CS345: Algorithms - II
Assignment 3 Solutions

Aditi Goyal - 190057
Yatharth Goswami - 191178

November 14, 2021

I Problem 1 Solution

In this problem, we were asked to modify the Ford Fulkerson algorithm to achieve a polynomial running time in size of input - $\mathcal{O}(m^2 \log c_{max})$ and provide time complexity analysis of the modified algorithm. We start with the pseudocode of the modified algorithm. Algorithm 1 described below is the modified Ford-Fulkerson algorithm.

1.1 Notations

- Capacity of path → This represents the minimum of the capacities of all the edges present in a path.
- $G = (V, E)$ → This represents the original graph G, for which max-flow is to be calculated.
- $G_f = (V, E_f)$ → This represents the residual graph with respect to graph G and flow f.
- c_{max} → This represents the value of max capacity edge in the graph G.
- f_{max} → This represents the value of max flow in graph G.
- $c(x, y)$ → This represents the capacity of edge (x, y) in graph G.

1.2 Pseudocode for finding max flow

Algorithm 1: Algorithm to find max flow (Max-cap-FF algorithm)

Input: Graph G, source vertex s and sink t
Output: Max flow in the graph G, f

```
1  Function Max-cap-FF {G, s, t}:
2      f ← 0
3      while there exists a path in G_f do
4          Let P be a max capacity path in G_f
5          for each edge (x, y) ∈ P do
6              if (x, y) is a forward edge then
7                  f(x, y) ← f(x, y)+ capacity(P)
8              if (x, y) is a backward edge then
9                  f(y, x) ← f(y, x) - capacity(P)
10         Update G_f
11     return f
```

Algorithm 2: Algorithm to find max flow (Poly-FF algorithm)

Input: Graph G, source vertex s and sink t
Output: Max flow in the graph G, f

```
1  Function Poly-FF {G, s, t}:
2      f ← 0
3      c_max ← maximum capacity of any edge in G
4      k ← c_max
5      while k ≥ 1 do
6          while there exists a path of capacity ≥ k in G_f do
7              Let P be any path in G_f with capacity at least k
8              for each edge (x, y) ∈ P do
9                  if (x, y) is a forward edge then
10                     f(x, y) ← f(x, y)+ capacity(P)
11                 if (x, y) is a backward edge then
12                     f(y, x) ← f(y, x) - capacity(P)
13             Update G_f
14         k ← ⌈k/2⌉
15     return f
```

1.3 Proof of Correctness

We will first prove that for any graph G, the worst case number of augmenting paths used in the algorithm 1 is upper bounded by the worst case number of augmenting paths used in the algorithm 2. For any graph G, consider the sequence of paths chosen by algorithm 1. We can choose exactly the same sequence of paths in algorithm 2 as well because at line 7 (in algorithm 2), we are guaranteed that there exists a path of capacity $\geq k$. Since, we can consider any path with capacity $\geq k$ so we consider the max capacity path which is chosen by algorithm 1. In this way, we can continue and choose the same path chosen by algorithm 1 at every stage. In the time complexity analysis, we will prove that number of augmenting paths in algorithm 2 is bounded by $\mathcal{O}(m \log_2 c_{max})$, thus, bounding the number of augmenting paths in algorithm 1 by $\mathcal{O}(m \log_2 c_{max})$.

Next, we will focus on proving the correctness of the Poly-FF algorithm. First we will show that the algorithm will terminate in finite number of iterations. Each iteration of the while loop at line 6 strictly increases the flow in the graph G (since source only has forward edges and hence flow from source will always increase). Also, the total flow in the graph is bounded by max flow, which is finite (value of min-cut), so the algorithm will terminate in finite number of steps.

Next, we will show that this algorithm will return max-flow in the graph G. The Ford-Fulkerson algorithm terminates when there is no $s-t$ path in residual graph G_f. If we show that on termination of Poly-FF algorithm, no $s-t$ path is present in G_f, the correctness of this algorithm follows from max-flow min-cut theorem. A sketch of the proof is as follows, we will consider the $s-t$ cut defined by the set of nodes which are reachable from s in residual graph (call set A) on termination

of Poly-FF algorithm and prove that the flow along this cut will be equal to it's capacity. Similar to the proof shown in class, every outgoing edge from the set A, will carry flow equal to it's capacity as if not we will reach a contradiction of having selected every reachable node from source s to set A. Similarly, all the incoming edges to set A will carry flow of 0, since otherwise there will be a backward outgoing edge in the residual graph and hence we again arrive at contradiction of having selected every reachable node from source s to set A. Now, since every flow is less than or equal to capacity of any cut and we found a flow equal to capacity of a cut, by maxflow-mincut theorem we get that the flow is max-flow in graph G.

