Ford-Fulkerson algorithm in polynomial time for integer capacity

1 Brief Description

The Ford-Fulkerson(FF) Algorithm discussed in class is not polynomial in the input size for a graph with integer capacites. Its time complexity was shown to be $\mathcal{O}(mC_{max})$, where m is the no. of edges and C_{max} is the maximum capacity. The intuition behind designing a polynomial time algorithm was to choose the path with maximum bottleneck capacity in each iteration from the graph G_f as discussed in class.
In this assignment we will be showing that the no. of augmenting paths in this Modified FF Algorithm is $\mathcal{O}(mlog_2(C_{max}))$. And the time comlexity is $\mathcal{O}(m^2log_2(C_{max}))$.

2 PseudoCodes

Algorithm 1 Modified-FF(G, s, t)

$f \leftarrow 0$
while there is an s-t path in G_f **do**
    Let P be the path with maximum capacity in G_f
    Let c' be the bottleneck capacity of 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**
            $f(y, x) \leftarrow f(y, x) - c'$
    $f \leftarrow f + c'$
    Update G_f accordingly.
return f

Algorithm 2 Poly-FF(G, s, t)

```
f ← 0
k ← maximum capacity of any edge in G
while k ≥ 1 do
    while there exists a path of capacity ≥ k in G_f do
        Let P be any path in G_f with capacity at least k
        Let c' be the bottleneck capacity of path P
        for each edge (x, y) in P do
            if (x, y) is a forward edge then
                f(x, y) ← f(x, y) + c'
            if (x, y) is a backward edge then
                f(y, x) ← f(y, x) − c'
        f ← f + c'
        Update G_f accordingly
    k ← k/2
return f
```

3 Proof of correctness

Our algorithm 1, Modified-FF(G, s, t) is a mere modification of the ford fulkerson algorithm. The only difference is in the order we choose the paths. Paths were chosen arbitrarily in ford fulkerson, but here, we are choosing the paths in a more clever fashion i.e. in a particular order. We proved the correctness for ford fulkerson algorithm in the class lectures. The cut at the termination of this algorithm and that of ford fulkerson is exactly the same since we only differ in the order of choosing the augmenting paths. Therefore, correctness must hold for our algorithm also. Hence Modified-FF(G, s, t) correctly computes the maximum flow in the given graph G
Now, when our algorithm 2 ends, it has traversed all the paths from s to t in a particular order. This is evident from the code itself. Hence, the same cut can be used for algorithm 2 when it terminates i.e. a cut (A, A') where A consists of all the vertices reachable from s. Going along the same lines as that of lecture slides, we can easily establish the fact that our algorithm 2 indeed correctly computes the maximum flow. Since we have based the proofs of out algorithms 1 and 2 on ford fulkerson algorithm, it goes without saying that all the assumptions that were made while discussing ford fulkerson algorithm in class still holds i.e. we assume that the edge capacities are integers. Hence the integrality theorem of max flows hold.

Since the value of max flow is unique, all the algorithms 1,2 and ford fulkerson output the same result. Now, we present the following lemma that relates algorithm 1 with 2.

Lemma 1: Worst case no. of augmenting paths used in the Modified-FF Algorithm is upper bounded by the worst case no. of augmenting paths used in the Poly-FF Algorithm.

Proof. Let the sequence of paths chosen by Modified-FF Algorithm in increasing order of iterations be $P = (P_1, P_2, \ldots, P_l)$, where P_l is the last path chosen before termination of loop. □

Lemma 1.1: The sequence P described above is one of the ways to choose paths in Poly-FF Algorithm.

Proof. Loop invariant maintained by Poly-FF is that when inner while loop corresponding to $k = k_o$ breaks, no path with capacity more than k_o present at that iteration.

Need to show that this invariant is also satisfied by the sequence P.

Let P_i be the first path with capacity k $< c_{max}$ in P. Also since P is a sequence obtained from Modified-FF Algorithm, this implies that the path P_i is the maximum capacity path in G_f present at this iteration. Hence

it implies that inner loop in Poly-FF Algorithm corresponding to $k = c_{max}$ will break because there is no path available in G_f with capacity k $\geq c_{max}$. Now again consider P_j be the first path in X with capacity k $< c_{max}/2$ also since this is maximum capacity path in G_f present at this iteration this implies that inner while loop corresponding to $k = c_{max}/2$ will break because there is no path available in G_f with capacity k $\geq c_{max}/2$ and so on for other values $k = c_{max}/4, c_{max}/8, \ldots$.
Hence, P is one of the ways to choose path in Poly-FF Algorithm. Poly-FF Algorithm's loop invariant is also satisfied by the sequence P and hence the sequence P is just one of the ways to choose paths in Poly-FF Algorithm.

□

Thus, the worst case number of augmenting paths from s to t used in Modified-FF Algorithm is upper bounded by the worst case number of augmenting paths used in Poly-FF Algorithm because from the above lemma choosing path from Modified-FF Algorithm is a special case of choosing paths in Poly-FF Algorithm.

4 Time Complexity

We establish the time complexity with the help of these lemmas -

Lemma 2 - If f is the current value of the (s,t) flow in G, then $f_{max} \leq f + 2mk_0$ where f is the maximum (s,t) flow in G and k_0 is the value of k at the beginning of the iteration of the outermost while loop.

