CME 305: Discrete Mathematics and Algorithms

Instructor: Professor Aaron Sidford (sidford@stanford.edu)

January 22, 2018

Lecture 5 - The Maximum Flow Problem[1]

In this lecture we continue our discussion of the maximum flow problem. We provide algorithms, prove the maximum flow / minimum cut theorem, and begin to discuss applications.

1 The Maximum Flow Problem

Let's recall the problem from last time. We have directed graph $G = (V, E)$, a source $s \in V$, sink $t \in V$, and edge capacities $u \in \mathbb{R}^E_{\geq 0}$. Our goal is to compute $f \in \mathbb{R}^E_{\geq 0}$ that is a s-t flow, meaning for all $v \notin \{s, t\}$ we have that the net flow out of v or the imbalance at v is 0 i.e.

$$\text{im}(f, v) = \sum_{e=(v,u)\in E} f(e) - \sum_{e=(u,v)} f(e) = 0$$

and that is feasible, meaning it also meets the capacity constraints, $f(e) \in [0, u_e]$ for all $e \in E$. Moreover, we wish to compute a maximum s-t flow, meaning a feasible s-t flow of maximum value, $v(f) = \text{im}(f, s)$.

1.1 Some Notation

Recall our interpretation of $f(e)$ for $e = (u, v)$ is the amount of flow leaving u and entering v or contributing positively to $\text{im}(f, u)$ and negatively to $\text{im}(f, v)$. Consequently, when thinking about imbalance, it is natural to think of $f(e)$ also having $-f(e)$ units flowing from v to u. Thus, for notational convenience, from now we will overload notation and let

$$f(u, v) = \sum_{e=(u,v)\in E} f_e - \sum_{e=(v,u)\in E} f_e$$

denote the *net flow* from u to v. Note that $f(u, v) = -f(u, v)$, i.e. it is anti-symmetric.

To avoid thinking about multi-edges and having to be careful about forward v.s. backward flow we will often work with this notion of *net flow* $f \in \mathbb{R}^{V\times V}$ i.e. just a function on pairs. Note that if we define $u(u, v) = \sum_{e=(u,v)\in E} u_e$, with the convention that summing over the emptyset yields 0 then a s-t flow is feasible if and only if $f(u, v) \leq u(u, v)$ for all $u, v \in V$.

Note that this suggests we could also define flows just as values of vertex pairs. For $g \in \mathbb{R}^{V\times V}$ we call g a *s-t (net) flow* if it is *antisymmetric*, $g(u, v) = -g(v, u)$ for all $u, v \in V$ and for $\text{im}(g, v) = \sum_{u\in V} g(v, u)$ and $v(g) = \text{im}(g, s)$ then if $\text{im}(g, u) = 0$ for all $u \in V \setminus \{s, t\}$ and $g(u, v) \leq u(u, v)$ for all $u, v \in V$ then in $O(|E|)$ time we can easily compute a feasible s-t flow $f \in \mathbb{R}^E_{\geq 0}$ with $v(f) = v(g)$ simply by mapping each non-zero value of $g(u, v)$ to an assignment to edges over that pair. Consequently, we will be fairly informal about using $f \in \mathbb{R}^E$ v.s. $f \in \mathbb{R}^{V\times V}$ notation.

[1] These lecture notes are a work in progress and there may be typos, awkward language, omitted proof details, etc. Moreover, content from lectures might be missing. These notes are intended to converge to a polished superset of the material covered in class so if you would like anything clarified, please do not hesitate to post to Piazza and ask.

1.2 Recap

Note that our cut characterization of flows is a little cleaner with this notation, last class we showed that for all s-t cuts S, i.e. $S \subseteq V$ with $s \in S$ and $t \notin S$ we have that

$$\sum_{(u,v)\in\partial(S)} f(u,v) - \sum_{(u,v)\in\partial(V\setminus S)} f(u,v) = v(f) \leq \min_{C\subseteq V, s\in C, t\notin C} u(C) = \sum_{e\in\partial(C)} u_e \,.$$

2 Starting to Solve Maximum Flow Problem

As we suggested last class, a good starting point for this question is to look at the extremes. First, can the maximum flow value v_* ever be negative? The answer is clearly no since $f = \vec{0}$ is always a feasible s-t flow trivially.

Next, we can ask when is the maximum flow value positive? As we saw, every simple s-t path always induces a flow.

