CS 31: Algorithms (Spring 2019): Lecture 15

Date: 14th May, 2019

Topic: Graph Algorithms 5: Flows and Cuts in a Graph

Disclaimer: These notes have not gone through scrutiny and in all probability contain errors. Please notify errors on Piazza/by email to deeparnab@dartmouth.edu.

We next move to flows in graphs, the cornerstone of a whole area called *combinatorial optimization* and *linear programming*.

Remark: *Before we begin, here is a notation we use throughout this lecture and the remainder. Suppose we have a real number $x : E \to \mathbb{R}$ associated with every edge of a graph. And suppose $B \subseteq E$ is a subset of edges. Then we use the shorthand $x(B)$ to denote the sum $\sum_{e \in B} x(e)$.*

1 Flows in a graph

Formally, a flow in a given graph $G = (V, E)$ is just an assignment of values to edges. What makes an assignment a *flow* is certain constraints that (a) capture some physical reality, and (b) which are extremely useful. The picture to keep in mind when thinking of flows is actually of "pipes" instead of edges, and the "assignment" is the rate at which some fluid (water?) is flowing through these pipes. The water could be coming from somewhere (sources), it could be going somewhere (sinks), there could be accumulation (excesses), and pipes mayn't be able to handle more than some rate (capacities). All these jargon is formalized below.

Definition 1 (Flow Network)**.** A flow network (G, s, t, u) consists of a directed graph $G = (V, E)$, two specified vertices s, t, and a capacity function $u : E \to \mathbb{Z}_{\geq 0}$ on edges of the graph. The vertex s is called the *source* vertex and the vertex t is called the *sink* vertex.

Definition 2 (Feasible Flow)**.** A feasible/valid/standard *flow* in a flow network is an assignment $f : E \to \mathbb{R}_{\geq 0}$ satisfying the following two constraints:

- (Capacity Constraints) For every edge $e \in E$, $0 \leq f(e) \leq u(e)$.
- (Conservation Constraints) For every vertex $v \neq \{s, t\}$, $\sum_{(u,v) \in E} f(u, v) = \sum_{(v,w) \in E} f(v, w)$.

The first constraint limits the amount of flow any edge by the capacity. The second is more interesting. To that end, let us make another definition.

Definition 3 (Excess)**.** Given any assignment $f : E \to \mathbb{R}_{\geq 0}$, we define the following excess_f vector for *every* vertex $v \in V$.

$$\mathsf{excess}_f(v) = \sum_{(u,v) \in E} f(u, v) - \sum_{(v,w) \in E} f(v, w)$$

That is, the $\text{excess}_f(v)$ is the total "in-flow" into v, that is, the total flow coming into v minus the total "out-flow" from v, that is, the total flow coming out of v. Taking the water metaphor again, $\text{excess}_f(v)$ is the rate at which surplus water accumulates at the vertex v if the flow is governed by the assignment f.

Going back to the definition of a feasible flow, we see that a flow is a feasible flow if (a) capacity constraint holds at every edge, and (b) the excess at any *non-source-non-sink* vertex is exactly 0.

Now that we have defined flows, two questions arise: (a) why?, and (b) do they always exist? We postpone the answer to (a), but give a quick answer to (b): Yes, just set $f(e) = 0$ for all edges. Intuitively, you must feel this is a bogus answer, and indeed in some sense it is. To make things more interesting, we need to have a notion of *value* of a flow.

Definition 4 (Value of a flow)**.** The *value* of a feasible flow is the total excess at the sink, which from the following lemma, is the negative of the total excess at the source. This will be denotes as $\text{val}(f)$.

Lemma 1 (Sum of Excesses)**.** For any flow $f : E \to \mathbb{R}_{\geq 0}$ (not necessarily a feasible flow), we have $\sum_{v \in V} \text{excess}_f(v) = 0$.

Proof. Write out the definition for every vertex and add. Follows since every edge (u, v) has a head and a tail. ✍

Exercise: *Complete this proof.*

□

Remark: *Although we have been talking about directed graphs, the above also makes sense for undirected graphs. However, remember that flows are* directed *constructs. That is, $f(u, v) = 1$ doesn't imply $f(v, u) = 1$. More precisely, the flow is defined on ordered pairs of E. In an undirected graph, we could have both $f(u, v)$ and $f(v, u)$ non-zero, but their sum must add up to less than that edge's capacity. For simplicity, let's assume we have a directed graph.*

MAX s, t FLOW
Input: A flow network (G, s, t, u).
Output: A feasible flow $f : E \to \mathbb{R}_{\geq 0}$ of *maximum* value.

