16
Approximate Max-Flows using Experts

We now use low-regret multiplicative-weight algorithms to give approximate solutions to the s-t-maximum-flow problem. In the previous chapter, we already saw how to get approximate solutions to general linear programs. We now show how a closer look at those algorithms give us improvements in the running time (albeit in the setting of undirected graphs), which go beyond those known via usual "combinatorial" techniques. The first set of results we give will hold for directed graphs as well, but the improved results will only hold for undirected graphs.

16.1 The Maximum Flow Problem

In the *s-t maximum flow problem*, we are given a graph $G = (V, E)$, and distinguished vertices s and t. Each edge has a capacities $u_e \geq 0$; we will mostly focus on the unit-capacity case of $u_e = 1$ in this chapter. The graph may be directed or undirected; an undirected edge can be modeled by two oppositely directed edges having the same capacity.

Recall that *an s-t flow* is an assignment $f : E \to \mathbb{R}^+$ such that

(a) $f(e) \in [0, u_e]$, i.e., capacity-respecting on all edges, and

(b) $\sum_{e=(u,v)\in E} f(e) = \sum_{e=(v,w)\in E} f(e)$, i.e., flow-conservation at all non-$\{s, t\}$-nodes.

The *value* of flow f is $\sum_{e=(s,w)\in E} f(e) - \sum_{e=(u,s)\in E} f(e)$, the net amount of flow leaving the source node s. The goal is to find an s-t flow in the network, that satisfies the edge capacities, and has maximum value.

Algorithms by Edmonds and Karp, by Yefim Dinitz, and many others can solve the s-t max-flow problem exactly in polynomial time. For the special case of (directed) graphs with unit capacities, Shimon Even and Bob Tarjan, and independently, Alexander Karzanov showed in 1975 that the Ford-Fulkerson algorithm finds

the maximum flow in time $O(m \cdot \min(m^{1/2}, n^{2/3}))$. This runtime was eventually matched for general capacities (up to some poly-logarithmic factors) by an algorithm of Andrew Goldberg and Satish Rao in 1998. For the special case of $m = O(n)$, these results gave a runtime of $O(m^{1.5})$, but nothing better was known even for approximate max-flows, even for unit-capacity undirected graphs—until a breakthrough in 2010, which we will see at the end of this chapter.

16.1.1 A Linear Program for Maximum Flow

We formulate the max-flow problem as a linear program. There are many ways to do this, and we choose to write an enormous LP for it. Let $\mathcal{P}$ be the set of all s-t paths in G. Define a variable f_P denoting the amount of flow going on path $P \in \mathcal{P}$. We can now write:

$$\begin{aligned} \max \quad & \sum_{P \in \mathcal{P}} f_P \\ & \sum_{P : e \in P} f_P \le u_e \qquad \forall e \in E \\ & f_P \ge 0 \qquad \forall P \in \mathcal{P} \end{aligned} \tag{16.1}$$

The first set of constraints says that for each edge e, the contribution of all possible flows is no greater than the capacity u_e of that edge. The second set of constraints say that the contribution from each path must be non-negative. This is a gigantic linear program: there could be an exponential number of s-t paths. As we see, this will not be a hurdle.

16.2 A First Algorithm using the MW Framework

To using the framework from the previous section, we just need to implement the ORACLE: i.e., we solve a problem with a single "average" constraint, as in (15.3). Specifically, suppose we want a flow value of F, then the "easy" constraints are:

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

Moreover, the constraint $\langle a, f \rangle \le \beta$ is not an arbitrary constraint—it is one obtained by combining the original constraints. Specifically, given a vector $p^t \in \Delta_m$, the average constraint is obtained by the convex combination of these constraints:

$$\sum_{e \in E} p_e^t \Big(\sum_{P : e \in P} f_P \le u_e \Big), \tag{16.2}$$

where f_e represents the net flow over edge e. By swapping order of summations, and using the unit capacity assumption, we obtain

$$\sum_{P \in \mathcal{P}} f_P \Big(\sum_{e \in P} p_e^t \Big) \leq \sum_e p_e^t u_e = 1.$$