Now, we show that on termination of Poly-FF, no $s-t$ path is present in G_f. In the last iteration of loop at line 5, $k = 1$. The while loop at line 6 terminates when there is no $s-t$ path of capacity ≥ 1. Since capacities are integers, any $s-t$ path has capacity ≥ 1. Saying no $s-t$ path of capacity ≥ 1 is present is equivalent to saying no $s-t$ path is present.

Hence, the algorithm runs in finitely many steps and returns the correct value of max-flow, hence the correctness.

1.4 Time complexity analysis

Claim 1.4.1. *The while loop at line 5 (outer while loop) executes for $\mathcal{O}(\log_2 c_{max})$ times only.*

Proof. k is set to c_{max} initially and at line 14, we halve k. So, while loop at line 5 gets executed for $\mathcal{O}(\log_2 c_{max})$ iterations. □

Claim 1.4.2. *Let f be the value of flow after the end of all iterations of inner while loop for a particular value of k, say k_0. Then, we have the relation*

$$f \geq f_{max} - mk_0$$

where f_{max} is the value of maximum $(s-t)$ flow in graph $G = (V, E)$.

Proof. Consider the situation when all iterations of inner while loop for a particular value of k, say k_0 have executed. Define graph G'_f as $G'_f = (V, E'_f)$, where E'_f consists of those edges which have a weight $\geq k_0$ in G_f. Define set A as set of all node reachable from s in G'_f. Note that $t \notin A$ because otherwise there would have been a path in G'_f with all the edges on the path having weight $\geq k_0$. This path would also be present in G_f because $E'_f \subseteq E_f$ (refer to Notations sub-section for E_f). This contradicts our assumption that all iterations of inner while loop are done because still a path with capacity $\geq k_0$ is present in G_f. Define $A' = V - A$. Consider the $s-t$ cut (A, A'). We will show that

$$f \geq \text{capacity}(A) - mk_0$$

For showing this, we have that

$$\begin{aligned} f &= f_{out}(A) - f_{in}(A) \\ &= \sum_{\substack{(x,y)\in E \\ x\in A, y\in A'}} f(x,y) - \sum_{\substack{(x,y)\in E \\ x\in A', y\in A}} f(x,y) \end{aligned}$$

We will first show a bound for the flow along any outgoing edge from set A to set A' in G. For any edge (x,y) in G, where $x \in A$ and $y \in A'$, $f(x,y) > c(x,y) - k_0$. Assume that this is not the case, i.e. $f(x,y) \leq c(x,y) - k_0$. Then, in G_f, we will have a forward edge (x,y) with weight $= c(x,y) - f(x,y) \geq k_0 \implies (x,y) \in E'_f$. Also, by our construction of set A, x is reachable from s in G'_f and $(x,y) \in E'_f$ implies y is reachable from s in G'_f. This is a contradiction because y should have been in A, by definition.

We will next show a bound for the flow along any incoming edge to set A from set A' in G. For any edge (x,y) in G, where $x \in A'$ and $y \in A$, $f(x,y) < k_0$. Assume that this is not the case, i.e. $f(x,y) \geq k_0$. Then, in G_f, we will have a backward edge (y,x) with weight $f(x,y) \geq k_0 \implies (y,x) \in E'_f$. Also, by our construction of set A, y is reachable from s in G'_f and $(y,x) \in E'_f$ implies x is reachable from s in G'_f. This is a contradiction because x should have been in A, by definition.

We have that

$$f = \sum_{\substack{(x,y)\in E \\ x\in A, y\in A'}} f(x,y) - \sum_{\substack{(x,y)\in E \\ x\in A', y\in A}} f(x,y)$$

From the bound proved for outgoing edges, we have

$$f \geq \sum_{\substack{(x,y)\in E \\ x\in A, y\in A'}} (c(x,y) - k_0) - \sum_{\substack{(x,y)\in E \\ x\in A', y\in A}} f(x,y)$$

From the bound proved for incoming edges, we have

$$f \geq \sum_{\substack{(x,y)\in E \\ x\in A, y\in A'}} (c(x,y) - k_0) - \sum_{\substack{(x,y)\in E \\ x\in A', y\in A}} k_0$$