Proof - Assume that after $i-1$ iterations i.e. at the beginning of i^{th} iteration of the outermost while loop, our algorithm has a value of $k = k_0$. Therefore, none of the paths at the beginning of i^{th} iteration have a capacity $\geq 2k_0$. Now, let us consider the beginning of the $(i-1)^{th}$ iteration. We choose paths of capacity at least $2k_0$.

So, if we remove edges that have capacity $< 2k_0$, the working of our algorithm remains the same. Let E' be the set of edges in E_f having capacity $< 2k_0$. Let us remove those edges and now we have the graph $G'_f = (V, E'_f = E_f \backslash E')$. Clearly the flow does not change even if we consider E'_f. Let us consider a cut (A, A') at the beginning of the i^{th} iteration i.e. after the end of $(i-1)^{th}$ iteration. Here A consists of all the vertices that are reachable from s having a path that has a capacity atleast $2k_0$. $A = A \cup s$. A' consists of rest all the vertices. Let f be the flow after the $(i-1)^{th}$ iteration i.e. at the beginning of the i^{th} iteration. Clearly, the cut is a valid cut since there exists no edge (u,v) in G_f such that $u \in A$ and $v \in A'$ and $f(u,v) >= 2k_0$. If there were any edge like that, then the while loop must have not terminated i.e. there must still exist a path from s to t having capacity at least $2k_0$. Hence a contradiction.

Now,

$$f_{max} = f_{out}(A) - f_{in}(A) = \sum_{eoutofA} f(e) - \sum_{eintoA} f(e)$$

Now, we claim that all outgoing edges must have $c(e) - f(e) < 2k_0$ satisfied. If it were not, then it means that there exists a path from A to A' having remaining capacity $\geq 2k_0$ which is a contradiction since we are looking at the end of the i^{th} iteration when none of the remaining paths have capacity $\leq 2k_0$.
Also, for all the incoming edges, $f(e) < 2k_0$ must be satisfied. If it were not the case, then there would be a backward edge from A to A'. Hence a contradiction.

Therefore,

$$f = f_{out}(A) - f_{in}(A) = \sum_{eoutofA} f(e) - \sum_{eintoA} f(e) \geq \sum_{eoutofA} (c(e) - 2k_0) - \sum_{eintoA} (2k_0)$$

$$\implies f \geq \sum_{eoutofA} c(e) - \sum_{eoutofA} (2k_0) - \sum_{eintoA} (2k_0)$$

Since the number of edges is bounded by m, therefore,

$$\sum_{eoutofA} (2k_0) + \sum_{eintoA} (2k_0) \leq 2mk_0$$

Also, $c(A, A') = \sum_{e out of A} c(e)$.

$$\implies f \geq c(A, A') - 2mk_0 \implies c(A, A') \leq f + 2mk_0$$

Now, the max flow min cut theorem states that $f_{max}(A, A') = min(c(A, A')) \implies f_{max} \leq c(A, A')$ Therefore,

$$f_{max} \leq f + 2mk_0$$

This was all at the end of the $(i-1)^{th}$ iteration i.e. at the beginning of the i^{th} iteration. Now in the i^{th} iteration, we will choose paths that have a capacity of atleast k_0. We also already have a flow of f and we cannot exceed the flow f_{max}. Therefore, we must not choose any more than $2m$ paths. If we chose more than $2m$, then we would trivially exceed f_{max} (since for each path, bottleneck capacity is $\geq k_o$) which is a contradiction. Therefore, the inner while loop must not run for more than $2m$ iterations i.e. it runs for $O(m)$ iterations.

Since we are halving the value of k in each iteration of the outermost loop with an initial value equal to the max capacity edge, it trivially runs for $O(log_2 C_{max})$ iteration.
Therefore, the no. of augmenting paths in Poly-FF Algorithm is $\mathcal{O}(log_2 C_{max)} * \mathcal{O}(m) = \mathcal{O}(mlog_2 C_{max})$.

Now, the innermost 'for' loop just searches for a path P and traverses it. This can be easily implemented using a BFS. BFS takes $O(m+n)$ time. Now, $m \geq n$ if the graph is connected. When the graph is not connected, we can simple remove the vertices that are not connected to s and t both. Therefore, we can easily assume $m \geq n$ i.e. BFS runs in $O(m)$ time.
Since our algorithm consisted of 2 nested while loops and one nester for loops, the complexity of our algorithm is trivially the product of the number of iterations each loop runs i.e. $O(log_2 C_{max} * m * m) = O(m^2 log_2 C_{max})$
Therefore, the algorithm 2 runs in time $O(m^2 log_2 C_{max})$.

Using lemma 1, Worst case no. of augmenting paths used in the Modified-FF Algorithm is upper bounded by the worst case no. of augmenting paths used in the Poly-FF Algorithm. The worst case no. of augmenting paths in Poly-FF algorithm is $O(mlog_2 C_{max})$.
Therefore, the worst case no. of augmenting paths used in Modified-FF algorithm is also $O(mlog_2 C_{max})$.
Also, for each augmenting path in Modified-FF Algorithm, the code inside the outermost loop runs in $\mathcal{O}(m)$ time(BFS + G_f updation). Therefore, the time complexity of our algorithm 1 is also $O(m^2 log_2 C_{max})$ which is polynomial in the input size.