Now, the inner summation is the path length of P with respect to edge weights p_e^t, which we denote by $\text{len}_t(P) := \sum_{e \in P} p_e^t$. The constraint now becomes:

$$\sum_{P \in \mathcal{P}} f_P \, \text{len}_t(P) \leq 1, \tag{16.3}$$

and we want a point $f \in K$ satisfying it. The best way to satisfy it is to place all F units of flow on the shortest path P, and zero everywhere else; we output "infeasible" if the shortest-path has a length more than 1. This step can be done by a single call to Dijkstra's algorithm, which takes $O(m + n \log n)$ time.

We already argued in Theorem 15.4 that if there exists a feasible flow of value F in the graph, we never output "infeasible". Here is a direct proof.

If there is a flow of value F, there are F disjoint s-t paths. The vector $p^t \in \Delta_m$, so its values sum to 1. Hence, one of the F s-t paths P^* has $\sum_{e \in P} p_e^t \leq 1/F$. Setting $f_P = F$ for that path satisfies the constraint.

Now Theorem 15.4 says that running this algorithm for $\Theta\Big(\frac{\rho^2 \log m}{\varepsilon^2}\Big)$ iterations gives a solution $f \in K$, that violates the constraints by an additive ε. Hence, the scaled-down flow $f/(1+\varepsilon)$ would satisfy all the capacity constraints, and have flow value $F/(1+\varepsilon)$, which is what we wanted. To complete the runtime analysis, it remains to bound the value of ρ, the maximum amount by which any constraint gets violated by a solution from the oracle. Since we send all the F units of flow on a single edge, the maximum violation is $F - 1$. Hence the total runtime is at most

$$O(m + n \log n) \cdot \frac{F^2 \log m}{\varepsilon^2}.$$

Moreover, the maximum flow F is m, by the unit capacity assumption, which gives us an upper bound of $O(m^3 \, \text{poly}(\log m/\varepsilon))$.

16.2.1 *A Better Bound, via an Asymmetric Guarantee for Hedge*

Let us state (without proof, for now) a refined version of the Hedge algorithm for the case of asymmetric gains, where the gains lie in the range $[-\gamma, \rho]$.

Theorem 16.1 (Asymmetric Hedge). *Let $\varepsilon \in (0, 1/2)$, and $\gamma, \rho \geq 1$. Moreover, let $T \geq \frac{\Theta(\gamma\rho \ln N)}{\varepsilon^2}$. There exists an algorithm for the experts problem such that for every sequence $g^1, \ldots, g^T$ of gains with $g \in [-\gamma, \rho]^N$, produces probability vectors $\{p^t \in \Delta_N\}_{t \in [T]}$ online such that for each i:*

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

The proof is a careful (though not difficult) reworking of the standard proof for Hedge. (We will add it soon; a hand-written

proof is on the webpage.) Moreover, we can use this statement to prove that the approximate LP solver can stop after $\frac{\Theta(\gamma\rho \ln m)}{\varepsilon^2}$ calls to an oracle, as long as each of the oracle's answer x guarantee that $(Ax)_i - b_i \in [-\gamma, \rho]$.

Since a solution f found by our shortest-path oracle sends all F flow on a single path, and all capacities are 1, we have $\gamma = 1$ and $\rho = F - 1 \leq F$. The runtime now becomes

$$O(m + n \log n) \cdot \frac{1 \cdot (F-1) \log m}{\varepsilon^2}.$$

Again, using the naïve bound of $F \leq m$, we have a runtime of $O(m^2 \operatorname{poly}(\log m/\varepsilon))$ to find a $(1+\varepsilon)$-approximate max-flow, even in directed graphs.

