CS 466/666: Algorithm Design and Analysis **University of Waterloo: Fall 2025**

Lecture 19

November 13, 2025

Instructor: Sepehr Assadi

Topics of this Lecture

We will have our last application of the MWU technique in this lecture.

1 The Maximum Flow Problem

In this lecture, we see another application of the MWU technique, this time to the *maximum flow* problem. Throughout this lecture, we limit our study to one of the simplest forms of this problem by restricting the input graph to be undirected and unit-capacity. Recall that this problem is defined as follows.

Problem 1. In the **maximum flow** problem, we are given an undirected graph $G = (V, E)$ and two vertices $s, t \in V$, and the goal is to find the maximum flow from s to t without sending more than one unit of flow over each edge.

We formulate the unit-capacity maximum flow problem as the following packing linear program (LP). Let $\mathcal{P}_{s,t}$ denote the set of all s-t paths in G.

$$\begin{aligned} \max_{f \in \mathbb{R}^{\mathcal{P}_{s,t}}} \quad & \sum_{p \in \mathcal{P}_{s,t}} f_p \\ \text{subject to} \quad & \sum_{p \ni e} f_p \leqslant 1 \qquad \forall e \in E \\ & f_p \geqslant 0 \qquad \forall p \in \mathcal{P}_{s,t}. \end{aligned}$$

You might have seen different formulations of the maximum flow problem previously, but it is easy also to verify that the above formulation works.

One of the earliest algorithms for maximum flow is the Ford-Fulkerson algorithm that solves this problem in $O(m \cdot F)$ time where F is the value of maximum flow which can be at most $O(n)$ on unit-capacity graphs. In this lecture, we discuss some of the ideas behind the breakthrough algorithm of [CKM+11] that achieved a runtime of $\widetilde{O}(m^{4/3} \cdot \text{poly}(1/\varepsilon))$ for $(1+\varepsilon)$-approximation of this problem. Specifically, we will cover an $\widetilde{O}(m^{3/2} \cdot \text{poly}(1/\varepsilon))$ time algorithm for this problem as the main topic, and leave extending this to the final result as an optional topic for interested readers in Appendix A[1].

1.1 Maximum Flow via MWU

The starting point of the work of [CKM+11] is a *multiplicative weight update* (MWU) method for solving maximum flow generally. To solve this problem using the MWU algorithm, we follow these steps.

Step 1: Feasibility LP. We first turn this problem into a feasibility LP such that given any value $F \geqslant 0$, we want to test if the following LP is feasible:

$$\text{Polytope (P) in } \mathbb{R}^{\mathcal{P}_{s,t}}:$$

$$\begin{aligned} &\sum_{p \in \mathcal{P}_{s,t}} f_p \geqslant F \\ &\sum_{p \ni e} f_p \leqslant 1 \qquad \forall e \in E \\ &f_p \geqslant 0 \qquad \forall p \in \mathcal{P}_{s,t}. \end{aligned}$$

Having solved this LP (approximately), we can use binary search on F to solve the original problem also. As a notation, for every edge $e \in E$, we define

$$f_e := \sum_{p \ni e} f_p,$$

to denote the total flow passed through the edge e.

Step 2: Easy and hard constraints. The next step is to partition the constraints into *"easy"* constraints and *"hard"* constraints: the easy constraints are the ones that we think we can easily satisfy "on our own", while the hard constraints are the ones we need the help of the MWU to satisfy. Specifically, we consider the easy constraints here to be the first constraint of (P) on the total flow as well as the non-negativity constraints on each path. We are thus left with $m := |E|$ hard constraints corresponding to the capacity constraints on the edges.

Step 3: Oracle LP. Let $\{w_e \mid e \in E\}$ be any set of positive weights on the hard constraints. If the original LP (P) is satisfiable, then, if we take a convex combination of its "hard" constraints weighted by w_e's, the single resulting constraint should also be satisfiable; this leads us to define the following **oracle LP**:

$$\text{Oracle LP (O) in } \mathbb{R}^{\mathcal{P}_{s,t}}:$$

$$\begin{aligned} &\sum_{e \in E} w_e \cdot \underbrace{\sum_{p \ni e} f_p}_{f_e} \leqslant \sum_{e \in E} w_e \ := W \\ &\sum_{p \in \mathcal{P}_{s,t}} f_p \geqslant F \\ &f_p \geqslant 0 \qquad \forall p \in \mathcal{P}_{s,t}. \end{aligned}$$

[1] Note that we did not cover this final algorithm in the class completely, and only briefly touched on its high level ideas.

We again emphasize that regardless of which weights we choose, if (P) is satisfiable, (O) is also satisfiable (although the reverse direction is not necessarily true always). It is also worth mentioning that the problem in (O) is to find an F-flow which is not necessarily unit-capacity anymore (we have no capacity constraint), but "on average", each edge should still only route one unit of flow, where the average is by the given weights.

