CSE 202 NOTES FOR NOVEMBER 19, 2002

Network Flows

Network flows are an important problem because they can be used in many reductions, a topic we will discuss shortly. The basic idea is that you need figure out how much stuff you can move between two nodes in a graph while obeying the intuitive conservation laws. More formally, the problem is listed below.

Instance: Symmetric, directed, weighted graph. That is, $(u, v) \in E \implies (v, u) \in E$, but $w(u, v) \neq w(v, u)$ in general. Two nodes, $s, t \in V$ are also given, where s is the source and t is the sink.

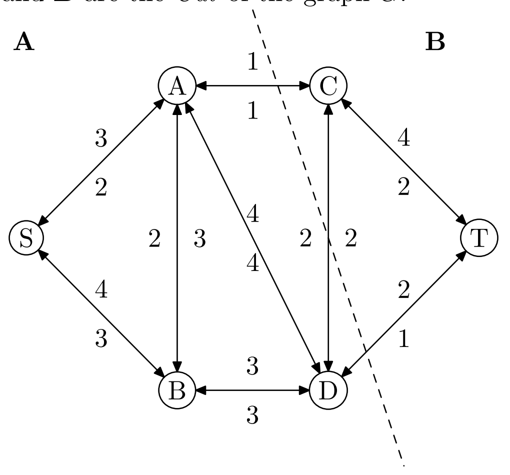
Solution Format: A sequence of paths and "flows" that determine the total quantity of "material" that moves from s to t. More specifically, the output is a set of pairings (e, f) where e is an edge in E and f is a real number indicating the flow of material on e. We will use $f(u, v)$ to denote the f associated with edge (u, v).

Constraints: $f(u, v) \leq w(u, v) \wedge f(u, v) = -f(v, u), \forall u, v \in V$. Further, $\forall u \in V - \{s, t\}, \sum_{v \in V} f(u, v) = 0$. Finally, $\sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t)$.

Objective: Maximize $\sum_{v \in V} f(s, v)$.

An example graph is shown in figure . We will describe the Ford-Fulkerson method for determining network flow, and introduce a set based on an idea that actually debuted in Dijkstra's algorithm for determining the Minimum Spanning Tree. We will also explore several properties of network flows that are determined through the Ford-Fulkerson method.

Figure 1. An example graph to illustrate the Ford-Fulkerson method. The edges that cross the borders between the regions labelled **A** and **B** are the *Cut* of the graph G.



The Ford-Fulkerson method proceeds by labelling all edges with a *current flow* of 0, and residual capacity of the edge weight . Next, it finds an arbitrary path with positive "residual capacity" (that is, locate $s \to \cdots \to v$ where for each (u, v) in the path, $f(u, v) \leq w(u, v)$. From this path, select the edge with the lowest residual capacity, and use this entire capacity for the entire path; modify the "residual capacities" of all the other edges in the path. For links going in the opposite direction, add the absolute value of the flow to it, and repeat the process.

As in the figure, you should see that any path between s and t must also go between regions **A** and **B**. The total capacity of all links crossing **A** and **B** is 5, so there couldn't possibly be a larger flow than 5. This forms the heart of the achieves-the-bound argument: the sum across all of the inbound links never gets larger than some number.

To formalize this, let's introduce some definitions.

Definition 0.1. Let S be a set of vertices, $s \in S, t \notin S$. Then $\text{Cut}(S) = \{(u, v) \in E : u \in S, v \notin S\}$. Also, let $\text{Cost}(\text{Cut}(S)) = \sum_{(u,v)\in\text{Cut}(S)} w(u, v)$. Also, a bit of notation, $f(p, q) = 0$ if $(p, q) \notin E$. Finally, let $\text{Flow}(f) = \sum_v f(s, v)$.

We will state and prove two important lemmas regarding Cuts and Costs.