16.2.2 *An Intuitive Explanation and an Example*

Observe that the algorithm repeats the following natural process:

1. it finds a shortest path in the graph,
2. it pushes F units of flow on it, and then
3. it increases the length of each edge on this path multiplicatively.

The factor happens to be $(1 + \varepsilon/F)$, because of how we rescale the gains, but that does not matter for this intuition.

This length-increase makes congested edges (those with a lot of flow) be much longer, and hence become very undesirable when searching for short paths. Note that the process is repeated some number of times, and then we average all the flows we find. So unlike usual network flow algorithms based on residual networks, these algorithms are truly greedy and cannot "undo" past actions (which is what pushing flow in residual flow networks does, when we use an arc backwards). This means these MW-based algorithms must ensure that very little flow goes on edges that are "wasteful".

To illustrate this point, consider an example commonly used to show that the greedy algorithm does not work for max-flow: Change the figure to make it more instructive.

16.3 *Finding Max-Flows using Electrical Flows*

The approach of the previous sections suggests a way to get faster algorithms for max-flow: *reduce the width of the oracle*. The approach of the above section was to push all F flow along a single path, which is why we have a width of $\Omega(F)$. Can we implement the oracle in a way that spreads the flow over several paths, and hence has smaller width? Of course, one such solution is to use the max-flow as the oracle response, but that would defeat the purpose of the MW approach. Indeed, we want a fast way of implementing the oracle.

We use the notation $\widetilde{O}(f(n))$ to hide factors that are poly-logarithmic in $f(n)$. E.g., $O(n \log^2 n)$ lies in $\widetilde{O}(n)$, and $O(\log n \log\log n)$ lies in $\widetilde{O}(\log n)$, etc.

For undirected graphs, one good solution turns out to be to use *electrical flows*: to model the graph as an electrical network, set a voltage difference between s and t, and compute how electrical current would flow between them. We now show how this approach gives us an $\widetilde{O}(m^{1.5}/\varepsilon^{O(1)})$-time algorithm quite easily; then with some more work, we improved this to get a runtime of $\widetilde{O}(m^{4/3}/\varepsilon^{O(1)})$. While we focus only on unit-capacity graphs, the algorithm can be extended to all undirected graphs with a further loss of poly-logarithmic factors in the maximum capacity, and moreover to get a runtime of $\widetilde{O}(mn^{1/3}/\operatorname{poly}(\varepsilon))$.

At the time this result was announced (by Christiano et al.), it was the fastest algorithm for the approximate maximum s-t-problem in undirected graphs. Since then, works by Jonah Sherman, and by Kelner et al. gave $O(m^{1+o(1)}/\varepsilon^{O(1)})$-time algorithms for the problem. The current best runtime is $O(m \operatorname{poly}\log m/\varepsilon^{O(1)})$-time, due to Richard Peng.

Christiano, Kelner, Madry, Spielman, and Teng (2010)

Sherman (2013)

Kelner, Lee, Orecchia, and Sidford (2013)

Peng (2014)

Interestingly, Shang-Hua Teng, Jonah Sherman, and Richard Peng are all CMU graduates.

16.3.1 Electrical Flows

Given a connected *undirected* graph with general edge-capacities, we can view it as an electrical circuit, where each edge e of the original graph represents a resistor with resistance $r_e = 1/u_e$, 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. Recall the following laws about electrical flows.

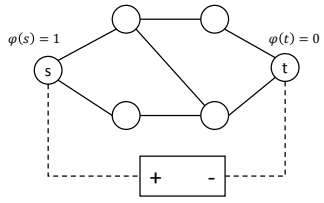

Figure 16.1: The currents on the wires would produce an electric flow (where all the wires within the graph have resistance 1).

Theorem 16.2 (Kirchoff's Voltage Law)**.** *The directed potential changes along any cycle sum to* 0*.*

This means we can assign each node v a potential ϕ_v. Now the actual amount of current on any edge is given by Ohm's law, and is related to the potential drop across the edge.

