Adapted From Virginia Williams'lecture notes.

COMP 285 (NC A&T, Spr '22) Lecture 34

Max Flows and Mininum Cuts II

1 Ford-Fulkerson

From this, we can construct an algorithm to find the maximum flow. Starting with some arbitrary flow of the graph, construct the residual network, and check if there is a path from s to t. If there is a path, update the flow, construct the new residual graph and repeat. Otherwise, we have found the max flow.

A path from s to t in the residual graph is called an augmenting path, and pushing flow through it to modify the current flow is referred to as augmenting along the path.

The run time of this algorithm is bounded by the number of times we update our flow. If the edge capacities are all integers, we can increase the flow by at least 1 each time we update our flow. Therefore, the runtime is $O(|f|m)$ where $|f|$ is the value of the max flow. If we have rational edge capacities, then we can multiply all edge capacities by a factor to make them all integers. However, the runtime blows up by that factor as well. If we have irrational edge capacities, then the algorithm is no longer guaranteed to terminate. So we have a problem.

Algorithm 1: maxflow(G, s, t)

```
f ← all zeros flow
G_f ← G
while t is reachable from s in G_f (check using DFS/BFS) do
    P ← path in G_f from s to t
    F ← min capacity on P
    f ← f' as defined in Lemma 5 above.
    Update G_f to the corresponding new flow.
return f
```

We will save the day in the next sections. Algorithm 1 is called the Ford-Fulkerson method.

It is actually part of a family of algorithms that depend on how the path P between s and t in G_f is selected. One can obtain P via DFS, BFS, or any other method for selecting paths. It turns out that two methods work particularly well: the shortest path method and the fattest path method. The shortest path method is known as the Edmonds-Karp algorithm or Dinic's algorithm.

The fattest path method. This method finds a path between s and t that maximizes $\min_{e \in P} c_f(e)$ among all $s-t$ paths P. Finding such a path can be done in $O(m+n)$ time by a clever mix of linear time median-finding and DFS.

The shortest path method (the Edmonds-Karp algorithm/Dinic's algorithm). This method picks the path between s and t using BFS, thus picking a path that minimizes the number of edges. Finding such a path also runs in $O(m)$ time: BFS takes $O(m+n)$ to explore the whole graph, but since we only care about the vertices reachable from s this is $O(m)$ time.

Since both methods of selecting a path run in linear time, the main question becomes, how many iterations does Ford-Fulkerson perform? We will answer these questions below in the next section.

2 Running time of various implementations of Ford-Fulkerson

Remark 1. *We did not discuss the details of this section in class, but it's in the notes for the interested reader.*

2.1 The fattest path version for Ford-Fulkerson

In this section we will show that the fattest path method results in a runtime of $O(m(m+n)log|f|)$ when run on a graph with integer capacities. Thus, when rational capacities are converted to integers by multiplying by N, we get a runtime of $O(m(m+n)(log|f|+logN))$ for rational capacities. Thus the effect of large N is mitigated. This method does not solve the issues when the capacities can be irrational.

To show the runtime, we prove a main claim that states that after each iteration of the algorithm, the maximum flow value in G_f goes down by a factor of $(1-1/m)$. This max flow value starts as $|f|$ since $G_f = G$ in the beginning of the algorithm, and ends at 0 as in the end s and t are disconnected.

Proposition 1. Let f' be the max flow in G_f . Then after one iteration of Ford-Fulkerson on G_f , the max flow value becomes $\leq |f'|(1-1/m)$.

Proof. Let P be the fattest path from s to t in G_f. Let $F = \min_{e\in P} c_f(e)$. Let S be the nodes reachable from s in G_f via paths composed of edges with residual capacities $> F$.

Thus, any edge (x, y) of G_f with $x \in tS, y \notin S$ must have $c_f(x, y) \leq F$. In particular, this means that the size of the cut between S and $V \setminus S$ is $\sum_{x\in S, y \int V\setminus S} c_f(x, y) \leq mF$. Thus, the size of the min s-t cut in G_f is at most mF.

