e) - \sum_{e\,into\,v} f(e)) = \sum_{e\,out\,of\,A} f(e) - \sum_{e\,into\,A} f(e)$$

As for the flow, if it is from S to $V - S$, it is a positive one, otherwise it is a negative one. Namely, the formula above it is

$$\sum_{u \in S, v \in V \setminus S} f(u, v)$$

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.

We use the water to embody the flow. If the water from s to r of value k is exactly that r to t, then obviously the statement holds. If there is some not flowing to t but to some other vertex, from the flow conservation we can know there must be some water flow into r with the same quantity, so there is also a flow from s to t.

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.

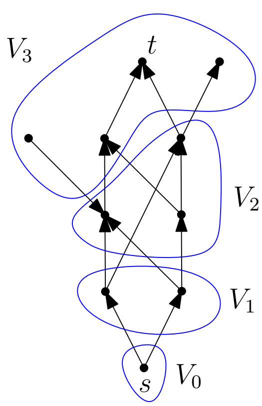
Algorithm 1 Ford-Fulkerson Method

procedure FF$(G = (V, E), s, t, c)$
    Initialize f to be the all-0-flow.
    while there is a path p form s to t in the residual network G_f **do**
        $c_{\min} := \min\{c_f(e) \mid e \in p\}$
        let f_p be the flow in G_f that routes $c_{\min}$ flow along p
        $f := f + f_p$
    end while
    // now f is a maximum flow
    $S := \{v \in V \mid G_f$ contains a path from s to $v\}$
    // S is a minimum cut
    return (f, S)
end procedure

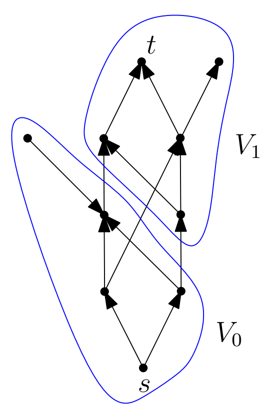
In a series of exercises, you will now show that this algorithm always terminates after at most $n \cdot m$ iterations of the while loop (here $n = |V|$ and $m = |E|$).

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.*

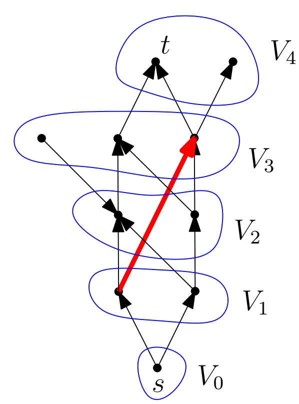
The figure below illustrates this concept: for one network we show two possible layerings and something that looks like a layering but is not:



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 most k edges.

Proof. Suppose s-t-path in G has m edges $(m < k)$, whose vertices are $(v_0, v_1, v_2, \ldots, v_m)$. Suppose v_i belongs to the layer $l(i)$, that is $v_i \in V_{l(i)}$. Obviously, l(0)=0 and l(m)=k. Since $(v_i, v_{i+1}) \in E$, $l(i+1) \leq l(i) + 1$. So $l(m) \leq l(m-1) + 1 \leq l(m-2) + 2 \leq \ldots \leq l(0) + m = m < k$, which contradicts $l(m) = k$. Therefore, every s-t-path in G has 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. Perform a breadth-first search on this graph until t is traversed. Since the path to each point obtained by breadth-first traversal is the shortest path, breadth-first tree has k+1 levels. Put the vertices in the same level in the tree to a layering successively, and vertices that have not yet been traversed are put into the last layering, too. According to the properties of breadth-first search, condition (1)(2)(3) is easy to meet. So we get 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 any edge (u, v) in path p in G, $u \in V_i$, $v \in V_j$, $j \leq i + 1$.
Since the length of the path is k, $j = i + 1$
In the residual graph G_f, (u, v) split to (u, v) and (v, u).
Obviously, (u, v) still satisfies the condition (3).
For (v, u), $i = j - 1 \leq j + 1$, thus it satisfies the condition (3), too.
Therefore, $V_0, V_1, \ldots, V_k$ is a layering of (G_f, s, t, c_f), too.

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

Case 1: When the provided path from s to t is the shortest-path with length k, according to **Exercies 10.5**, there is a k-layering. Thus, the network has an optimal layering.

Case 2: When the provided path from s to t is not the shortest-path, then there is another shortest-path from s to t with length k. According to **Exercies 10.5**, there is a k-layering. Thus, the network has an optimal layering.

Exercise 10.8. Imagine we are in some iteration of the while-loop of the Ford-Fulkerson method. 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.

Solution

After every iteration, an edge is completely reversed. Note that the reversed edge cannot appear in the shortest path again becauce it "goes back" to lower layer. There are total m edges, so after at most m iterations, all edges are reversed and there is no path from s to t. So $V_0, V_1, \ldots, V_n$ ceases to be an optimal layering.

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. First we construct an optimal layering by breath first search. We have to make sure that the $t \in V_k$, so whenever BFS reaches t, we should put t and the rest of vertices to V_k and stop BFS.

According to $ex.$10.8, after at most m iterations of Edmonds-Karp algorithm, the optimal k-layering ceases to be optimal, which means $dist_G(s, t) > k$. Then we will do BFS again to create a k'-layering, where $k' = dist_G(s, t) > k$. The maximum times we can add k by one is the number of vertices, that means we go through 0 layering to $n-1$ layering. So the algorithm terminates after $n \cdot m$ iterations.

$\square$

Exercise 10.10. Show that every network has a maximum flow f. That is, a flow f such that $\mathrm{val}(f) \geq \mathrm{val}(f')$ 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. According to ex10.9, we know that the Edmonds-Karp algorithm terminates after $n \cdot m$ iterations of the while-loop. At each iteration, the increase of flow is a finite value because capacity of edges is finite. The final flow is the sum of limited number of finite value, so it is reachable. $\square$