Theorem 16.3 (Ohm's Law)**.** *The electrical flow f_{uv} on the edge $e = uv$ is the ratio between the difference in potential ϕ (or voltage) between u, v and the resistance r_e of the edge:*

$$f_{uv} = \frac{\phi_u - \phi_v}{r_{uv}}.$$

Finally, we have flow conservation, much like in traditional network flows:

Theorem 16.4 (Kirchoff's Current Law)**.** *If we set s and t to some voltages, the electrical current ensures flow-conservation at all nodes except s, t: the total current entering any non-terminal node equals the current leaving it.*

These laws give us a set of linear constraints that allow us to go between the voltages and currents. In order to show this, we define the *Laplacian matrix* of a graph.

16.3.2 The Laplacian Matrix

Given an undirected graph on n nodes and m edges, with non-negative *conductances* c_{uv} for each edge $e = uv$, we define the Laplacian matrix to be a $n \times n$ matrix L_G, with entries

The conductance of an edge is the reciprocal of the resistance of the edge: $c_e = 1/r_e$.

$$(L_G)_{uv} = \begin{cases} \sum_{w:uw \in E} c_{uw} & \text{if } u = v \\ -c_{uv} & \text{if } (u,v) \in E \\ 0 & \text{otherwise} \end{cases}.$$

For example, if we take the 6-node graph in Figure 16.1 and assume that all edges have unit conductance, then its Laplacian L_G matrix is:

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

Note that L_G is not a full-rank matrix since, e.g., the columns sum to zero. However, if the graph G is corrected, then the vector $\mathbb{1}$ is the only vector in the kernel of L_G, so its rank is $n - 1$. (proof?)

Equivalently, we can define the Laplacian matrix L^{uv} for the graph consisting of a single edge uv as

$$L^{uv} := c_{uv}\,(e_u - e_v)^\intercal (e_u - e_v).$$

This Laplacian for the single edge uv has 1s on the diagonal at locations $(u,u), (v,v)$, and -1s at locations $(u,v), (v,u)$. Draw figure.

Now for a general graph G, we define the Laplacian to be:

$$L_G = \sum_{uv \in E} L^{uv}.$$

In other words, L_G is the sum of little 'per-edge' Laplacians L^{uv}. (Since each of those Laplacians is clearly positive semidefinite (PSD), it follows that L_G is PSD too.)

A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is called PSD if $x^\intercal A x \geq 0$ for all $x \in \mathbb{R}^n$, or equivalently, if all its eigenvalues are non-negative.

For yet another definition for the Laplacian, first consider the edge-vertex incidence matrix $B \in \{-1, 0, 1\}^{m \times n}$, where the rows are indexed by edges and the columns by vertices. The row corresponding to edge $e = uv$ has zeros in all columns other than u, v, it has an entry $+1$ in one of those columns (say u) and an entry -1 in the

other (say v).

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

The Laplacian matrix is now defined as $L_G := B^\intercal C B$, where $C \in R^{m \times m}$ is a diagonal matrix with entry C_{uv} containing the conductance for edge uv. E.g., for the example above, here's the edge-vertex incidence matrix, and since all conductances are 1, we have $L_G = BB^\intercal$.

16.3.3 Solving for Electrical Flows: $Lx = b$

Given the Laplacian matrix for the electrical network, we can figure out how the current flows by solving a linear system, i.e., a system of linear equations. Indeed, by Theorem 16.4, all the current flows from s to t. Suppose k units of current flows from s to t. By Theorem 16.3, the net current flow into a node v is precisely

$$\sum_{u:uv \in E} f_{uv} = \sum_{u:uv \in E} \frac{\phi_u - \phi_v}{r_{uv}}.$$

A little algebra shows this to be the v^{th} entry of the vector $L\phi$. Finally, by 16.4, this net current into v must be zero, unless v is either s or t, in which case it is either $-k$ or k respectively. Summarizing, if ϕ are the voltages at the nodes, they satisfy the linear system:

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

(Recall that k is the amount of current flowing from s to t, and e_s, e_t are elementary basis vectors.) It turns out the solutions ϕ to this linear system are unique up to translation, as long as the graph is connected: if ϕ is a solution, then $\{\phi + a \mid a \in \mathbb{R}\}$ is the set of all solutions.