Lemma 0.2. *$\forall S, s \in S \wedge t \notin S$, and any flow f,*

$$\sum_{v\in V} f(s, v) = \sum_{(u,v)\in Cut(S)} f(u, v)$$

Proof: Note that $\sum_{s'\neq s, s'\in S} \sum_v f(s', v)$, is 0 by construction: this double sum calculates the flow from any one node (that is not the starting point) to any other node in the graph. This must be 0 because we will always add $f(s', v)$ and $f(v, s')$ for all pairs of adjacent nodes, since we sum across all nodes in the vertex set; also keep in mind that we augmented our definition of f to include edges that were missing from E to be 0, so this will not affect our total. Keep in mind to avoid future confusion that s' is not related to s here, except by the relation that s' ***IS NOT*** s. So, we have:

$$\sum_v f(s, v) = \sum_v f(s, v) + \sum_{s'\neq s, s'\in S} \sum_v f(s', v) \tag{0.1}$$

$$= \sum_{s\in S} \sum_{v\in V} f(s, v) \tag{0.2}$$

$$= \sum_{s\in S} \sum_{s'\in S} f(s, s') + \sum_{s\in S} \sum_{v\notin S} f(s, v) \tag{0.3}$$

$$= \sum_{u\in S} \sum_{v\notin S} f(u, v) \tag{0.4}$$

$$= \sum_{(u,v)\in\text{Cut}(S)} f(u, v).\square \tag{0.5}$$

Where 0.1 is due to the above logic. 0.2 incorporates the first single sum into the double sum of the second term, 0.3 partitions this into two more meaningful pieces: the sum between pairs of things where both elements are inside S and the sum between pairs of things where one is in S and the other isn't. 0.4 can be concluded because as mentioned before the first term of 0.3 is 0, and we finally arrive at 0.5 by the definition of $\text{Cut}(S)$.

Our second lemma is central to the achieves-the-bounds argument.

Lemma 0.3. *Let S be any set as above and let f be any flow. Then $\mathit{Flow}(f) \leq \mathit{Cost}(\mathit{Cut}(S))$.*

Proof: By definition, $\text{Flow}(f) = \sum_v f(s,v)$. By the above lemma, this is $\sum_{(u,v)\in\text{Cut}(S)} f(u,v)$ and because we know that all $f(u,v) \leq w(u,v)$, we know that $\sum_{(u,v)\in\text{Cut}(S)} f(u,v) \leq \sum_{(u,v)\in\text{Cut}(S)} w(u,v) = \text{Cost}(\text{Cut}(S))$.

Finally, we state and prove the lemma that shows that the Ford-Fulkerson method actually achieves this bound that we've placed on the Cuts and the Costs.

Lemma 0.4. *Let F be the flow achieved by the Ford-Fulkerson method. Then $\exists S : \mathit{Flow}(F) = \mathit{Cost}(\mathit{Cut}(S))$.*

Proof: Define a new graph from G called G_F, where $G_F = \{(u,v) : w_r(u,v) - f(u,v) > 0\}$ (in other words, the graph of vertices and edges that can still accept more stuff). Let S be the set of all nodes that are reachable from s in G_F. We know that $t \notin S$ because if it were, then a path from s to t with positive flow would exist, and the Ford-Fulkerson method would have picked it. The set S, then, is the set of vertices in the lemma above. We know from lemma 0.2 that $\text{Flow}(F) = \sum_v f(s,v) = \sum_{(u,v)\in\text{Cut}(S)} F(u,v)$; we also know that for any $e = (u,v) \in \text{Cut}(S)$ that the residual capacity of the edge is 0 (because of the logic above involving G_F), so we have that $\sum_{(u,v)\in\text{Cut}(S)} F(u,v) = \sum_{(u,v)\in\text{Cut}(S)} w(u,v)$; therefore this particular cut achieves the bound and we have optimality. □

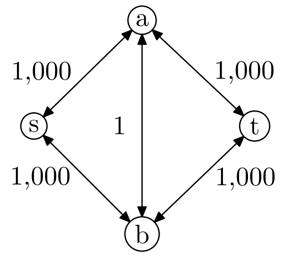
We have two corollaries regarding this algorithm. First, the maximum flow is equal to the cost of the minimum cut in a graph; in fact, the maximum flow happens across the minimum cut, so you can quickly find the minimum cut. Secondly, and possibly more importantly, if all of the capacities (that is, the weights) are integers, then there is an integral max flow.

