In today's lecture we will use electrical flows algorithms to find approximate max-flows in (unit-capacity) undirected graphs in $\tilde{O}(m^{4/3}/\text{poly}(\varepsilon)$ time. As mentioned on the blog, this 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.

1 Solving Flow Problems using Multiplicative Weights

Remember that we defined K_{easy} to be:

$$K = \{\bar{f} | f_p \geq 0, \sum_p f_p = F\}$$

where we have a variable f_P for every s, t path P. Constraints are of the form

$$f_e := \sum_{p: e \in p} f_p \leq 1 \quad \forall e \in E$$

Henceforth we will use the shorthand $f_e := \sum_{p:e\in p} f_p$ (the flow on edge e). We have an oracle which given weights q_e for the edges, and we want to find a flow $f \in K$ such that

$$\sum_{e \in E} q_e f_e \leq \sum_{e \in E} q_e \tag{$\star$}$$

We define the width of the oracle as the smallest ρ such that

$$\max_e f_e \leq \rho. \tag{15.1}$$

We saw that using the multiplicative weights (MW) algorithm, we find a $(1+\varepsilon)$-approximate max flow $\hat{f}$—i.e., a flow of value F that has $\hat{f}_e \leq 1 + \varepsilon$—using $O(\frac{\rho \log m}{\varepsilon^2})$ calls to the oracle.

In Lecture #14, we saw that using shortest-path routing, you can get $\rho = F$. Since we can use Dijkstra's $O(m + n \log n)$ to implement the oracle, this gives an $\tilde{O}(\frac{mF}{\varepsilon^2})$ time algorithm.

Relaxed Oracle: For the rest of this section, we are going to relax the requirements for the oracle, so that we merely want the flow to satisfy the capacity constraints approximately:

$$\sum_{e \in E} q_e f_e \leq (1+\varepsilon) \sum_{e \in E} q_e + \varepsilon \tag{$\star\star$}$$

This will be useful in reducing the width from F down to $\tilde{O}(m^{1/2})$ and even lower.

2 Review of Electrical Networks

Given an *undirected* graph, we can consider it to be an electrical circuit as shown in Figure 15.1: 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.

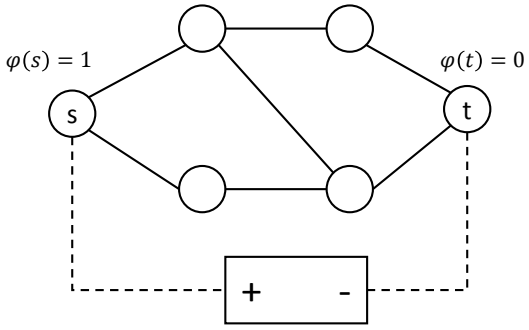

Figure 15.1: *The currents on the wires would produce an electric flow (where all the wires within the graph have resistance 1).*

2.1 Electrical Flows

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.

Theorem 15.1 (Ohm's Law)**.** *If $e = (u, v)$ is an edge, the electrical flow f_{uv} on this edge equals the difference in potential (or voltage) divided by the resistance of the edge, where r_e is the resistance of edge e.*

Theorem 15.2 (Kirchoff's Voltage Law)**.** *When you look at a cycle, the directed potential changes along the cycle sum to 0.*

Theorem 15.3 (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, define the Laplacian of a single edge uv will be a $n \times n$ matrix, which has a 1's at the (u, u) and (v, v) positions, -1's at the (u, v) and (v, u) positions, and zeroes elsewhere.

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

$$L(G) = \sum_{uv \in E} L_{uv} = diag(d(v_1), \ldots d(v_n)) - A$$

where A is the adjacency matrix. If we have resistances of edges r_e, then we get that our Laplacian:

$$L(G, r) = \sum_{uv} \frac{1}{r_{uv}} L_{uv}$$

If we take the 6-node graph in Figure 15.1, the resulting Laplacian L_{uv} is given below.

$$L_{uv} = \begin{array}{c} \\ s \\ t \\ u \\ v \\ w \\ x \end{array} \begin{array}{c} \begin{array}{cccccc} s & t & u & v & w & x \end{array} \\ \begin{pmatrix} 2 & 0 & -1 & -1 & 0 & 0 \\ 0 & 2 & 0 & 0 & -1 & -1 \\ -1 & 0 & 3 & 0 & -1 & -1 \\ -1 & 0 & 0 & 2 & 0 & -1 \\ 0 & -1 & -1 & 0 & 2 & 0 \\ 0 & -1 & -1 & -1 & 0 & 3 \end{pmatrix} \end{array}$$

We can distill Ohm's and Kirchoff's laws down to get: If we send one unit of current from s to t, the voltage vector $\phi = (\phi_v)_{v \in V}$ is obtained by solving the linear system

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

And once we have the vertex voltages ϕ_v, we can use Ohm's law to get the current flowing on each edge. So figuring out the electrical current on each edge, if we want to send F amperes from s to t, we solve the system $L\phi = F(\chi_s - \chi_t)$, and let $f_e = \frac{\phi_u - \phi_v}{r_{uv}}$. 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 2.1.3.