Great: we have $n + 1$ unknowns so far: the potentials at all the nodes, and the current value k. The above discussion gives us potentials at all the nodes in terms of the current value k. Now we can set unit potential at s, and ground t (i.e., set its potential to zero), and solve the linear system (with $n - 1$ linearly independent constraints) for the remaining $n - 1$ variables. The resulting value of k gives us the s-t current flow. Moreover, the potential settings at all the other nodes can now be read off from the ϕ vector. Then we can use Ohm's law to also read off the current on each edge, if we want.

How do we solve the linear system $L\phi = b$ (subject to these boundary conditions)? We can use Gaussian elimination, of course, but the best implementations can take n^ω time in the worst-case. Thankfully, there are faster (approximate) methods, which we discuss in §16.3.5.

16.3.4 Electrical Flows Minimize Energy Burn

Here's another useful way of characterizing this current flow of k units from s and t: *the current flow is one minimizing the total energy dissipated*. Indeed, for a flow f, the *energy burn* on edge e is given by $(f_{uv})^2 r_{uv} = \frac{(\phi_u - \phi_v)^2}{r_{uv}}$, and the total energy burn is

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

The electrical flow f produced happens to be

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

We often use this characterization when arguing about electrical flows.

16.3.5 Solving Linear Systems

We can solve a linear system $Lx = b$ fast? If L is a Laplacian matrix and we are fine with approximate solutions, we can do things much faster than Gaussian elimination. A line of work starting with Dan Spielman and Shang-Hua Teng, and then refined by Ioannis Koutis, Gary Miller, and Richard Peng shows how to (approximately) solve a Laplacian linear system in the time essentially near-linear in the number of non-zeros of the matrix L.

Spielman and Teng (200?)

Koutis, Miller, and Peng (2010)

Theorem 16.5 (Laplacian Solver)**.** *There exists an algorithm that given a linear system $Lx = b$ with L being a Laplacian matrix (and having solution $\bar{x}$), find a vector $\hat{x}$ such that the error vector $z := L\hat{x} - b$ satisfies*

$$z^\intercal L z \leq \varepsilon(\bar{x}^\intercal L \bar{x}).$$

The algorithm is randomized? and runs in time $O(m \log^2 n \log 1/\varepsilon)$.

Given a positive semidefinite matrix A, the A-norm is defined as $\|x\|_A := \sqrt{x^\intercal A x}$. Hence the guarantee here says

$$\|L\hat{x} - b\|_L \leq \varepsilon \|\bar{x}\|_L .$$

Moreover, Theorem 16.5 can be converted to what we need; details appear in the Christiano et al. paper.

Corollary 16.6 (Laplacian Solver II)**.** *There is an algorithm given a linear system $Lx = b$ corresponding to an electrical system as above, outputs an electrical flow f that satisfies*

$$\mathcal{E}(f) \leq (1 + \delta)\mathcal{E}(\widetilde{f}),$$

where $\widetilde{f}$ is the min-energy flow. The algorithm runs in $\widetilde{O}(\frac{m \log R}{\delta})$ time, where R is the ratio between the largest and smallest resistances in the network.

For the rest of this lecture we assume we can compute the corresponding minimum-energy flow *exactly* in time $\widetilde{O}(m)$. The arguments can easily be extended to incorporate the errors.

16.4 An $\widetilde{O}(m^{3/2})$-time Algorithm

Recall the setup from §16.2: given the polytope

$$K = \{f \mid \sum_{P \in \mathcal{P}} f_P = F, f \geq 0\},$$

and some edge weights p_e, we wanted a vector in K that satisfies

$$\sum_e p_e f_e \leq 1. \tag{16.4}$$

where $f_e := \sum_{P: e \in P} f_P$. Previously, we set $f_{P^*} = F$ for P^* being the shortest s-t path according to edge weights p_e, but that resulted in the *width*—the maximum capacity violation—being too as large as $\Omega(F)$. So we want to spread the flow over more paths.

