Lecturer: Anupam Gupta Scribe: Chun Kai Ling and Hoon Oh

In this lecture we will use multiplicative weight algorithms to solve approximate max flows problem.

Theorem 17.1. *For every $1/2 < \varepsilon \leq 1$, there exists an algorithm $Hedge_g(\varepsilon)$ such that for all times $T > \frac{\gamma\rho \ln N}{\varepsilon^2}$, for every sequence of gain vectors $(g^1, \ldots, g^T)$ with $g \in [-\gamma, \rho]^T$, and for every $i \in \{1, \ldots, n\}$, at every time $t \leq T$, $Hedge_g(\varepsilon)$ produces $p^t \in \Delta_N$ such that*

$$\frac{1}{T}\sum_{t=1}^{T}\langle g^t, p^t\rangle \geq \frac{1}{T}\sum_{t=1}^{T}\langle g^t, e_i\rangle - \varepsilon$$

where e_i is the ith vector in the standard basis of $\mathbb{R}^N$. Note that the first term on the right hand side represents the gain of the ith expert, and the last two terms represents the regret of not having always chosen the ith expert.

1 MW Application to Finding Max Flows

In the max flow problem, we are given a graph $G = (V, E)$ and vertices s and t. We will assume each edge has unit capacity and it is undirected. We want to find approximate max flow on the network, i.e. maximum flow from s to t by violating capacity constraint by $1 + \varepsilon$.

We will formulate the problem as a linear programming. Let $\mathcal{P}$ be the set of all s-t paths in G. Let f_P be the variable denoting the amount of flow going through path $P \in \mathcal{P}$.

$$\begin{aligned} \max \quad & \sum_{P\in\mathcal{P}} f_P \\ & \sum_{P:e\in P} f_P \leq 1 \qquad \forall e \in E \\ & f_P \geq 0 \qquad \forall P \in \mathcal{P} \end{aligned}$$

The first set of constraints says that for each edge e, the contribution of all possible flows is no greater than the capacity of that edge. The second set of constraints say that the contribution from each path must be non-negative.

Unfortunately, this is a gigantic linear program as there could be an exponential number of s-t paths. To overcome this, we employ two ideas: 1) take the weighted sum of constraints to obtain an 'average' constraint and 2) make the assumption that the optimal flow F over feasible s-t flows in G, is somehow known. This second idea yields the following linear *feasibility* problem.

$$K := \{f : f \geq 0, \sum_{P\in\mathcal{P}} f_p = F\}.$$

We will maintain weight vectors $p^t \in \Delta_m$. The 'average' constraint is obtained by the convex combination with weights given by p^t.

$$\sum_{e\in E} p_e f_e = \sum_{e\in E} p_e^t \sum_{P:e\in P} f_P \leq \sum_{e\in E} p_e^t = 1, \tag{17.1}$$

where f_e represents the net flow over edge e. By swapping order of summations), we obtain

$$\sum_{P \in \mathcal{P}} f_P \left(\sum_{e \in P} p_e^t \right) \leq 1.$$

Now, the inner summation is the path length of P with respect to wedge weights p_e^t. We denote this by $\mathsf{len}_t(P) := \sum_{e \in P} p_e^t$. Observe that the most aggressive way satisfy this constraint is to place all F units of flow into the shortest path according to $\mathsf{len}_t(P)$ subject to $f \in K$ and output INFEASIBLE if the shortest path gives a length of more than 1. This step can be done by a single call to Dijkstra's algorithm, in $O(m + n \log n)$ time.

1.1 The Algorithm

Plugging this into the Hedge-based algorithmic framework, we get the following algorithm:

1. $p^1 \leftarrow (1/m, \ldots, 1/m)$.
2. Call shortest-path oracle to give $f^t \in K$ such that $\sum_{P \in \mathcal{P}} f_P \sum_{e \in P} p_e^t \leq 1$. (If oracle says INFEASIBLE, it means that the original LP is infeasible, as any feasible solution to the original LP would satisfy a convex combination of the LP's constraints. That is, F was not the maximum flow in the first place.)
3. Define the gain vector by $g_e^t = f_e^t - 1$, where f^t is the flow found by the LP at round t.
4. Update p^{t+1} using Hedge(ε) using the gain vectors g_e^t.
5. Run steps 1 to 4 for $T := \Theta(\rho \ln m/\varepsilon^2)$ rounds, where $\rho := F$.
6. Return the average flow $\hat{f} \leftarrow \sum_{t=1}^{T} f^t/T$.