Thus,

$$f \geq \sum_{\substack{(x,y)\in E \\ x\in A, y\in A'}} c(x,y) - k_0(m_{incA} + m_{outA})$$

where m_{incA} is the number of incoming edges to A and m_{outA} is the number of outgoing edges from A. We have that $m_{incA} + m_{outA} \leq m$. Now,

$$\sum_{\substack{(x,y)\in E \\ x\in A, y\in A'}} c(x,y) = capacity(A)$$

and by maxflow-mincut theorem, $capacity(A) \geq f_{max}$. Thus, we have

$$f \geq f_{max} - mk_0$$

□

Claim 1.4.3. *For any fixed value of k, say k_0 the inner while loop executes for $\mathcal{O}(m)$ iterations only.*

Proof. Note that before the start of the iteration of outer while loop with $k = k_0$, the value of k would have been $2k_0$ in the previous iteration and hence from **claim** 1.4.2, we have that $f_{init} \geq f_{max} - 2mk_0$ where f_{init} is the flow before execution for $k = k_0$. Also note that when the inner while loop for $k = k_0$ terminates we have the condition that $f_{end} \leq f_{max}$, where f_{end} is the flow after execution for $k = k_0$. Noticing the fact that each iteration of inner while loop increases the value of flow by at least k_0 (since we are choosing paths with capacity at least k_0), there cannot be more than $2m$ number of iterations of inner while loop for $k = k_0$, as otherwise we would have $f > f_{max}$. Hence proved at for each value of k, the number of iterations of the inner while loop are $\mathcal{O}(m)$. □

Claim 1.4.4. *The time complexity of the Poly-FF algorithm is $\mathcal{O}(m^2 \log c_{max})$*

Proof. Let us analyze the time complexities of different parts of the algorithm.

- Lines 2-4 can be done in $\mathcal{O}(m)$ time since we need to calculate the maximum capacity among all edges which requires iterating over all the edges.
- The outer while loop (line 5) runs for $\mathcal{O}(\log_2 c_{max})$ iterations as shown above.
- The inner while loop (line 6) runs for $\mathcal{O}(m)$ iterations for any value of k.
- Each iteration of the inner while loop (line 6) involves two things. First is finding a path P with capacity above a certain threshold and then iterating over the edges in path P and updating the edges of G_f using edges in path P. The first step of finding a path can be done by creating a new graph G' with edges having capacity greater than or equal to the threshold from the residual graph and then selecting a path in this graph using either BFS or DFS traversal. The second step is just updating the capacities and flow along them by iterating over them. All of this is $\mathcal{O}(m)$ in time complexity.

Since the inner while loop runs for $\mathcal{O}(m)$ iterations (for a particular value of k), each of which takes $\mathcal{O}(m)$ in total, we have the running time of inner while loop as $\mathcal{O}(m^2)$. And since the outer loop runs for $\mathcal{O}(\log_2 c_{max})$ iterations, each of which runs the inner while loop once we have that the overall complexity of the algorithm becomes $\mathcal{O}(m^2 \log_2 c_{max})$. Combining the complexity of lines 2-4, still leads to time complexity being $\mathcal{O}(m^2 \log_2 c_{max})$. Hence shown that the Poly-FF algorithm results in the desired time complexity. □

Claim 1.4.5. *The number of augmenting paths used in algorithm 1 is upper bounded by $\mathcal{O}(m \log_2(c_{max}))$.*

Proof. We will use the fact that the number of augmenting paths used by algorithm 1 is upper bounded by the number of augmenting paths used by algorithm 2 (Refer Poof of Correctness sub section). Notice that in algorithm 2, each execution of inner while loop uses exactly 1 augmenting path and for any value of k, there are at most $\mathcal{O}(m)$ iterations of inner while loop, this implies that for each iteration of outer while loop, we see $\mathcal{O}(m)$ augmenting paths. Since, outer while loop executes for $\log_2(c_{max})$ iterations, we have total augmenting paths seen by algorithm 2 to be $\mathcal{O}(m \log_2(c_{max}))$. Using the fact stated at the start of the proof, we have that number of augmenting paths used by algorithm 1 is upper bounded by $\mathcal{O}(m \log_2(c_{max}))$. □

Claim 1.4.6. *Algorithm 1 runs in polynomial time.*

Proof. Algorithm 1 is a polynomial time running algorithm in input size because total number of iterations of the while loop (equal to the number of augmenting paths used) are polynomial ($\mathcal{O}(m \log_2(c_{max}))$) as proved above and in each iteration, we need to find max capacity path which can be done by Dijkstra's algorithm, which also takes polynomial time ($\mathcal{O}(m + n \log n)$). Other operations in the while loop are also polynomial: updating the flow and residual network can also be achieved in polynomial time $\mathcal{O}(m)$. □

Hence shown that choosing max capacity path in every iteration of ford-fulkerson algorithm runs in polynomial time in size of inputs.