Our solution will now be to have the oracle return a flow with width $O(\sqrt{m/\varepsilon})$, and which satisfies the following weaker version of the length bound (16.4) above:

$$\sum_{e \in E} p_e f_e \leq (1+\varepsilon) \sum_{e \in E} p_e + \varepsilon = 1 + 2\varepsilon.$$

It is a simple exercise to check that this weaker oracle changes the analysis of Theorem 15.4 only slightly, still showing that the multiplicative-weights-based process finds an s-t-flow of value F, but now the edge-capacities are violated by $1 + O(\varepsilon)$ instead of just $1 + \varepsilon$.

Indeed, we replace the shortest-path implementation of the oracle by the following electrical-flow implementation: we construct a *weighted* electrical network, where the resistance for each edge e is defined to be

$$r_e^t := p_e^t + \frac{\varepsilon}{m}.$$

This idea of setting the edge length to be p_e *plus a small constant term* is a general technique useful in controlling the width in other settings, as we will see in a HW problem.

We now compute currents f_e^t by solving the linear system $L^t \phi = F(e_s - e_t)$ and return the resulting flow. It remains to show that this flow spreads its mass around, and yet achieves a small "length" on average.

Theorem 16.7. *If f^* is a flow with value F and f is the minimum-energy flow returned by the oracle, then*

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

2. *(width)* $\max_e f_e \leq O(\sqrt{m/\varepsilon})$.

Proof. Since the flow f^* satisfies all the constraints, it burns energy

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

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

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

Now, using Cauchy-Schwarz,

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

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

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

Since each term in the leftmost summation is non-negative,

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

for each edge e. □

Using this oracle within the MW framework means the width is $\rho = O(\sqrt{m})$, and each of the $O(\frac{\rho \log m}{\varepsilon^2})$ iterations takes $\widetilde{O}(m)$ time by Corollary 16.6, giving a runtime of $\widetilde{O}(m^{3/2})$.

In fact, this bound on the width is tight: consider the example network on the right. The effective resistance of the entire collection of black edges is 1, which matches the effective resistance of the red edge, so half the current goes on the top red edge. If we set $F = k + 1$ (which is the max-flow), this means a current of $\Theta(\sqrt{m})$ goes on the top edge.

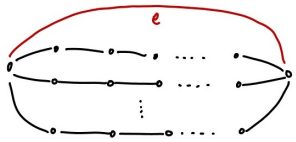

Figure 16.2: There are $k = \Theta(\sqrt{m})$ black paths of length k each. All edges have unit capacities.

Sadly, while the idea of using electrical flows is very cool, the runtime of $O(m^{3/2})$ is not that impressive. The algorithms of Karzanov, and of Even and Tarjan, for exact flow on *directed* unit-capacity graphs in time $O(m \min(m^{1/2}, n^{2/3}))$ were known even back in the 1970s. (Algorithms with similar runtime are known for capacitated cases too.) Thankfully, this is not the end of the story: we can take the idea of electrical flows further to get a better algorithm, as we show in the next section.

16.5 Optional: An $\widetilde{O}(m^{4/3})$-time Algorithm

The idea to get an improved bound on the width is to use a crude but effective trick: if we have an edge with electrical flow of more than

$\rho \approx m^{1/3}$ in some iteration, we delete it for that iteration (and for the rest of the process), and find a new flow. Clearly, no edge now carries a flow more than ρ. The main thrust of the proof is to show that we do not end up butchering the graph, and that the maximum flow value reduces by only a small amount due to these edge deletions. Formally, we set:

$$\rho = \frac{m^{1/3} \log m}{\varepsilon}. \tag{16.5}$$

and show that at most εF edges are ever deleted by the process. The crucial ingredient in this proof is this observation: every time we delete an edge, the effective resistance between s and t increases by a lot.