The analysis above ensures $\hat{f} \in K$, i.e. $\hat{f}$ has flow of value F from s to t. Also note that $\hat{f}$ satisfies flow conservation constraints, because each constituent flow f^t does. We will now show that the capacity constraint for each edge is approximately satisfied, i.e., $\hat{f}_e \leq 1 + \varepsilon$ for all edges.

Proof. By 17.1 we have $\frac{1}{T} \sum_{t=1}^{T} \langle p^t, g^t \rangle \geq \frac{1}{T} \sum_{t=1}^{T} \langle e_i, g^t \rangle - \varepsilon$ for all $i \in [m]$. The LHS may be upper bounded as follows,

$$\begin{aligned}
\langle p^t, g^t \rangle &= \sum_{e \in E} p_e^t \left(\sum_{P : e \in P} f_P - 1 \right) \\
&= \sum_{P \in \mathcal{P}} f_P \mathsf{len}_t(P) - 1 \\
&= F \mathsf{len}_t(P_{\min}) - 1 \leq 0 \qquad \text{(where } P_{\min} \text{ is the shortest path w.r.t. } \mathsf{len}_t\text{)}
\end{aligned}$$

Thus we find that $0 \geq$ L.H.S. Now we will look at the R.H.S. Note $\langle e_i, g^t \rangle$ correspond to violation on edge e_i at iteration t, where negative values imply that the capacity constraint for e_i is satisfied. If e_i was one of the edges in the shortest path at iteration t, then the violation is equal to $F - 1$. If e_i was not in the shortest path, then the violation is equal to -1. In either case, the violation is equal to $f_e^t - 1$. Thus we get R.H.S. $= \frac{1}{T} \sum_{t=1}^{T} (f_e^t - 1) - \varepsilon = \hat{f}_e - 1 - \varepsilon$.

Combining this with the upper bound on the LHS yields $\hat{f}_e \leq 1 + \varepsilon$ as desired. □

To get a flow that respects the constraints, we can scale down $\hat{f}$ by $(1 + \varepsilon)$ to get a feasible flow that has value $\frac{1}{1+\varepsilon} F \geq (1 - \varepsilon) F$. The running time of the algorithm is $O(\rho \log m(m + n \log n)/\varepsilon^2)$.

1.2 Remarks

- Observe that every gain vector g^t here is within the range $[-1, F]^m$. So we can use an asymmetric version of the Hedge guarantee. This analysis shows that if the gains are in $[-\gamma, \rho]^N$ for N experts, then we need only $O(\gamma\rho \ln N/\varepsilon^2)$ rounds. Using this we immediately get the runtime down to $O(F \log m/\varepsilon^2)$ iterations.

  Now each iteration takes $O(m + n \log n)$ to find the shortest path according to the lengths induced by p^t. So the total runtime is $\tilde{O}(mF)$. This is a lackluster result, since the Ford Fulkerson algorithm can solve max-flow problems *exactly* with the same amount of work. The next section will outline a a way to reduce the runtime via electrical flows.

- The algorithm repeats the following "natural" process: it finds a shortest path in the graph, pushes flow on it, and exponentially increases the length of the edge. This makes congested edges (those with a lot of flow) become very undesirable. Note that unlike usual network flows, these algorithms are greedy and cannot "undo" past actions (which is what pushing flow in residual flow networks does, when we use an arc backwards). So MW-based algorithms must ensure that very little flow goes on edges that are "wasteful".

- Since all edges in G has unit capacity and F is an optimal max flow, there exist F distinct paths from s to t. Further $\sum_{e \in E} p_e^t = 1$ implies $p^t \in \Delta_m$. Thus $\frac{1}{|\mathcal{P}|} \sum_{P \in \mathcal{P}} \mathsf{len}_t(P) \leq \frac{1}{F}$. This implies $\mathsf{len}_t(P_{\min}) \leq \frac{1}{F}$, wher $P_{\min}$ is the shortest path on G with respect to $\mathsf{len}_t(\cdot)$.