Lemma 1. *Given any simple s-t path $e_1, ..., e_k$ we can find a feasible s-t flow f of value $\alpha = \min_i u_{e_i}$ in $O(k)$ time.*

Proof. Let $f_e = \alpha$ if $e = e_i$ for some $i \in [k]$ and $f_e = 0$ otherwise. It is easy to check that the constraints are met. □

When is $v_* = 0$? We can show this is precisely when there is no s-t path on positive capacity edges or equivalent there is a s-t cut of capacity 0.

Lemma 2. *$v_* > 0$ if and only if there is a s-t path on positive capacity edges or equivalently there are no s-t cuts of capacity 0.*

Proof. We have done this sort of a proof many times in previous lectures so we only sketch it here. Suppose there is no s-t path on positive capacity edges, i.e. t is not reachable from s. Let $R \subseteq V$ be the set of vertices reachable from R on positive capacity edges. If there was $u \in R$ and $v \notin R$ with $u(u,v) > 0$ then v would be reachable from s. Consequently, $u(R) = 0$ and the result follows as we have shown $v(f) \leq u(R)$. On the other hand if there is a s-t path on positive capacity edges then we have shown there is a flow with positive value and clearly all s-t cuts have positive capacity as there is an edge on the path that crosses it. □

Consequently, we see that computing a maximum flow is a generalization of computing reachability. It is a measure of not just whether s can reach t, but how robustly s can reach t. Actually, as we will show later in undirected graphs it is exactly computing the number of edge disjoint paths from s to t.

So how do we use this knowledge to compute a maximum flow? We could try the greedy algorithm. Find a s-t path, send flow on it, delete those edges and repeat until nothing left, i.e. keep sending flow until nothing is left. Unfortunately there is an easy counterexample that shows this fails. Make two parallel edge independent s-t paths of length two, with vertices s,v_1,t and s, v_2, t respectively and add an edge from v_1 to v_2. The maximum flow value on this graph is two but if we send one unit on the $s \to v_1 \to v_2 \to t$ path then after deleting those edges t is not longer reachable from s. It is easy to show we can make this worse and worse (easily losing a factor of n from optimal even for unit capacity graphs). Consequently, we need to do something smarter

3 Residual Graphs

So how do we do better? The key problem is that greedy doesn't work. We can't simply make a decision and never undo it. We need a mechanism for after we route along a path, to not have lost our ability to still find an optimal flow. To do this, let's think about what we can add to a feasible s-t flow while maintaining a s-t flow. I think this is more natural to think about in the all pairs notion of flow.

Suppose we have feasible s-t flow f and we wish to know whether we can add some g to f to and maintain feasibility. Note that antisymmetry and being a s-t flow are homogeneous linear constraints and consequently if f is a s-t flow and g is a s-t flow then $f+g$ is a s-t flow. Furthermore, the value of a flow is a homogeneous function of the flow, so $v(f+g) = v(f) + v(g)$. This also means that given any s-t flow f and have a maximum flow f_* then $f - f_*$ is also a s-t flow. In short, if we optimize over s-t flows g such that $f+g$ meets the capacity constraints then this is the same as optimizing over all s-t flows.

So what happens to the capacity constraints? We need $f(u,v) \leq u(u,v)$ for all u, v so we want $f(u,v) + g(u,v) \leq u(u,v)$ for all $(u,v) \in E$. Or in other words $g(u,v) \leq u(u,v) - f(u,v)$. Note that if $f(u,v)$ is feasible then $u(u,v) - f(u,v) \geq 0$ and this makes sense. What this means is if we have an edge with α capacity from u to v and β capacity from v to u and a feasible flow f that sends γ from u to v then so long as we send at most $\alpha - \gamma$ from u to v or $\beta + \gamma$ from v to u then we are still feasible. Updating with these capacities is known as residual capacities and this is known as the residual graph.

Definition 3 (Residual Graph)**.** For capacitated graph $G = (V, E, u)$ and feasible s-t flow f the *residual graph* $G_f = (V, E_f, u_f)$ is the graph with capacities $u_f(u,v) = u(u,v) - f(u,v)$ for all pairs u and v and E_f is the the set of edges where $u_f > 0$.

We designed the residual graph so that it suffices to solve the problem on the residual graph. We write this formally as a lemma; the proof is essentially immediate from our discussion.