By the max-flow-min-cut theorem from last lecture, the size of the min s-t cut is at least the size of the max-flow $|f'|$ in G_f , and so $|f'| \leq mF$. Thus $F \geq |f'|/m$. Now, when we augment (push flow) along P, the flow in G increases by F , while the flow in G_f decreases by F . Thus, the new flow in G_f after augmenting along P becomes $|f'| - F \leq |f'|(1-1/m)$. □

Now that the main claim has been proven, we can conclude with a discussion of the runtime.

Consider how the max flow value in G_f evolves after t iterations. It starts as $|f|$ (where f is the max flow in G) and then after t iterations is

$$\leq |f|(1-1/m)^t$$

.

If $t = m \ln |f|$, we get that the max flow value in G_f is

$$\leq |f|((1-1/m)^m)^{\ln |f|} < |f|(1/e)^{\ln |f|} = 1.$$

Since all the capacities are integers, all the residual capacities are also integers, and so the max flow value in G_f is an integer. Since it is < 1, it must be 0. Hence after $mln|f|$ iterations, the max flow value in G_f is zero, s and t are disconnected and the computed flow in G is maximum. The runtime is $O((m+n)mlog|f|)$.

2.2 The shortest path version of Ford-Fulkerson

Here we analyze running Ford-Fulkerson using BFS to find a path between s and t in G_f .

With each augmentation along a path P in G_f , at least one edge is removed from G_f , namely the edge with residual capacity $F = \min_{e \in P} cf(e)$. The main claim that we need to prove the runtime is that the number of times an edge can be removed from G_f is small. Since each iteration of the algorithm causes at least one removal, the main proposition will show that the number of iterations is small and hence the runtime is small as well.

Proposition 2. Fix any (u, v) that is ever an edge in G_f. Then the number of times that (u, v) can disappear from G_f is at most $n/2$.

Once this proposition is proven, we would get that the total number of edge disappearances is at most $mn/2$ and hence the number of iterations of the algorithm is also $\leq mn/2$. Because of this, the algorithm's runtime is $O((m+n)mn)$. To prove the proposition, we will need a useful lemma (see below) that shows that as G_f evolves through the iterations, for any v , the (unweighted) distance from s to v in G_f cannot go down.

Let's begin with some notation. LetG_f^i be the residual network after the i-th iteration of the algorithm; $G_f^0 = G$. For a vertex v , let $d_i(v)$ be the (unweighted) distance from s to v in G_f^i .

Lemma 1. *For all $i \geq 1$, and all $v \in V$, $d_{i-1}(v) \leq d_i(v)$.*

Proof. Fix i. We will prove the statement for i by induction on $d = d_i(v)$.

The inductive hypothesis is that for all d and all v with $d_i(v) = d$, $d_{i-1}(v) \leq d_i(v)$. The base case is $d = 0$. We note that if $d_i(v) = 0$, then $v = s$ since we view G_f^i as an unweighted graph. But then we also have $d_{i-1}(s) = 0 \leq d_i(s)$.

For the induction, let's assume that the inductive hypothesis holds for $d-1$, i.e. that for all x with $d_i(x) = d-1$, $d_{i-1}(x) \leq d_i(x)$. We want to show that for all v with $d_i(v) = d$, we also have $d_{i-1}(v) \leq d_i(v)$.

Consider some v with $d_i(v) = d$. Let u be the node just before v on a shortest $s - v$ path in G_f^i . Then, $d_i(u) = d_i(v) - 1 = d - 1$ and the inductive hypothesis applies to it so that $d_{i-1}(u) \leq d_i(u)$. We consider two cases.

Case 1. $(u, v) \in G_f^{i-1}$. Then, by the triangle inequality in G_f^{i-1} , we have that $d_{i-1}(v) \leq d_{i-1}(u) + 1$. Since $d_{i-1}(u) \leq d_i(u)$, we get that

$$d_{i-1}(v) \leq d_i(u) + 1 = (d_i(v) - 1) + 1 = d_i(v)$$

.