1.3 An Example

To see how the algorithm corrects the paths it chooses towards better paths, we demonstrate the first few steps of the algorithm on an example. For simplicity of calculations, we ignore the ε in updating the weights, i.e., assume that $w_e^{t+1} \leftarrow w_e^t(1 + g_e^t/\rho)$. The label on edge e in the t^{th} round indicates w_e^t. The green path in the tth round indicates the shortest path the oracle gives according to weights w_e^t.

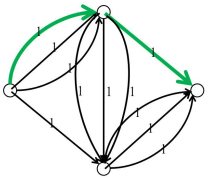

Figure 17.1: Round 1

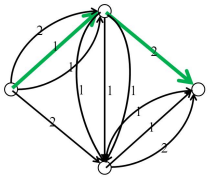

Figure 17.2: Round 2

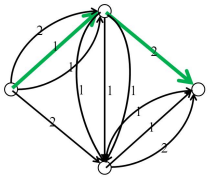

Figure 17.3: Round 3

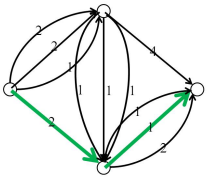

Figure 17.4: Round 4

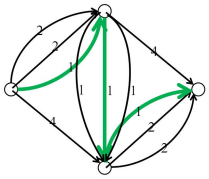

Figure 17.5: Round 5

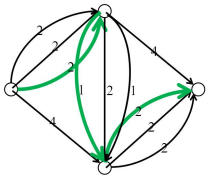

Figure 17.6: Round 6

2 Finding Max Flows using Electrical Networks

Recall that our naive algorithm pushed flow along shortest paths every iteration. This does not seem like an efficient method, since each path comprises only a fraction of edges. In fact, the problem is apparent by looking at Theorem 17.1. Observe that the rate of convergence depends strongly on the values γ, ρ. We define the quantity $\max\{\gamma, \rho\}$ as the *width* of the problem. Our goal is to keep the width as small as possible.

Recall that the i-th coordinate of the gain corresponds to the violation of capacity constraints. By restricting ourselves to pushing flows along paths, we are subject to a potentially large violation of $\Omega(m)$, which in turn implies a large width. In fact, we chose to use the shortest path as a solution to the feasibility problem mainly out of convenience. Can we do better? Our instinct might be to try to push flows along many paths simultaneously, since this would lead to flows which are more 'dispersed' and hence a smaller width. However, one must do so in a computationally efficient way – for example, one could solve the max-flow problem directly, which would defeat the purpose of the MW approach.

It turns out that modelling the graph as an electric circuit will lead us to the right mix of accuracy and speed. This section shows how one could use electrical flows algorithms to find approximate max-flows in (unit-capacity) undirected graphs in $\tilde{O}(m^{4/3}/\text{poly}(\varepsilon))$ time. The approach can be extended to all undirected graphs, and the runtime can be improved to $\tilde{O}(mn^{1/3}/\text{poly}(\varepsilon))$. At the time this result was announced (in 2011), it was the fastest algorithm for the problem.

2.1 Electrical Flows

Given an *undirected* graph, we can consider it to be an electrical circuit as shown in Figure 17.7: each edge of the original graph represents a resistor, and we connect (say, a 1-volt) battery between s to t. This causes electrical current to flow from s (the node with higher potential) to t.

How do we figure out what this electrical flow is going to be? We use the following laws we know to hold about electrical flows.

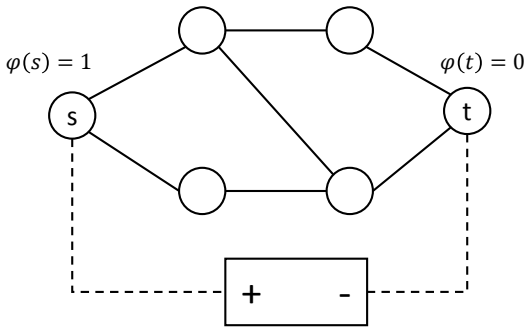

Figure 17.7: *The currents on the wires would produce an electric flow (where all the wires within the graph have resistance 1).*

Theorem 17.2 (Ohm's Law)**.** *If $e = (u, v)$ is an edge, the electrical flow f_{uv} on this edge is the ratio between the difference in potential ϕ (or voltage) and the resistance of the edge, where r_e is the resistance of edge e, i.e. $f_u v = \frac{\phi_u - \phi_v}{r_{uv}}$*

Theorem 17.3 (Kirchoff's Voltage Law)**.** *Along a cycle, the directed potential changes along the cycle sum to* 0*.*

Theorem 17.4 (Kirchoff's Current Law)**.** *The sum of the currents entering a node is the same as the sum of the currents leaving a node; i.e., there is flow-conservation at the nodes.*

These laws give us a set of linear constraints on the electrical flow values f_e, and solving this system of linear constraints tells us what the electrical flows f_e are. (For an example, see Wikipedia.)