The main step in using the MWU framework is to figure out an *efficient* way of solving the oracle LP (efficiency means a couple of different things here simultaneously, and we elaborate more on this later). The rest is taken care of by the MWU algorithm as we saw in the previous lectures and repeated here again.

Algorithm 1. The MWU Algorithm for Maximum Flow. Given an undirected graph $G = (V, E)$, vertices s and t, integer $F \geqslant 1$, and parameter $\varepsilon \in (0, 1)$, returns a unit-capacity s-t flow f with value at least $(1 - \varepsilon) \cdot F$ or declare maximum flow in G is less than F.

The algorithm uses two parameters $\rho \geqslant 1$ and $T \in \mathbb{N}$ to be determined later.

1. Start with $w_e^{(1)} = 1$ for every edge $e \in E$.
2. For $t = 1$ to T iterations:
   (a) Solve the oracle LP (O) with weights $\left\{w_e^{(t)} \mid e \in E\right\}$ to obtain a F-flow $f^{(t)}$ (not necessarily unit-capacity); if the oracle LP is infeasible, output (P) is also infeasible.
   (b) For every $e \in E$, update
   $$w_e^{(t+1)} := w_e^{(t)} \cdot \left(1 + \frac{\varepsilon}{2\rho} \cdot f_e^{(t)}\right).$$
3. Return $\bar{f} := (1 - \varepsilon) \cdot \frac{1}{T} \cdot \sum_{i=1}^{T} f^{(t)}$ as the final flow obtained by the algorithm.

Let us parse the MWU algorithm now. It starts by putting uniform weights over the constraints (edge capacity constraints), namely, "treating them equally", and obtain a F-flow $f^{(1)}$ based on that, namely, a flow that is on "average" unit-capacity. The algorithm then adjusts the weights over the edges: by "penalizing" edges that already passed a flow in this iteration. This has the following natural effect; in the next iteration that we are running the oracle LP, the edges with less flow over them in previous iterations will be given "more priority" so that the oracle is better off by routing flow over them.

The following theorem captures the guarantee of the MWU algorithm.

Theorem 1. *There is an absolute constant $b \geqslant 1$ such that the following is true. Suppose we run Algorithm 1 on any undirected unit-capacity graph $G = (V, E)$, integer $F \geqslant 1$, and parameter $\varepsilon \in (0, 1/100)$. If*

$$\rho \geqslant \max_{t \in [T], e \in E} f_e^{(t)} \qquad \text{and} \qquad T = b \cdot \left(\frac{\rho \cdot \log m}{\varepsilon^2}\right)$$

then, the returned flow $\bar{f}$ is a unit-capacity flow of value at least $(1 - \Theta(\varepsilon)) \cdot F$ or the algorithm correctly identifies that maximum flow in G is less than F.

Moreover, if when solving the oracle in Line (2a) of Algorithm 1, instead of returning an F-flow, the algorithm returns a $(1-\Theta(\varepsilon))F$-flow, the final solution remains a $(1-\Theta(\varepsilon))$-approximate unit-capacity flow.

The proof of this theorem is the same exact ones as in Lectures 17 and 18 and we do not repeat it here.

1.2 The Plan for the Maximum Flow Algorithm

Given Algorithm 1 and Theorem 1, our plan for designing a maximum flow algorithm is the following. We simply run Algorithm 1 and only have to design an "efficient" way of solving the oracle LP in each iteration

in Line (2a). But, what does efficient mean here?

- **Time efficiency:** Firstly, we need to solve this step *fast enough* if our goal is to obtain a fast algorithm at the end for the entire maximum flow problem also;
- **Width efficiency:** Secondly, while our hand is technically open here and we are not restricted to finding a unit-capacity flow, we still need want to find a F-flow which is "close" to unit-capacity; in other words, we would like each flow $f^{(t)}$ that we find to not route "too much" flow through any single edge. This is because the maximum flow $f_e^{(t)}$ routed through any edge determines the parameter ρ of Theorem 1 (the **width** of the oracle). Thus, our goal is to keep this parameter as small as possible so that the number of iterations of Algorithm 1 remains low, leading to a faster final algorithm.

These two goals are clearly at odds with each other. Let us show two extreme approach for designing the oracle that demonstrate this.

An obvious oracle focusing on width efficiency. If our goal is to minimize the value of ρ in the algorithm, there is clearly an optimal strategy: simply compute a unit-capacity F-flow and return it as the answer to the oracle. This ensures that $f_e \leqslant 1$ for every edge which means we can set $\rho = 1$ even.

Of course, the problem with this oracle is that it is too restrictive: the only way to do this is to solve the original problem! Thus, this way, we get no benefit from using MWU.