1.1 A glimpse into Linear Programming

This subsection was not covered in class.

Before we move forward, let's put this MAX FLOW problem as a *special* case of a more general problem. For this, think of $f(e)$'s as "variables" and let $\mathbf{f}$ be the m-dimensional vector of these variables. The capacity constraints on the flow $\mathbf{f}$ can be written as

$$\mathbf{0} \leq I_m \cdot \mathbf{f} \leq \mathbf{u} \tag{1}$$

where I_m is the $m \times m$ *identity* matrix (diagonal 1s, everywhere else 0).

What about the conservation constraints? Consider the $n \times m$ matrix B where rows correspond to vertices and columns corresponds to edges. The column corresponding to edge $(u,v) \in G$ has a $+1$ in the vth row, a -1 in the uth row, and a 0 in every other row. Let b_v be the row corresponding to the vertex v, and note that $\langle b_v, \mathbf{f} \rangle$ is precisely $\sum_{(u,v)\in E} f(u,v) - \sum_{(v,w)\in E} f(v,w)$. Thus, the conservation constraints of the flow can be succinctly written as

$$B \cdot \mathbf{f} = \begin{cases} 0 & \text{if } v \neq \{s,t\} \\ -F & \text{if } v = s \\ +F & \text{if } v = t \end{cases} \tag{2}$$

where F is the value of the flow.

Therefore, the maximum flow problem is precisely the problem of maximizing the variable F subject to the constraints (1) and (2), where both these constraints are described by providing upper and lower bounds on some *linear* functions of the variables $\mathbf{f}$. This is a *special* case of a much more general problem called linear programming.

LINEAR PROGRAMMING
Input: $M \times N$ matrix A, an M dimensional constraint vector b, and an N-dimensional cost vector c.
Output: An N-dimensional vector x which is a solution to

$$\begin{aligned} \max \quad & \langle c, x \rangle \\ & A \cdot x \leq b \end{aligned}$$

Size: The number of bits required to describe all the data.

Remark: *One of* **the** *most remarkable facts ever discovered is that* LINEAR PROGRAMMING *can be solved in polynomial time. We will not see any algorithms to solve linear programming. But I will try, once in a while, to mention how one would solve linear programs in general using the ideas we use to solve maximum flow.*

1.2 Flow Decomposition into Paths and Cycles

This subsection was not covered in class.

Any feasible s,t flow $f : E \to \mathbb{R}_{\geq 0}$ in a flow network can be decomposed into paths and cycles such that the flow on any edge can be read out by looking at the flows on the paths and cycles containing this edge.

Theorem 1. Given any feasible s,t flow $f : E \to \mathbb{R}_{\geq 0}$, we can find a collection of s,t paths $\mathcal{P}$, and a collection of cycles $\mathcal{C}$ and an assignment $f : \mathcal{P} \cup \mathcal{C} \to \mathbb{R}_{\geq 0}$, such that $|\mathcal{P}| + |\mathcal{C}| \leq m$

and for every edge $e \in E$, we have

$$f(e) = \sum_{p \in \mathcal{P} : e \in p} f(p) + \sum_{C \in \mathcal{C} : e \in C} f(C)$$

Proof. For convenience, we add a dummy edge $(t, s) \in E$ and send $f(t, s) = \mathsf{val}(f)$ on it. Note that this new flow has $\mathsf{excess}(v) = 0$ for all v. We now show an algorithm to find the above decomposition.

Now pick an arbitrary edge $e = (u, v)$ in E with $f(e) > 0$. Since $\mathsf{excess}(v) = 0$, there must exist an edge (v, w) such that $f(v, w) > 0$. Similarly, there must exist and edge (w, x) with $f(w, x) > 0$. We keep following these edges till a vertex repeats itself. At that point we have found a cycle C. Let $\delta = \min_{e \in C} f(e)$. There are two cases – either $(t, s) \in C$ or not. In the former case, we delete (t, s) from C and add the remaining s, t-path p in $\mathcal{P}$ and set $f(p) = \delta$. In the latter case, we add C to $\mathcal{C}$ and set $f(C) = \delta$. We modify the flow f by setting $f(e) \leftarrow f(e) - \delta$ for all $e \in C$. And then we repeat.

Note that in every step at least one edge e gets 0 flow. We stop when all edges have 0 flow. Since each time $|\mathcal{P}| + |\mathcal{C}|$ goes up by one, we see that the final size is $\leq m$. □

2 Cuts in a graph

Now for something completely different.