2.1.1 The Laplacian

It turns out that we can write these linear constraints obtained above as follows, if we introduce a convenient matrix called the *graph Laplacian*. Given an undirected graph on n nodes and m edges with nonnegative *conductances* (weights) c_{uv} , we define the Laplacian matrix to be a $n \times n$ matrix, L.

$$L_{uv} \begin{cases} deg(u) & \text{if } u = v \\ -c_{uv} & \text{if } (u,v) \in E \\ 0 & \text{otherwise} \end{cases}$$

Another way to write this matrix is $L_{uv} = c_{uv} \cdot (e_u - e_v)^T (e_u - e_v)$. In a general graph G, we define the Laplacian to be:

$$L(G) = \sum_{(u,v)\in E} L_{uv}.$$

One may think of $L(G)$ as the sum of little 'edge-wise' Laplacians L_{uv}. Yet another definition for the Laplacian is $L = ACA^T$, where A is the vertex-edge matrix and C is a diagonal matrix of *conductances* (also known as inverse resistances $C_{ee} = 1/r_e$). If we take the 6-node graph in Figure 17.7 and assume that all edges have unit conductance, then the Laplacian L and vertex-edge matrix A are:

$$L = \begin{array}{c} \\ s \\ t \\ u \\ v \\ w \\ x \end{array} \begin{array}{c} \begin{array}{cccccc} s & t & u & v & w & x \end{array} \\ \begin{pmatrix} 2 & 0 & -1 & -1 & 0 & 0 \\ 0 & 2 & 0 & 0 & -1 & -1 \\ -1 & 0 & 3 & 0 & -1 & -1 \\ -1 & 0 & 0 & 2 & 0 & -1 \\ 0 & -1 & -1 & 0 & 2 & 0 \\ 0 & -1 & -1 & -1 & 0 & 3 \end{pmatrix} \end{array} = AIA^T$$

$$A = \begin{array}{c} \\ s \\ t \\ u \\ v \\ w \\ x \end{array} \begin{array}{c} \begin{array}{ccccccc} su & sv & uw & ux & vx & wt & xt \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & 1 \end{pmatrix} \end{array}$$

2.1.2 Solving Electrical Networks

Using the Laplacian allows us to solve for desired quantities in matrix representation conveniently. For example, suppose we want to find the unit resultant voltages upon applying a unit *current source* from vertices s to t. Recall $\phi = (\phi_v)_{v\in V}$ be a a vector of voltages. Then $(L\phi)_u = \sum_{v\in V} L_{uv}\phi_v = \sum_{v\in V} c_{uv}(\phi_u - \phi_v)$, which is equivalent to the net *outflow* of current from vertex u. From Kirchoff's

Current Law, the net current flow in all vertices $u \notin \{s,t\}$ is equal to 0, while flow out of s,t are $1,-1$ respectively. Hence, ϕ is obtained by solving the linear system

$$L\phi = (e_s - e_t),$$

where e_i represents the elementary vector with 1 at position i and 0's elsewhere. Once we have ϕ, we can use Ohm's law to get the current flowing on each edge, i.e., $CA^T\phi$. Hence, if we want to send F units of flow from s to t, we solve the system $L\phi = F(e_s - e_t)$, and let $f_{uv} = \frac{\phi_u - \phi_v}{r_{uv}}$. So now it remains to find the voltages for each of the vertices. How do we solve the linear system $L\phi = b$? We can use Gaussian elimination, of course, but there are faster methods: we'll discuss this in Section 2.1.4.

