CS345A Assignment 6

Abhay 180014 Abhishek Mittal 180022

10 November 2020

1 Problem Statement

Ford-Fulkerson algorithm in polynomial time for integer capacity

In Lecture 20, we showed a graph with integer capacities on which the Ford-Fulkerson algorithm can be made to run in $\Theta(\mathrm{m}c_{max})$ time, where c_{max} is the maximum capacity of any edge in G.

The objective of this assignment problem is to slightly modify the algorithm so that it runs in polynomial time for all integer capacity graphs. In fact, the intuition underlying the modified algorithm is based on the graph that we discussed in Lecture 20.

Spend sufficient time to ponder over the slight modification in the Ford-Fulkerson algorithm.

2 Solution

2.1 Algorithm

In order to attain a polynomial time complexity in the ford-fulkerson method, we must change the ford-fulkerson algorithm slightly. If we choose augmenting paths more carefully then the time-complexity can improve. Hence, the change we suggest is as follows :

"In each iteration, pick the path with maximum capacity in the residual network G_f and use it to increase the flow in G during that iteration."
Now, we need to show that after this modification, the FF algorithm shall use O(mlog$_2 c_{max}$) augmenting paths to compute maximum flow from s to t. Let us consider the algorithm given below :

Algorithm 1 : Poly-FF(G, s, t)

```
$f \leftarrow 0$ ;
$k \leftarrow$ maximum capacity of any edge in G;
while $k \geq 1$ do
    while there exists a (s-t) path of capacity $\geq k$ in $G_f$ do
        Let P be any simple path in $G_f$ with capacity atleast k;
        Let c be the bottleneck capacity of this path P;
        for each edge (x, y) $\in P$ do
            if (x, y) is a forward edge then f(x, y) $\leftarrow$ f(x, y) + c
            if (x, y) is a backward edge then f(x, y) $\leftarrow$ f(x, y) - c
        end for
    end while
    $k \leftarrow k/2$;
end while
return f ;
```

Now, we shall show that the time complexity of the algorithm in which we choose maximum capacity path (i.e path with maximum bottleneck capacity) is upper bounded by the time complexity of Poly-FF algorithm. It is important to note here that for a particular value of variable k, say k_0, we are choosing "any" path with edge capacities greater than equal to k_0. Now, suppose in each iteration of the outer while loop (i.e for a particular value of variable k), if we choose paths in the decreasing order of bottleneck capacity. Moreover, suppose if there are two paths with same maximum capacity then we would choose them in the order we are choosing in the max. capacity algorithm to make them same. Then, essentially in each iteration of the inner

while loop, we would be choosing same path as in max. capacity algorithm. Thus, we can say that the maximum capacity path augmentation algorithm is a special case of the Poly-FF algorithm. Thus, we can also say that the worst case number of augmenting paths used by max. capacity algorithm would be upper bounded by the worst case number of augmenting paths used by Poly-FF algorithm.
Hence, it is sufficient to show that Poly-FF algorithm uses $O(mlog_2c_{max})$ augmenting paths.

Note that in the given Poly-FF algorithm, we have only changed the way of choosing augmenting paths. Hence, all the properties in normal Ford-Fulkerson algorithm for integer capacity edges shall hold true i.e in Poly-FF algorithm, the flow and residual capacities shall remain integer-valued for integer edge capacities.

Lemma 1 : *The outer while loop of the algorithm shall run for O(log_2c_{max}) times only.*

Proof The value of k at beginning of the while loop is equal to c_{max} i.e maximum capacity edge in graph G and in each iteration of the outer while loop, we are dividing k by 2 due to which it will run for at most 1 + $[log_2c_{max}]$ times. Hence, the outer loop runs for O(log_2c_{max}) times.