Definition 5. Given a flow network (G, s, t, u), a subset $F \subseteq E$ of edges is an *s, t-cut* if there is no path from s to t in $G \setminus F$. The *capacity* of the cut $F \subseteq E$ is defined to be $\mathsf{cap}(F) := \sum_{e \in F} u(e)$.

An s, t-cut $F \subseteq E$ is *minimal* if any strict subset $F' \subseteq F$ is *not* an s, t-cut.

Every ***minimal*** s, t-cut can ***equivalently*** be represented as the *boundary* edges of a subset of vertices.

Definition 6. Given a directed graph $G = (V, E)$ and a a subset $S \subseteq V$ vertices, the *out-boundary* of S, denoted as $\partial^+ S$, is defined to be

$$\partial^+ S = \{(x, y) \in E : x \in S, y \notin S\}$$

Remark: *There is an analogous definition of*

$$\partial^- S = \{(x, y) \in E : x \notin S, y \in S\}$$

which is the t, s cut edges.

The following claim shows the equivalence.

Claim 1. The out-boundary of any subset $S \subseteq V$ such that $s \in S, t \notin S$ is an s,t-cut. Any *minimal* s,t-cut is the out-boundary of some subset $S \subseteq V$ with $s \in S, t \notin S$.

Proof. Fix a subset $S \subseteq V$ with $s \in S, t \notin S$. Let $F := \partial^+ S$. Consider the graph $H := G \setminus F$. Suppose, for contradiction, there is a path from s to t in H. Let this path be $s = x_0, x_1, \ldots, x_k = t$ where each (x_i, x_{i+1}) is an edge in H. Since $x_0 \in S$ and $x_k \notin S$, there must exist some i such that $x_i \in S$ and $x_{i+1} \notin S$. This implies (x_i, x_{i+1}) is an edge in $\partial^+ S$. But this implies $(x_i, x_{i+1}) \notin H$. Contradiction.

For the other direction, suppose $F \subseteq E$ is a minimal s,t-cut. Consider the vertices S that are reachable from s in $G \setminus F$. We claim that $F = \partial^+ S$. To see $\partial^+ S \subseteq F$, consider any edge $(u,v) \in \partial^+ S$. Since u is reachable from s in $G \setminus F$, if the edge $(u,v) \notin F$, that is, $(u,v) \in G \setminus F$, then v would be reachable from s in $G \setminus F$ as well. This contradicts that $v \notin S$. To see $F \subseteq \partial^+ S$, we use the first part to say that (a) $\partial^+ S$ is an s,t-cut, and then since it is a subset of F and F is *minimal*, we must have $F = \partial^+ S$. □

Remark: *Henceforth, when we talk about s,t-cuts, we may just think of them as out-boundaries of some subset S which contains s but doesn't contain t.*

MIN s,t CUT
Input: A flow network (G, s, t, u).
Output: An s,t-cut of *minimum* capacity.

The following lemma shows the (weak) relationship between flows and cuts. It states that the value of *any* s,t flow must be at most the value of *any* s,t cut.

Lemma 2. Let f be any feasible s,t flow and $\partial^+ S$ be any s,t cut. Then

$$\mathsf{val}(f) := \ \mathsf{excess}_f(t) \leq u(\partial^+ S) \ =: \mathsf{cap}(S)$$

Proof. Since we know that $\mathsf{excess}_f(t) = -\mathsf{excess}_f(s)$, the lemma is equivalently asking us to show that

$$u(\partial^+ S) + \mathsf{excess}_f(s) \geq 0$$

To get this, let's add the excesses for every $v \in S$. Why? At some level this is because we wish to argue about the relation between $\mathsf{excess}_f(s)$ and $u(\partial^+ S)$, and the edges participating in the latter may be "far away" from the vertex s. Rather, they involve vertices on the "boundary" of the set S and somehow we need to "propagate" their information to the vertex s which may be deep inside the set. The truer, less enlightening answer is that it works.

In any case, we get

$$\sum_{v \in S} \mathsf{excess}_f(v) = \sum_{v \in S} \left(\sum_{(u,v) \in E} f(u,v) - \sum_{(v,w) \in E} f(v,w) \right) = \sum_{(x,y) \in E} \left(f(x,y) \cdot \mathbf{1}_{y \in S} - f(x,y) \cdot \mathbf{1}_{x \in S} \right)$$

where $\mathbf{1}_{x \in S}$ is the indicator variable for $x \in S$ and takes value 1 if $x \in S$ and 0 if $x \notin S$.