2.1.3 Electrical Flows Minimize Energy Burn

Here's another useful way of characterizing this flow. If we set voltages at s and t, this causes some amount I of flow to go between s to t. But how do these I units of flow split up? This flow happens to be the one that minimizes the total energy dissipated by the flow (subject to the flow value being I). Indeed, for a flow f, the energy burn on edge e is $f_{uv}^2 r_{uv} = \frac{(\phi_u - \phi_v)^2}{r_{uv}}$, and the total energy burn is

$$\mathcal{E}(f) = \sum_{e \in E} f_e^2 r_e = \sum_{(u,v) \in E} \frac{(\phi_u - \phi_v)^2}{r_{uv}} = \phi^T L \phi.$$

The electrical flow f produced happens to be

$$\arg\min_{f \text{ is an } s\text{-}t \text{ flow of value } I} \{\mathcal{E}(f)\}.$$

2.1.4 Solving Linear Systems

How do we solve the linear system $Lx = b$ fast? For the case when L is a Laplacian matrix, we can do things much faster than Gaussian elimination—essentially do it in time near-linear in the number of non-zeros of the matrix L.

Theorem 17.5 ([KMP10, KMP14]). *Suppose we are given a linear system $Lx = b$ for the case where L is a Laplacian matrix, with solution $\bar{x}$. Then we can find a vector $\hat{x}$ in time $O(m \log^2 n \log(1/\varepsilon)$ such that the error $z := L\hat{x} - b$ satisfies $z^\intercal L z \leq \varepsilon(\bar{x}^\intercal L \bar{x})$.*

Some history: Spielman and Teng [ST04, ST14] gave an algorithm to solve Laplacian linear systems that takes time about $O(m\, 2^{\sqrt{\log n \log\log n}})$. Koutis, Miller, and Peng [KMP10] improved this to $O(m \log^2 n)$; the current best time is $O(m\sqrt{\log n})$.

Moreover, Theorem 17.5 can be converted to what we need:

Theorem 17.6 ([CKM+11]). *There is an algorithm given a linear system $Lx = b$ (for L being a Laplacian matrix), outputs in $\tilde{O}(\frac{m \log R}{\delta})$ time a flow f that satisfies $\mathcal{E}(f) \leq (1+\delta)\mathcal{E}(\tilde{f})$, where $\tilde{f}$ is the min-energy flow, and R is the ratio between the largest and smallest resistances in the network.*

For the rest of this lecture we will assume that given a linear system $Lx = b$, we can compute the corresponding min-energy flow *exactly* in time $\tilde{O}(m)$. The argument can be extended to incorporate the errors, etc., fairly easily.

2.2 Obtaining an $\tilde{O}(m^{3/2})$ time Flow Algorithm

We will assume that the flow instance is feasible, i.e., that there is some flow f^* which is in K and satisfies all the edge constraints. Recall, our oracle takes as input $p \in \Delta_m = \{x \in [0,1]^m : \sum_e x_e = 1\}$. Here, we instead implement an oracle that enjoys a width of $O(\sqrt{m/\varepsilon})$, but satisfies a weaker version of inequality 17.1

$$\sum_{e \in E} p_e f_e \leq (1+\varepsilon) \sum_{e \in E} p_e + \varepsilon.$$

This idea is a simple modification to the hedge-based algorithm of Section 1.1. Instead of solving a shortest path problem to find a feasible solution to the linear system with averaged constraints, we will construct a *weighted* electrical network based off p_e, solve for electrical flows and push them through the original graph. All other steps remain the same. Specifically, the following step replaces the call to the shortest path oracle:

> Define resistances $r_e^t = p_e^t + \frac{\varepsilon}{m}$ for all edges. Compute currents f_e^t by solving the linear system $L^t \phi = F(e_s - e_t)$ and return the resultant flow f.

This idea of setting the resistance to be p_e plus a small error term is useful in controlling the width in non-electrical flows too.

Theorem 17.7. *If f^* is a flow with value F and f is the minimum energy flow (which is returned by the oracle), then*

1. $\sum_{e \in E} p_e f_e \leq (1+\varepsilon) \sum_{e \in E} p_e$,
2. $\max_e f_e \leq O(\sqrt{m/\varepsilon})$.