Lemma 4. *Given a feasible s-t flow f in capacitated graph $G = (V, E, u)$ we can compute residual graph G_f in $O(|E|)$ time and given any feasible s-t flow g in G_f we can compute a flow h in G in $O(|E|)$ time with $v(h) = v(f) + v(g)$. Furthermore, the maximum flow value in G_f denote v_f^* satisfies $v_* = v(f) + v_f^*$.*

This is a powerful type of self reduction. This lemma says given a maximum flow problem and a flow we can reduce the problem to a maximum flow problem on a graph with a smaller flow value. It suggests the following meta-algorithm

Algorithm 1 Augmenting Flow Meta Algorithm

Input: capacitated graph $G = (V, E, u)$ and vertices $s, t \in V$
Let $f_0 = \vec{0}$ and $i = 0$
while t is reachable from s in G_{f_i} **do**
  Compute s-t flow g_i in G_{f_i}
  Let f_{i+1} be s-t flow of value $v(f_i) + v(g_i)$ by combining f and g
  $i := i + 1$
end while
Return f

Many classic algorithms for the maximum flow problem fall in this framework. Where they differ is in how the g_i are computed. When the g_i is a path, it is known as an augmenting path. As we have seen this algorithm works more broadly when g_i is a flow and some of the improvements in the past came from using better nearly linear time algorithms for computing the g_i.

4 Ford Fulkerson

So how do we find a g_i? One of the most natural approaches is simply to take an arbitrary s-t path and send one unit of flow along this path. This is precisely the approach of Ford Fulkerson, a classic maximum flow algorithm. We analyze it as follows.

Theorem 5. *The Ford Fulkerson algorithm, when g_i is chosen to be an aribtrary s-t path and one unit is sent along it, and when the capacities are integers can be used to solve the maximum flow problem in $O(mv_*)$ time.*

Proof. Note since each g_i is integer the residual capacities are always integer and thus it is always feasible to send one unit of flow along a path in G_{f_i} when it is found. In every iteration then $v(g_i) = 1$ and consequently in each iteration $v(f_{i+1}) = v(f_i)+1$. Since the maximum flow value is v_* this algorithm works as desired. □

So what does this say about the value of minimum cuts? First, note that we can compute exactly what the capacities are in G_f based on the capacities in G and the value of f.

Lemma 6. *For all capacitated $G = (V, E, u)$, $S \subseteq V$, feasible s-t flows f, and residual graph $G_f = (V, E, u_f)$ we have*

$$u_f(S) = \begin{cases} u(S) - v(f) & \text{if } s \in S, t \notin S \\ u(S) + v(f) & \text{if } s \notin S, t \in S \\ u(S) & \text{otherwise} \end{cases}$$

Proof. Follows from direct calculation. □

From this we see that Ford Fulkerson essentially proves that the value of the maximum flow equals the value of the minimum cut.

Corollary 7. *The value of the maximum flow equals the value of the minimum cut.*

Proof. Note that when Ford Fulkerson terminates there is a s-t cut of capacity 0 in the residual graph which means there is a cut of capacity $v(f)$ in the original graph. This proves the result when the capacities are integer. Now note that if we multiply the capacity of all edges by α then the value of the maximum flow increases exactly by α. Consequently, if capacities are rational we can multiply them by a value to make them all integer and multiply by the inverse to get the resulting flow and again the result follows. When the capacities are irrational we can take an arbitrarily good rational approximation (smaller than the smallest difference between cut values) and the result follows. □

We can also use this to give a running time just in terms of the capacities of the edges.

Corollary 8. *If all edges have integer capacities between 1 and U then we can compute the maximum flow in $O(m^2U)$ time. If the graph is also simple then we can compute the maximum flow in $O(mnU)$ time.*

Proof. If all edges have capacity U then all cuts have size at most mU so the minimum cut has size $O(mU)$ and $v_* \leq mU$ and the result follows by Ford Fulkerson. Further, if the graph is simple, then the s-t cut $\{s\}$ has capacity at most $(n-1)U$ yielding $v_* = O(nU)$. □

5 Improving

So how do we get a better algorithm? Before that, we should ask, do we have a polynomial time algorithm for maximum flow? In some sense, we do not. Suppose are capacities are integers. We can write the capacities in binary which takes $O(\log U)$ bits, so if we want to be polynomial in the input size (represented in bits), then our running time needs to be $O(((m+n)\log U)^c)$ but the running time we have provided could be much worse. Such a running time if we got it would be known as a *weakly polynomial running time* we could also ask for a *strongly polynomial running time* where the number of operations we perform doesn't depend on U at all. We discuss this and application, e.g. computing edge disjoint paths and matchings, in the next lecture.