Now note that since f is feasible, the LHS is precisely $\mathsf{excess}_f(s)$. On the other hand, the RHS is precisely $f(\partial^- S) - f(\partial^+ S)$, that is, the total flow on the $\partial^- S$ edges minus the total flow on the $\partial^+ S$ edges. Together, we get,

$$\mathsf{excess}_f(s) = f(\partial^- S) - f(\partial^+ S)$$

Now we use the observations

Observation 1. (a) Since $f(e) \geq 0$, we get $f(\partial^- S) \geq 0$, and (b) since $f(e) \leq u(e)$, we get $f(\partial^+ S) \leq u(\partial^+ S)$.

Taking this together, we get what we need to prove the lemma

$$\mathsf{excess}_f(s) \geq -u(\partial^+ S) \quad \Rightarrow \quad u(\partial^+ S) + \mathsf{excess}_f(s) \geq 0$$

□

The obvious corollary to the above fact is that the maximum feasible s, t flow in any graph is at most the minimum s, t cut.

Theorem 2 (Weak Duality)**.** In any graph network (G, s, t, u), the maximum s, t flow is at most the minimum s, t cut value.

Corollary 1 (Corollary to Lemma 2)**.** If we every find a feasible s, t flow f and an s, t cut S such that

1. $f(e) = u(e)$ for all $e \in \partial^+ S$
2. $f(e) = 0$ for all $e \in \partial^- S$

Then f is a maximum s, t flow and S is a minimum s, t cut.

Proof. The assumptions above imply that the inequalities in Observation 1 are indeed equalities. That would imply $\mathsf{excess}_f(s) + u(\partial^+ S) = 0$ implying $\mathsf{val}(f) = \mathsf{val}(S)$. □

3 The Residual Network

We now embark upon an algorithm for finding maximum flows. One of the key concepts is that of the *residual network*. Before we define this, let us first look at a natural algorithm that doesn't quite work.

Recall what we need to do: we need to find a valid flow $f : E \to \mathbb{R}_{\geq 0}$ such that $\mathsf{excess}_f(t)$ is maximized. We start with the zero flow: $f(e) = 0$ for all $e \in E$. As noted before, this is a valid flow. Now consider an s, t-path p in the graph G. Given such a path, we can *augment* the current flow f *along the path* p as follows:

- Let $\delta = \min_{e \in p} u(e)$

- For every $e \in p$, set $f(e) \leftarrow f(e) + \delta$.

Note that flow conservation remains valid; the total in-flow at any $v \neq s, t$ is equal to the total out-flow – it is either δ or 0. Also note that by choice of δ and since we started from the 0-flow, the capacity constraint also remains valid. Finally, the $\mathsf{excess}_f(t)$ increases by δ. Progress!

How should we proceed? We could repeat the steps above, namely, find an s, t-path p and then augment flow along path p. However, we have already sent some flow which could have used up some capacity of certain edges e. In the augmentation step we should be wary of this lest we violate the capacity constraint. The fix is to maintain a *residual capacity* $u_f(e)$ for every edge e. These are initially set to $u(e)$, the original capacity, but for every unit of flow that we pass through this edge, we must *decrease* its residual capacity. This leads to the following augmentation procedure along path p *given* we have sent flow f:

- Let $\delta = \min_{e \in p} u_f(e)$
- For every $e \in p$, set $f(e) \leftarrow f(e) + \delta$.
- For every $e \in p$, set $u_f(e) \leftarrow u_f(e) - \delta$.

The above process can be repeated over and over again, and every time the value of the flow increases by δ. We stop when $\delta = 0$, that is, we can't find any path p from s to t with $\min_{e \in p} u_f(e) > 0$. How would we check this? Simple; remove all edges with $u_f(e) = 0$ and check if there is a path from s to t. We write the full algorithm below.

```
1: procedure NaiveMaxFlow($G, s, t, u$):
2:     Start with $f \equiv 0$ and $u_f(e) = u(e)$ for all $e$.
3:     ▷ Invariant: $u_f(e) + f(e) = u(e)$ for all $e$.
4:     while true do:
5:         Find any path $p$ from $s$ to $t$ with $\min_{e \in p} u_f(e) =: \delta > 0$.
6:         If no such path break
7:         For every edge $e \in p$: $f(e) \leftarrow f(e) + \delta$; $u_f(e) \leftarrow u_f(e) - \delta$.
8:     return $f$
```

As can be guessed by the name and the color of the shading, the algorithm above, although a solid try, doesn't return the correct solution. Let's see an example where it fails (maybe you'd like to try one first?): see Figure 1

Note that flow of value 2 can be "decomposed" into two flows: one which send flow of one unit along path s, a, q, t and the other which sends one unit of flow along the path s, p, b, t. In a sense, the flow we chose to send, that is the one on the path (s, a, b, t) was a *mistake*, and as of now we have not kept any safeguards against mistakes. The correct algorithm for maximum flow does precisely that. And this, finally, brings us to the notion of *the residual network*.