Case 2. $(u, v) \notin G_f^{i-1}$. Then, since $(u, v) \in G_f^i$, we must have that (v, u) was on the $(i - 1)$-st augmenting path. Hence $d_{i-1}(u) = d_{i-1}(v) + 1$. Hence:

$$d_{i-1}(v) = d_{i-1}(u) - 1 \leq d_i(u) - 1 = d_i(v) - 2 \leq d_i(v).$$

In both cases $d_{i-1}(v) \leq d_i(v)$ and the induction is complete. □

Now we are ready to prove the main proposition.

Fix some (u, v) that is an edge in G_f at some point. Let's consider two consecutive disappearances of (u, v). Suppose that $(u, v) \in G_i$ but $(u, v) \notin G_{i+1}$.

If after this disappearance (u, v) had another one later on, then at some point (u, v) must have appeared in G_f again. Let j be the first iteration after i so that the jth augmenting path made (u, v) appear in G_f^{j+1} . Because $(u, v) \in G_f^i$ but $(u, v) \notin G_f^{i+1}$, (u, v) must have been in the i-th augmenting path P_i . Because $(u, v) \notin G_f^j$ but $(u, v) \in G_f^{j+1}$, (v, u) must have been in the j-th augmenting path P_j .

From this we obtain that $d_i(v) = d_i(u) + 1$ and $d_j(u) = d_j(v) + 1$. Using the fact that $j > i$ and the key lemma from above we obtain

$$d_j(u) = d_j(v) + 1 \geq d_i(v) + 1 = d_i(u) + 2$$

.

Thus, between (u, v)'s disappearance and its next reappearance, the distance from s to u increased by $+2$. Hence between any two consecutive disappearances the distance to u increases by ≥ 2. The distance starts as ≥ 0 and can be $\leq n - 1$ before becoming ∞. Thus the total number of disappearances of $(u, v)$is $\leq n/2$.

This completes the proof of the main claim and the proof of the runtime.

3 Applications

We wrap up by talking about some applications of the Ford-Fulkerson algorithm.

3.1 Bipartite Perfect Matching

Let $G = (V, E)$ be an undirected, unweighted bipartite graph: the set of vertices is partitioned into V_1 and V_2 so that there are no edges with two endpoints entirely in V_1 or entirely in V_2. A matching in G is a collection of edges, no two of which share an end point. A perfect matching is a matching M such that every node in V has exactly one incident edge in M. In order for G to have a perfect matching, we need that $|V_1| = |V_2|$. The perfect matching problem is, given a bipartite graph G with $|V_1| = |V_2| = n$ and on m edges, determine whether G has a perfect matching.

We will solve the bipartite perfect matching problem by creating an instance of max flow and using Ford-Fulkerson's algorithm. Given $G = (V_1 \cup V_2, E)$, direct all the edges in E from V_1 to V_2. Add two extra nodes s and t. Add (directed) edges from s to every node in V_1 and from every node of V_2 to t. In this new graph H, let all the edge capacities be 1 and then run the Ford-Fulkerson algorithm to compute the max flow.

Suppose that G has a perfect matching M. Then, H has max flow value $n = |V_1| = |V_2|$. This is because we can set $f(e) = 1$ for every $e \in M$, ell the edges out of s and all the edges out of t. All other flow values are 0. The capacity constraints are trivially satisfied. The flow conservation constraints are satisfied since for every $x \in V_1$ there is exactly one edge (s, x) into x that has flow 1, and exactly one edge $(x, y) \in M$ with flow 1; similarly for every $x \in V_2$ there is exactly one edge (x, t) out of x that has flow 1, and exactly one edge $(y, x) \in M$ with flow 1.

Suppose now that Ford-Fulkerson returns a flow f of value n. Hence $f(s, x) = f(y, t) = 1$ for all $x \in V_1, y \in V_2$. Because Ford-Fulkerson causes all flow values on the edges to be integers, the flow values on all edges are either 1 or 0. Because of this, every node $x \in V_1$ gets flow of 1 going into it and a flow of 1 needs to come out so that there is a single edge (x, y) that has flow value 1 and all other edges out of x have flow value 0. Similarly, for every $y \in V_2$ there is a unique edge into y with positive flow value 1. The edges in $V_1 \cup V_2$ with positive flow through them must hence form a perfect matching.

3.2 More applications

There are many applications of max-flow and min-cut! We may talk about a few more in class if time (check the slides).