Now that we have proved that the Ford-Fulkerson method is correct, I'll list it here slightly more completely.

```
while Positive paths can be found do
    Find a path in $G_R$, through Depth-First-Search, such that the total flow is
    positive.
    Augment the flow in the graph by the minimum residual capacity of an edge
    in the path you've chosen.
    Modify the residual capacities along the path.
end while
```

The first step may involve a depth-first-search, so it's $O(|E|)$. The second step may require adjusting everything in the path, and the path can be at most $|V|$ since we will never use loops in a flow, so it's $O(|V|)$. Modifying the residual capacities similarly takes at most $O(|V|)$ time, and we iterate at most $O(|E|)$ times. Can we bound the number of iterations, though? The answer is yes — we can bound it above by the flow in the following manner. Assume that all of the link capacities are integers for simplicity. $\text{Flow}(f)$ increases by at least one on each iteration, so the number of iterations is at most $\text{Flow}(f)$. Therefore, the total time of the algorithm is $O(|\text{Flow}|\,(|E| + |V|)) = O(|\text{Flow}|\,|E|)$.

Figure 2. The worst-case scenario for the Ford-Fulkerson algorithm. Note that the edge (a, b) flips its direction each time; because its capacity is so much smaller compared to the rest of the graph, it is a bottleneck.



Unfortunately, this gives us a perfect worst case scenario, shown in figure 2. In this case, it is possible that the Ford-Fulkerson method will choose the incorrect path every time that involves (a, b); at this point, the flow for (a, b) reverses. This clearly takes a really long time to run, and meets the $O(|\text{Flow}|\,|E|)$ bound.

There is another approach to solving the network flow problem that will eliminate this worst case, and it will involve a greedy heuristic on Ford-Fulkerson.

Greedy/Hill-climbing Network Flow

To remedy this worst-case situation, let's try to use a greedy heuristic to augment Ford-Fulkerson. Find the highest capacity augmented path, recalling that the capacity of a path is the smallest capacity of any link in the path. One particularly easy-to-think-about approach to finding the highest capacity augmented path is to delete edges in G until there is no path from s to t. If we sort the edges in increasing order of capacity and delete edges in order until a path no longer exists between s and t, then choose this path and augment and adjust the graph, we have a workable greedy solution, though we won't prove that it works. In this case, the running time is $O(|E|^2 + |E| \log |E|)$ to sort the edges and then iterate $O(|E|)$ times performing $O(|E|)$ work (that is, a depth-first-search) each iteration.

Perhaps instead of deleting paths until $s \not\rightsquigarrow t$ we can approach this the opposite way and add edges from the largest to the smallest until $s \rightsquigarrow t$. Is this any better? Not asymptotically, but the Professor claims that if we do the addition of paths that we can get this down to $O(|E|)$. In fact, if you think about this from the perspective of amortized analysis, you will see that it is true. Read on...

In doing a Depth-First-Search you are looking for nodes that are connectable from s, but you're really performing an additive operation. On each iteration you can only perform a Depth-First-Search to a particular set of nodes, say S; when you add another edge, you may get another set, S' that is strictly larger than S. In fact, if node $u \in S$, then by definition $u \in S'$, and $S \subset S'$. This holds for all sets of nodes that are reachable until all edges are added (at which point we have E). So, if you consider the amount of E that's added to the set at each iteration, you will see that across all iterations you will never visit any edge more than once (when it makes the transition from $E - S$ to S as $S \to E$).

So we figure that we can find the largest path in $O(|E|)$ time, though we have to sort the edges. If we use a red-black tree we can actually maintain sorted order, so we have $O(|E| + |V| \log |V|)$; I haven't thought about this too much so I'll just take it on faith. We now need to figure out how long it will take to perform the entire algorithm with this new bound, and it's not very interesting. It works out to $O(|E|^2 \log |E| \log |\text{Flow}|)$ by my notes. Thus, this new greedy-hill-climbing method is better than Ford-Fulkerson when $|\text{Flow}| > |E|$, but a bit worse otherwise.