CS345

Algorithms -II

Indian Institute of Technology, Kanpur

Anshit Arya (190158), Saket Dhakar (190739)

Assignment 3

Question 1

Solution

We need to modify the algorithm such that it will run in polynomial time for all integer capacity graphs. We will introduce the following modification in the Ford-Fulkerson algorithm - "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"

The **Modified Ford-Fulkerson** algorithm is-

```
Modified_FF( G , s , t ){
    f = 0 //for all edges in G
    While( there is a s-t path in Gf ) {
        Pick path P in Gf with maximum capacity as c
                    // Path P have maximum capacity in Gf
        For each edge e(x,y) in P{
            if (x,y) is a forward edge then:
                f(x,y)=f(x,y)+c;
            else:
                f(y,x)=f(y,x)-c;
        }
    }
}
```

Normally we were just arbitrarily choosing the path but after the modifications we are choosing path P in decreasing order of maximum capacity so correctness of algorithm also holds for **Modified_FF**. Since we have only changed the order of choosing the path P so the max-flow remains the same as the cut obtained after the end is same.
Now we need to prove that the **Modified_FF** runs in polynomial time.

To do so we will consider algorithm **Poly-FF** and then we will show that the number of augmenting paths used by **Modified _ FF** in the worst case is upper bounded by the number of paths used by **Poly-FF**. And then we will prove that the **Poly-FF** uses O(m log C_{max}) augmenting paths in the worst case.

Lets complete the **Poly-FF** algorithm given in the question.

```
Poly-FF( G , s , t ){
    f = 0 // for all edges in G
    k = maximum capacity of any edge in G;
    While( k >= 1 ){
        While( there exists a path of capacity >= k in Gf ){
            Let P be any path in Gf with capacity c where c>=k;
            for each edge e(x,y) in P do{
                if (x,y) is a forward edge then:
                    f(x,y)=f(x,y)+c;
                else:
                    f(y,x)=f(y,x)-c;
            }
        }
        k = k/2;
    }
}
```

Now we will modify the **Modified _ FF** algorithm to make it similar to the **Poly-FF** algorithm since we want to make comparisons between them. We will add the variable k in the **Modified _ FF** algorithm.

```
New_Modified_FF( G , s , t ){
    f = 0 // for all edges in G
    k = maximum capacity of any edge in G;
    While( k >= 1 ){
        While( there exists a path of capacity >= k in Gf ){
            Pick path P in Gf with maximum capacity as c
            for each edge e(x,y) in P do{
                if (x,y) is a forward edge then:
                    f(x,y)=f(x,y)+c;
                else:
                    f(y,x)=f(y,x)-c;
            }
        }
        k = k/2;
    }
}
```

This new algorithm is same as Modified_FF. We have just added a extra while loop on k which will run till k>=1 and in that while loop we will pick the path with maximum capacity as we were choosing in the modified_FF. Now at any iteration if there are paths with capacity >= k then we will choose maximum of them (in Modified_FF we were choosing the maximum of all the paths) and if there is no such path then while loop will break and k will be reduced to k/2. So we can say that they both have same augmenting paths.
We will now try to relate New_Modified_FF with Poly-FF. In the inner while loop the New_Modified_FF chooses the path with maximum capacity while the Poly-FF chooses any arbitrary path satisfying capacity>=k. So the way New_Modified_FF chooses path in a order is a subset of the possible sequences of path that is used by Poly-FF. Also the outer while loop is same for both the algorithms and just defines a lower bound on the capacity. So the worst case number of

augmenting paths used by New_Modified_FF is upper bounded by the number of paths used by Poly-FF algorithm.
Since time taken by Modified_FF to process a single augmenting path is similar to the time taken by New_Modified_FF we can say that runtime for the Modified_FF algorithm is also upper bounded by the runtime for the Poly-FF algorithm, since it uses fewer number of augmentations

Proof that POLY-FF uses $O(m\ log_2 c_{max})$ augmenting paths:
The number of iterations for outer while loops are $O(log_2 c_{max})$ because k varies from c_{max} to 1, each time getting halved. We have to show tat number of iterations of inner while loop = $O(m)$. Let the value of the flow be f and k during ith iteration be k'. Till now, we have considered those paths which had a capacity $\geq 2k'$, that means that even if we don't take paths with capacity $< 2k'$, the algorithm will remain the same. Consider a different graph H obtained from G in which there are no edges which had a capacity $< 2k'$ and check its residual graph H_f produced by Ford Fulkerson, the set of vertices can be divided into 2 sets:
the ones reachable from s in a set A and
the ones non-reachable in $\bar{\mathbb{A}}$.
A and $\bar{\mathbb{A}}$ form an $s - t$ cut in H, we can also run $i - 1$ iterations of POLY-FF on H to get a flow equal to f as it does not depend on the edges with a capacity $< 2k'$.
Now, we will again convert graph H to G and consider the cut between A and $\bar{\mathbb{A}}$, we now have to consider the maximum extra flow we can get from A to $\bar{\mathbb{A}}$, There are atmost m such edges with capacity $< 2k'$. So, we can get a constraint for the maximum flow f_{max} considering all edges as follows:

$$f \geq f_{max} - 2mk'$$

Now takiing inner while loop in consideration, for every iteration the total flow can increase in the range $[k', 2k')$, so we get maximum of 2m iterations for inner while loop, so order of iteration of inner while loop is $O(m)$.
⇒ maximum number of augmenting paths used by POLY-FF = $O(m\ log_2 C_{max})$

We know that maximum number of augmenting paths used by Poly-FF is $O(m\ log_2 C_{max})$ then the maximum number of augmenting paths used by Modified_FF algorithm is also $O(m\ log_2 C_{max})$. Now finding a augmenting path and sending the flow through a path both takes O(m) time so the overall complexity of the algorithm becomes $O(m^2\ log_2 C_{max})$