We assume that a flow value of F is feasible; moreover, $F \geq \rho$, else Ford-Fulkerson can be implemented in time $O(mF) \leq \widetilde{O}(m^{4/3})$.

Since we need to argue about how many edges are deleted in the entire algorithm (and not just in one call to the oracle), we explicitly maintain edge-weights w_e^t, instead of using the results from the previous sections as a black-box.

16.5.1 The Effective Resistance

Loosely speaking, the *effective resistance* between nodes u and v is the resistance offered by the network to electrical flows between u and v. There are many ways of formalizing this: the most useful one in this context is the following.

Definition 16.8 (Effective Resistance)**.** The effective resistance between s and t, denoted by $R_{\text{eff}}(st)$, is the energy burned if we send one unit of electrical current from s to t.

Since we only consider the effective resistance between s and t in this lecture, we simply write R_{eff}. The following results relate the effective resistances before and after we change the resistances of some edges.

Lemma 16.9. *Consider an electrical network with edge resistances r_e.*

1. *(Rayleigh Monotonicity) If we increase the resistances to $r'_e \geq r_e$ for all e, the resulting effective resistance is*

$$R'_{\text{eff}} \geq R_{\text{eff}}.$$

2. *Suppose f is an s-t electrical flow, suppose e is an edge with energy burn $f_e^2 r_e \geq \beta \mathcal{E}(f)$. If we set $r'_e \leftarrow \infty$, then the new effective resistance*

$$R'_{\text{eff}} \geq \left(\frac{R_{\text{eff}}}{1-\beta}\right).$$

Proof. Recall that if we send electrical flow from s to t, the resulting flow f minimizes the total energy burned $\mathcal{E}(f) = \sum_e f_e^2 r_e$. To prove the first statement: for each flow, the energy burned with the new resistances is at least that with the old resistances. Need to add in second part. □

16.5.2 *A Modified Algorithm*

Let's give our algorithm that explicitly maintains the edge weights: We start off with weights $w_e^1 = 1$ for all $e \in E$. At step t of the algorithm:

1. Find the min-energy flow f^t of value F in the remaining graph with respect to edge resistances $r_e^t := w_e^t + \frac{\varepsilon}{m} W^t$.
2. If there is an edge e with $f_e^t > \rho$, delete e (for the rest of the algorithm), and go back to Item 1.
3. Update the edge weights $w_e^{t+1} \leftarrow w_e^t(1 + \frac{\varepsilon}{\rho} f_e^t)$. This division by ρ accounts for the edge-capacity violations being as large as ρ.

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

16.5.3 *The Analysis*

Let us first comment on the runtime: each time we find an electrical flow, we either delete an edge, or we push flow and increment t. The latter happens for T steps by construction; the next lemma shows that we only delete edges in a few iterations.

Lemma 16.10. *We delete at most $m^{1/3} \leq \varepsilon F$ edges over the run of the algorithm.*

We defer the proof to later, and observe that the total number of electrical flows computed is therefore $O(T)$. Each such computation takes $\widetilde{O}(m/\varepsilon)$ by Corollary 16.6, so the overall runtime of our algorithm is $O(m^{4/3}/\operatorname{poly}(\varepsilon))$.

Next, we show that the flow $\widehat{f}$ is an $(1 + O(\varepsilon)$-approximate maximum s-t flow. We start with an analog of Theorem 16.7 that accounts for edge deletions.

Lemma 16.11. *Suppose $\varepsilon \leq 1/10$. If we delete at most εF edges from G:*

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

Proof. We assumed there exists a flow f^* of value F that respects all capacities. Deleting εF edges can only hit εF of these flow paths, so 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 using 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, we again use the Cauchy-Schwarz inequality:

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

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 guarantees . Indeed, define $p_e^t := \frac{w_e^t}{W^t}$ for each time t; the previous part implies that the flow f^t satisfies

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

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

Finally, it remains to prove Lemma 16.10.