2.1.2 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 does this 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 $f_e^2 r_e$, and the total energy burn is

$$\mathcal{E}(f) = \sum_e f_e^2 r_e.$$

The electrical flow produced happens to be

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

Why? **Anupam to add in some text here.**

2.1.3 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 15.4 ([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 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 $O(m\sqrt{\log n})$.

This can be converted to what we need:

Theorem 15.5 ([CKM+11]). *There is an algorithm given a linear system $Lx = b$ (for L being a Laplacian matrix), outputs in $O(\frac{m \log R}{\delta})$ 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.

3 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 $q \in \Delta_m = \{x \in [0,1]^m : \sum_e x_e = 1\}$. We show how to implement an oracle that satisfies the weaker requirement (⋆⋆) and that has width $O(\sqrt{m/\varepsilon})$.

> Define resistances $r_e = q_e + \frac{\varepsilon}{m}$ for all edges, compute currents f_e by solving the linear system $L\phi = F(\chi_s - \chi_t)$. Return f.

This idea of setting the resistance to be q_e plus a small error term is useful in controlling the width in non-electrical flows too; see HW#4.

Theorem 15.6. *If $f \in K$ is the flow returned by the oracle, then*

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

Proof. Recall that flow $f^* \in K$ satisfies all the constraints, and let $f \in K$ be the minimum energy flow that we find. Then

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

Here we use that $\sum_e q_e = 1$. 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:

$$\sum_e q_e f_e \leq \sqrt{(\sum_e q_e f_e^2)(\sum_e q_e)} \leq \sqrt{1+\varepsilon} \leq 1 + \varepsilon$$

This proves the first part of the theorem. For the second part, look at the energy burned on e: this is $f_e^2 r_e \geq f_e^2 \frac{\varepsilon}{m}$. The total energy burned is more than the energy on that single edge, so we get that $f_e^2 \frac{\varepsilon}{m} \leq \mathcal{E}(f) \leq 1 + \varepsilon$, and hence

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

This proves the theorem. □

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 15.5. 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.

4 A Faster Algorithm

Our target is to find an oracle with width

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

The two main ideas are:

1. We find electrical flows, but if any edge has more than ρ flow, then we kill that edge. 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 some change occurs. 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 $F \geq \rho$, else use Ford Fulkerson to find max-flows exactly in $\tilde{O}(m^{4/3})$ time.
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. As you know 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 15.7 (Effective Resistance)**.** The effective resistance between s and t, denoted $R_{\text{eff}}^{s,t}$ 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 drop the superscript and merely write R_{eff}.

Lemma 15.8 ([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 $R'_{\text{eff}} \geq R_{\text{eff}}$.*
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 $R'_{\text{eff}} \geq (\frac{R_{\text{eff}}}{1-\beta})$.*

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 15.6, but note that the process deletes edges along the way, which we need to take care of.

Claim 15.9. *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 $q_e^t = \frac{w_e^t}{W^t}$, and then the flow f^t satisfies

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

for precisely the q^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 15.10. *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,

$$\begin{aligned} W^{t+1} &= \sum_e w_e^t \left(1 + \frac{\varepsilon}{\rho} f_e^t\right) = W_t + \frac{\varepsilon}{\rho}\sum_e w_e^t f_e^t \\ &\leq W^t + \frac{\varepsilon}{\rho}(2W^t) \qquad \text{(From Claim 15.9)} \end{aligned}$$

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 q_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 15.9. 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.

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. 15.5, 15.8

[ET75] S. Even and R.E. Tarjan. Network flow and testing graph connectivity. *SIAM Journal of Computing*, 4:507–518, 1975. 3

[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. 3

[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. 15.4, 2.1.3

[KMP14] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD linear systems. *SIAM J. Comput.*, 43(1):337–354, 2014. 15.4

[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. 2.1.3

[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. 2.1.3