Proof. Recall that flow f^* satisfies all the constraints, and that f is the minimum energy flow. Then

$$\mathcal{E}(f^*) = \sum_e (f_e^*)^2 r_e \leq \sum_e r_e = \sum_e (p_e + \frac{\varepsilon}{m}) = 1 + \varepsilon.$$

Here we use that $\sum_e p_e = 1$. But since f is the flow K that minimizes the energy,

$$\mathcal{E}(f) \leq \mathcal{E}(f^*) \leq 1 + \varepsilon.$$

By Cauchy-Schwarz, we have that:

$$\sum_e r_e f_e = \sum_e (\sqrt{r_e} f_e \cdot \sqrt{r_e}) \leq \sqrt{(\sum_e r_e f_e^2)(\sum_e r_e)} \leq \sqrt{1+\varepsilon}\sqrt{1+\varepsilon} = 1 + \varepsilon$$

This proves the first part of the theorem. For the second part, we may use the bound on energy burnt to obtain

$$\sum_e f_e^2 \frac{\varepsilon}{m} \leq \sum_e f_e^2 \left(p_e + \frac{\varepsilon}{m}\right) = \sum_e f_e^2 r_e \leq 1 + \varepsilon$$

Each term in the leftmost summation is non-negative, hence for every edge e, we have the inequality

$$f_e^2 \frac{\varepsilon}{m} \leq 1 + \varepsilon,$$

which when rearranged yields for all edges e,

$$f_e \leq \sqrt{\frac{m(1+\varepsilon)}{\varepsilon}} \leq \sqrt{\frac{2m}{\varepsilon}}.$$

□

The first part of theorem 17.7 tells us that the average constraint is obeyed approximately, up to a factor of $1+\varepsilon$. The second part shows that individual edge capacities are obeyed somewhat, i.e., the width is bounded grows in $\sqrt{m}$, as opposed to the linearly when using shortest paths. This is precisely the kind of result we need – we now may efficiently find a flow which almost satisfies the average constraint, while simultaneously having a low width!

Using this oracle with the MW framework gives an algorithm which runs in time $\tilde{O}(m^{3/2}\varepsilon^{-5/2})$. Indeed, we have to run $\frac{\rho \log m}{\varepsilon^2}$ iterations, where the width ρ is $O(\sqrt{m/\varepsilon})$, and each iteration takes $\tilde{O}(m)$ time due to Theorem 17.6. And this runtime is tight; see, e.g., the example here.

Unfortunately, this runtime of $O(m^{3/2})$ is not that impressive: in the 1970s, Karzanov [Kar73], and Even and Tarjan [ET75] showed how to find maximum flows exactly in unit-capacity graphs in time $O(m \min(m^{1/2}, n^{2/3}))$. And similar runtimes were given for the capacitated problem by Goldberg and Rao in the late 1990s. Thankfully, we can take the electrical flows idea even further, as we show in the next section.

3 A Faster Algorithm

It seems that reducing the width to of order $\sqrt{m}$ was not quite enough. Our target is to find an oracle with width $\approx m^{1/3}$. Unfortunately, our above analysis is tight. Yet surprisingly we can do even better using a crude but effective trick – we find all edges have flow greater than $\rho \approx m^{1/3}$ and enforce them to be 0 by simply deleting them from the electrical network. Doing this forces the width to be of order $m^{1/3}$ which is great for convergence. However, the worry is that the graph has been butchered to the extent that the max flow has been significantly reduced. It turns out this is not the case, this section is dedicated to showing that by setting the threshold

$$\rho = \frac{m^{1/3} \log m}{\varepsilon},$$

we will be able to obtain an approximate solution. The two key ideas are:

1. We find electrical flows, but if any edge has more than ρ flow, then we kill that edge (set its resistance to ∞). We show that we don't kill too many edges—less than εF edges.
2. Each time we kill an edge, we will show that the effective resistance between s and t increases by a lot each time an edge is killed.

A couple observations and assumptions:

1. We assume that $F \geq \rho$
2. Instead of using the multiplicative weights process as a black box, we will explicitly maintain edge weights w_e^t. We use the notation $W^t := \sum_e w_e^t$.

Now we will need to define the *effective resistance* between nodes u and v. This is the resistance offered by the whole network to electrical flows between u and v. There are many ways of formalizing this, we'll use the one that is most useful in this context.

Definition 17.8 (Effective Resistance)**.** The effective resistance between s and t, denoted R_{eff}^{st} is the energy burn if we send 1 unit of electrical current from s to t. Since we only consider the effective resistance between s and t, we simply write R_{eff}.