Proof of Lemma 16.10. We track two quantities: the total weight W^t and the s-t-effective resistance R_{eff}. First, the weight starts at $W^0 = m$, and when we do an update,

$$\begin{aligned} W^{t+1} &= \sum_e w_e^t \left(1 + \frac{\varepsilon}{\rho} f_e^t\right) = W_t + \frac{\varepsilon}{\rho} \sum_e w_e^t f_e^t \\ &\leq W^t + \frac{\varepsilon}{\rho}(2W^t) \qquad \text{(From Claim 16.11)} \end{aligned}$$

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

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

Therefore, the total weight is at most $m^{1+2/\varepsilon}$. Next, we consider the s-t-effective resistance R_{eff}.

1. At the beginning, all edges have resistance $1+\varepsilon$. When we send F flow, some edge has at least F/m flow on it, so the energy burn is at least $(F/m)^2$. This means R_{eff} at the beginning is at least $(F/m)^2 \geq 1/m^2$.

2. The weights increase each time we do an update, so R_{eff} does not decrease. (This is one place it is more convenience to argue about weights w_e^t explicitly, and not just the probabilities p_e^t.)

3. Each deleted edge e has flow at least ρ, and hence energy burn at least $(\rho^2)\, w_e^t \geq (\rho^2)\, \frac{\varepsilon}{m}\, W^t$. Since the total energy burn is at most $2W^t$ from Lemma 16.11, the deleted edge e 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 $x \in [0, 1/4]$.

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 $R_{\text{eff}}^{final} \leq \frac{2W^T}{F^2} \leq 2W^T$.

(All these calculations hold as long as we have not deleted more than εF edges.) Now, to show that this invariant is maintained, suppose D edges are deleted over the course of the T steps. Then

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

Taking logs and simplifying, we get that

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

This bounds the number of deleted edges D as desired. □

16.5.4 *Tightness of the Analysis*

This analysis of the algorithm is tight. Indeed, the algorithm needs $\Omega(m^{1/3})$ iterations, and deletes $\Omega(m^{1/3})$ edges for the example on the right. In this example, $m = \Theta(n)$. Each black gadget has a unit effective resistance, and if we do the calculations, the effective resistance between s and t tends to the golden ratio. If we set $F = n^{1/3}$ (which is almost the max-flow), a constant fraction of the current (about $\Theta(n^{1/3})$) uses the edge e_1. Once that edge is deleted, the next red edge e_2 carries a lot of current, etc., until all red edges get deleted.

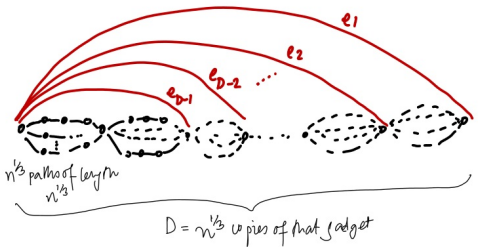

Figure 16.3: Again, all edges have unit capacities.

16.5.5 Subsequent Work

Sherman (2013)

Kelner, Lee, Orecchia, and Sidford (2013)

Peng (2014)

A couple years after this work, Sherman, and independently, Kelner et al. gave $O(m^{1+o(1)}/\varepsilon^{O(1)})$-time algorithms for approximate max-flow problem on undirected graphs. This was improved, using some more ideas, to a runtime of $O(m \operatorname{poly} \log m/\varepsilon^{O(1)})$-time by Richard Peng. These are based on the ideas of *oblivious routings*, and *non-Euclidean gradient descent*, and we hope to cover this in an upcoming lecture.

There has also been work on faster directed flows: work by Madry, and thereafter by more refs here, have improved the current best result for max-flow in unweighted directed graphs to $\widetilde{O}(m^{4/3})$, matching the above result.