Lemma 2 : *Consider the beginning of the iteration of the outermost While loop for any value, say k_0, of variable k. If f is the current value of the (s, t)-flow in G, then f $\geq$ f_{max} - 2mk_0, where f_{max} is the maximum (s, t)-flow in G.*

Proof When $k_0 = c_{max}$ (i.e for the first iteration of the outer while loop), the lemma is easy to prove. It is because the maximum flow which is possible can be mk_0 and f = 0 in the beginning. Hence, $(f = 0) \geq$ f$_{max} - 2mk_0 = mk_0 - 2mk_0 = -mk_0$.
Now, let us consider the iteration ending before the beginning of the iteration mentioned in lemma (i.e when $k = 2k_0$). Now, let us consider a cut in G_f, say A and B = V - A, such that A is the set of vertices reachable from s with paths with edge capacities $\geq 2k_0$.

- Now, consider an edge e = (u, v) in G for which u $\in$ A and v $\in$ B (i.e edge going out of A). We claim that the capacity of this edge(c_e) < flow through this edge(f(e)) + $2k_0$. If it wasn't true, then we could have reached vertex v from s since the capacity of this edge would then be greater than equal to $2k_0$ which would contradict our assumption that v lies in B.

- Similarly, consider any edge e′ = (u′, v′) in G such that u′ ∈ B and v′ ∈ A (i.e e′ is an edge coming into A). We claim that f(e′) < $2k_0$. If it wasn't true, then we would get a backward edge from A to B with edge capacity greater than equal to $2k_0$ (i.e u′ would be reachable from s and would thus lie in A) which would contradict our assumption that u' lies in B.

Now, we have,

$$\begin{aligned} f(totalflow) &= \sum f_{out}(A) - \sum f_{in}(A) \\ &\geq \sum_{\text{e out of A}} (c_e - 2k_0) - \sum_{\text{e into A}} (2k_0) \qquad \text{(From above points)} \\ &= \sum_{\text{e out of A}} c_e - \sum_{\text{total edges in cut A}} 2k_0 \\ &\geq c(A,B) - 2mk_0 \qquad \text{(Total edges in G is m)} \\ &\geq f_{max} - 2mk_0 \qquad \text{(Max-flow is bounded by capacity of any s-t cut)} \end{aligned}$$

Hence, proved

Lemma 3 : *Consider the beginning of the iteration of the outermost While loop for any value, say k_0, of variable k. In each iteration of the inner while loop for a given value k_0 of k, the flow increases by atleast k_0.*

Proof We know that total flow in the graph is equal to the flow coming out from the source s. In the algorithm, we consider a simple path P with edge capacities $\geq k_0$ (for a given iteration of outer while loop when k=k_0) and increase the flow on each edge of the path P by the bottleneck capacity. Since, each edge capacity $\geq k_0$, bottleneck capacity is also $\geq k_0$.

The first edge of P must be an edge coming out from s. Also, since the path is simple, there is no edge going to s again. Thus, this first edge in path P must be a forward edge on which we increase flow by bottleneck capacity ($\geq k_0$). Hence, on each iteration of inner while loop, we increase flow by k_0.

Lemma 4 : *The number of iterations of inner while loop in the algorithm is O(m) only.*

Proof From lemma 3, we have proved that on each iteration of inner while loop, flow increases by at least k_0. From lemma 2, we have proved that the maximum (s-t) flow, $f_{max} \leq f + 2mk_0$. Hence, it takes at most 2m iterations of the inner while loop to reach maximum flow. Hence, the number of iterations is O(m) for given value of variable k.

Theorem : The running time of Poly-FF algorithm is $O(m^2 log_2 c_{max})$.
Proof The Lemma 1 shows that the outermost while loop runs for at most O($log_2 c_{max}$) times. We have proved in Lemma 4 that the number of iterations of the inner while loop are O(m) only. Moreover, in each iteration of inner while loop, we are iterating in all the edges of the choosen path P and choose such a simple path P with either DFS or BFS which takes O(m + n) time. Here, we assume that our network is a connected graph. Thus, m $\geq$ n which makes it O(m) time only. Hence, the total time taken in the algorithm is O($m^2 log_2 c_{max}$).