Lemma 17.9 ([CKM+11]). *Consider an electrical network with edge resistances r_e.*

1. *(Rayleigh Monotonicity) If we change the resistances to $r'_e \geq r_e$ for all e then $R'_{eff} \geq R_{eff}$.*
2. *Suppose f is an s-t electrical flow, suppose e is an edge such that $f_e^2 r_e \geq \beta \mathcal{E}(f)$. If we set $r'_e = \infty$, then $R'_{eff} \geq (\frac{R_{eff}}{1-\beta})$.*

We'll skip the proof of this (simple) lemma. Let's give our algorithm. We start off with weights $w_e^0 = 1$ for all $e \in E$. At step t of the algorithm:

- Find the min-energy flow f^t of value F with respect to edge resistances $r_e^t = w_e^t + \frac{\varepsilon}{m} W^t$.
- If there is an edge e with $f_e^t > \rho$, delete e, recompute the flow f^t as in the above step.
- Else update the weights $w_e^{t+1} \leftarrow w_e^t (1 + \frac{\varepsilon}{\rho} f_e^t)$.

Stop after $T = \frac{\rho \log m}{\varepsilon^2}$ iterations, and output $\hat{f} = \frac{1}{T} \sum_t f^t$.

We want to argue like for Theorem 17.7, but note that the process deletes edges along the way, which we need to take care of.

Claim 17.10. *Suppose $\varepsilon \leq 1/10$. If we delete at most εF edges from the graph, the following hold:*

1. *the flow f^t at step t has energy $\mathcal{E}(f^t) \leq (1 + 3\varepsilon) W^t$.*
2. *$\sum_e w_e^t f_e^t \leq (1 + 3\varepsilon) W^t \leq 2W^t$.*
3. *If $\hat{f} \in K$ is the flow eventually returned, then $\hat{f}_e \leq (1 + O(\varepsilon))$.*

Proof. Remember there exists a flow f^* of value F that respects all capacities. Deleting εF edges means there exists a capacity-respecting flow of value at least $(1 - \varepsilon)F$. Scaling up by $\frac{1}{(1-\varepsilon)}$, there exists a flow f' of value F that uses each edge to extent $\frac{1}{(1-\varepsilon)}$. The energy of this flow according to resistances r_e^t is at most

$$\mathcal{E}(f') = \sum_e r_e^t (f'_e)^2 \leq \frac{1}{(1-\varepsilon)^2} \sum_e r_e^t \leq \frac{W^t}{(1-\varepsilon)^2} \leq (1 + 3\varepsilon) W^t,$$

for ε small enough. Since we find the minimum energy flow, $\mathcal{E}(f^t) \leq \mathcal{E}(f') \leq W^t(1 + 3\varepsilon)$. For the second part,

$$\sum_e w_e^t f_e^t \leq \sqrt{(\sum_e w_e^t)(\sum_e w_e^t (f_e^t)^2)} \leq \sqrt{W^t \cdot W^t (1 + 3\varepsilon)} \leq (1 + 3\varepsilon) W^t \leq 2W^t.$$

The last step is very loose, but it will suffice for our purposes.

To calculate the congestion of the final flow, observe that even though the algorithm above explicitly maintains weights, we can just appeal directly to the MW algorithm guarantee. The idea is simple: define $p_e^t = \frac{w_e^t}{W^t}$, and then the flow f^t satisfies

$$\sum_e p_e^t f_e^t \leq 1 + 3\varepsilon$$

for precisely the p^t values that MW would return if we gave it the flows $f^0, f^1, \ldots, f^{t-1}$. Using the MW guarantees, the average flow $\hat{f}$ uses any edge e to at most $(1 + 3\varepsilon) + \varepsilon$. □

Finally, all these calculations assumed we did not delete too many edges. Let us show that is indeed the case.

Claim 17.11. *We delete at most εF edges.*

Proof. The proof tracks two things, the total weight W^t and the s-t effective resistance R_{eff}. First the weight: we start at $W^0 = m$. When we do an update,

$$\begin{aligned} W^{t+1} &= \sum_e w_e^t \left(1 + \frac{\varepsilon}{\rho} f_e^t\right) = W_t + \frac{\varepsilon}{\rho} \sum_e w_e^t f_e^t \\ &\leq W^t + \frac{\varepsilon}{\rho}(2W^t) \qquad \text{(From Claim 17.10)} \end{aligned}$$

Hence we get that for $T = \frac{\rho \ln m}{\varepsilon^2}$,

$$W^T \leq W^0 \cdot \left(1 + \frac{2\varepsilon}{\rho}\right)^T \leq m \cdot \exp\left(\frac{2\varepsilon \cdot T}{\rho}\right) = m \cdot \exp\left(\frac{2\ln m}{\varepsilon}\right).$$

Now for the s-t effective resistance R_{eff}.