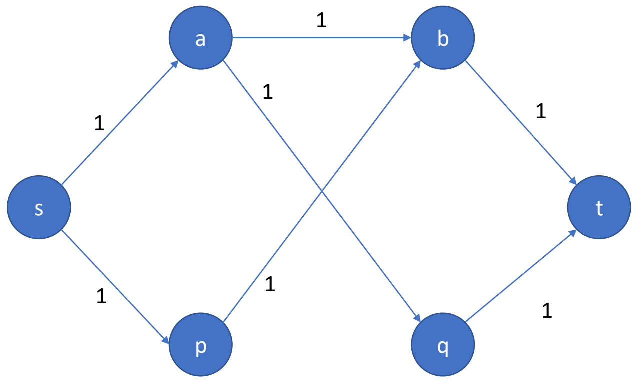

Figure 1: In this graph G, if we send our first augmentation along the path (s, a, b, t), then we would send 1 unit of flow on this. All these edges would have $u_f(e) = 0$ and deleting these edges disconnects s and t. Thus the NaiveMF algorithm would terminate. On the other hand, there is a flow of value 2 which sets $f(e) = 1$ for all edges except (a, b).

Definition 7. Given a flow network (G, s, t, u) and a valid flow $f : E \to \mathbb{R}_{\geq 0}$, the *residual network* with respect to flow f denoted as G_f is defined as follows:

- $G_f = (V, E_f)$ where $E_f = E \cup E_{\text{rev}}$
- $E_{\text{rev}} = \{(v, u) : f(u, v) > 0\}$, that is, E_{rev} contains the *reverse* of all edges which carry positive flow.
- The residual capacity on edges in E_f is defined as follows

$$u_f(x, y) = \begin{cases} u(x, y) - f(x, y) & \text{if } (x, y) \in E \\ f(y, x) & \text{if } (x, y) \in E_{\text{rev}} \end{cases}$$

Let us draw the reverse graph for the network in Figure 1 with respect to the flow of unit 1 sent along the path s, a, b, t. This is shown in Figure 2.

Why is the residual network important? Well, note that after the flow f is sent on the path (s, a, b, t), the residual network G_f does have a path from s to t where every edge has a residual capacity $u_f(e) > 0$; this path is $q = (s, p, b, a, q, t)$. As you can see, this path contains one edge (b, a) which is not in E but in E_{rev}.

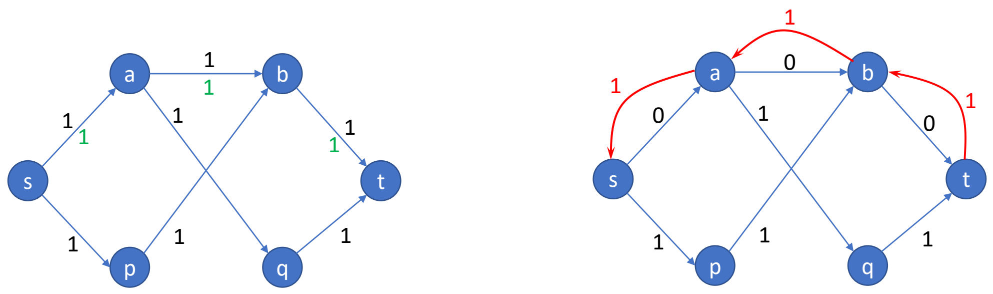

Figure 2: The graph in the left shows the flow in green. The graph in the right is the residual graph. The red edges are E_{rev}. The numbers are the residual capacities.

The question that should come into your mind now is: so what? The edge (b, a) doesn't even exist in the graph G; why are we bothering with such abstract constructs? Well, suppose you suppressed those thoughts and tried to *augment* flow along this path q. Wait! Firstlu, there is no edge (b, a) and now you are asking me to send flow across it?

But here's the point: we know that since $(b, a) \in E_{\text{rev}}$ there must exist $(a, b) \in E$ with $f(a, b) > 0$. Indeed, $f(a, b) = u_f(b, a)$. So *increasing* flow along the dummy reverse edge $(b, a) \in E_{\text{rev}}$ is actually just a short-hand for *decreasing* the flow along the edge (a, b). This augmentation is indicating that our first choice of sending flow across the edge (a, b) was perhaps a "mistake", and this is fixing it. Indeed, this is the conceptual abstraction of the residual network: send flow along edges, but keep the reverse edges as stop guards to rectify potential mistakes. Now we are ready to formally give the algorithm.