- Initially, since we send F flow, there is some edge with at least F/m flow on it, and hence with energy burn $(F/m)^2$. So R_{eff} at the start is at least $(F/m)^2 \geq 1/m^2$.
- Every time we do an update, the weights increase, and hence R_{eff} does not decrease. (This is why we argued about the weights w_e^t explicitly, and not just the probabilities p_e^t.)
- Every time we delete an edge e, it has flow at least ρ, and hence energy burn at least $(\rho^2) w_e^t \geq (\rho^2)\frac{\varepsilon}{m} W^t$. The total energy is at most $2W^t$ from Claim 17.10. This means it was burning at least $\beta = \frac{\rho^2 \varepsilon}{2m}$ fraction of the total energy. Hence

  $$R_{\text{eff}}^{new} \geq \frac{R_{\text{eff}}^{old}}{(1 - \frac{\rho^2\varepsilon}{2m})} \geq R_{\text{eff}}^{old} \cdot \exp\left(\frac{\rho^2\varepsilon}{2m}\right)$$

  if we use $\frac{1}{1-x} \geq e^{x/2}$ when $x \in [0, 1/4]$.
- For the final effective resistance, note that we send F flow with total energy burn $2W^T$; since the energy depends on the square of the flow, we have $R_{\text{eff}}^{final} \leq \frac{2W^T}{F^2} \leq 2W^T$.

Observe that all these calculations depended on us not deleting more than εF edges. So let's prove that this is indeed the case. If D edges are deleted in the T steps, then we get

$$R_{\text{eff}}^0 \exp\left(D \cdot \frac{\rho^2\varepsilon}{2m}\right) \leq R_{\text{eff}}^{final} \leq 2W^T \leq 2m \cdot \exp\left(\frac{2\ln m}{\varepsilon}\right).$$

Taking logs and simplifying, we get that

$$\begin{aligned} \frac{\varepsilon\rho^2 D}{2m} &\leq \ln(2m^3) + \frac{2\ln m}{\varepsilon} \\ \implies D &\leq \frac{2m}{\varepsilon\rho^2}\left(\frac{O(\ln m)(1+\varepsilon)}{\varepsilon}\right) \ll m^{1/3} \leq \varepsilon F. \end{aligned}$$

So, D is small enough as desired, and we don't remove too many edges. □

To end, let us note that the analysis is tight; see, e.g., the second example here.

Acknowledgments

These lecture notes were scribed by Chun Kai Ling and Hoon Oh, based on previous scribe notes of Jennifer Iglesias and Xue An Chuang.

References

[CKM+11] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *STOC '11*, pages 273–282, New York, NY, USA, 2011. ACM. 17.6, 17.9

[ET75] S. Even and R.E. Tarjan. Network flow and testing graph connectivity. *SIAM Journal of Computing*, 4:507–518, 1975. 2.2

[Kar73] A.V. Karzanov. Tochnaya otsenka algoritma nakhozhdeniya maksimal'nogo potoka, primenennogo k zadache "o predstavitelyakh". *Voprosy Kibernetiki. Trudy Seminara po Kombinatorno Matematike*, 1973. Title transl.: An exact estimate of an algorithm for finding a maximum flow, applied to the problem "on representatives", In: Issues of Cybernetics. Proc. of the Seminar on Combinatorial Mathematics. 2.2

[KMP10] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving sdd linear systems. In *FOCS '10*, pages 235–244, Washington, DC, USA, 2010. IEEE Computer Society. 17.5, 2.1.4

[KMP14] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD linear systems. *SIAM J. Comput.*, 43(1):337–354, 2014. 17.5

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC '04*, pages 81–90, 2004. 2.1.4

[ST14] Daniel A. Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM J. Matrix Analysis Applications*, 35(3):835–885, 2014. 2.1.4