A simple oracle focusing on time efficiency. On the other hand, if we solely focus on solving the oracle LP, we can do the following. Let us rewrite the weighted constraint of the oracle LP as:

$$\sum_{e \in E} w_e \cdot \sum_{p \ni e} f_p = \sum_{p \in \mathcal{P}_{s,t}} f_p \sum_{e \in p} w_e \leqslant \sum_{e \in E} w_e := W$$

Notice that the second term from left basically "charges" each path $p \in \mathcal{P}_{s,t}$ proportional to the total weight of the path. Thus, to satisfy the oracle LP, we can simply find the *s-t (weighted) shortest path* in G using the weights w_e on the edges $e \in E$ of G. Let p be this path. We then set $f_p = F$ and $f_{p'} = 0$ for $p' \neq p \in \mathcal{P}_{s,t}$. This is a feasible solution for the oracle LP as long as the oracle LP has a feasible solution (which is the case if the original LP (P) is feasible as argued earlier). The reason for this to be true is because if a combination of F paths with different weights can have a total weight no more than W, then picking the shortest path F times cannot increase the total weight, and thus leads to an oracle solution.

In terms of time efficiency, this oracle is pretty efficient and simple. We can just run Dijsktra's algorithm and find its answer in $\widetilde{O}(m)$ time. On the other hand, in terms of width, we have $\rho = F$ because in each step we are routing F units of flow over some edge. The end result is that even though we can solve each oracle step in $\widetilde{O}(m)$ time, we also need $\widetilde{O}(F/\varepsilon^2)$ steps, leading to an algorithm with $\widetilde{O}(m \cdot F/\varepsilon^2)$ time – this is weaker than the original Ford-Fulkerson algorithm we already had on every front! That being said, we at least now have an actual complete algorithm for solving maximum flow which serves the purpose of giving a very simple application of the MWU method and setting up the stage for the next oracle algorithm.

2 Next Attempt: A Stronger Oracle for Maximum Flow

We now present a stronger oracle for the maximum flow problem based on [CKM+11]. Consider the basic oracle of the previous subsection. It did not achieve a good runtime because the number of iterations of MWU was high due to width being $\rho = \Omega(F)$. Our goal now is to obtain an oracle with a smaller width.

The basic approach of subprevious section based on shortest path computation results in a large width for this oracle because it routes all the flow through just a single path, leading to $f_e = F$ on the edges of the path. Thus, our goal will be to find a flow of value F which is "well spread", namely, does not increase the value of f_e on any edge too much. On the other hand, we also do not want to go to the extreme case of width parameter $\rho = 1$ which requires us to solve the original maximum s-t flow problem. But, we can

exploit the fact that in the oracle LP, we do not need to preserve the capacity constraint exactly, rather, we only try not too violate it "too badly" so that the width remains small. We do this using the notion of *electrical flows* which we define next.

2.1 Detour: Electrical Flows

Let $G = (V, E)$ be any undirected graph with given function $r : E \to \mathbb{R}_{\geqslant 0}$ over the edges. Interpret G as an electrical network where an edge e is replaced by a **resistor** of resistance $r(e)$ ohm.

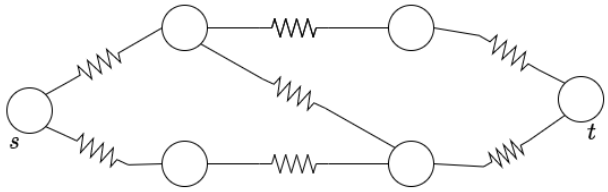

Figure 1: An electrical network interpretation of a graph.

Now suppose we attach a battery to vertices s and t (with the positive voltage at s) to induce a **current** $i : E \to \mathbb{R}$ from s to t and a **voltage potential** $\phi : V \to \mathbb{R}$ on the vertices. By **Ohm's law** from physics, we have that for every edge $e = (u, v)$,

$$i(e) = \frac{\phi(u) - \phi(v)}{r(e)}.$$

Definition 2. An **electrical flow** $f_e \in \mathbb{R}^E$ of value F on an undirected graph $G = (V, E)$ with resistors $r : E \to \mathbb{R}_{\geqslant 0}$ is the unique s-t flow corresponding to the current $i : E \to \mathbb{R}$, i.e., $f_e = i(e)$, that sends F units of current from s to t in the electrical network represented by G.

Remark. Let us take an even further detour for readers with some familiarity with spectral graph theory. Suppose G has a unit resistor on every edge. Then, the Ohm law can be written as $i = B \cdot \phi$ and $B^\top i = F \cdot (\chi(s) - \chi(t))$ where B is the signed edge-incidence matrix of the graph and $\chi(s)$ and $\chi(t)$ are characteristic vectors of s and t in $\mathbb{R}^V$. Combining these together implies that $F \cdot (\chi(s) - \chi(t)) = B^\top B \cdot \phi = L \cdot \phi$ where L is Laplacian of G. Thus by solving this Laplacian system we can obtain the vector of vertex potentials and then use Ohm's law to obtain the electrical flow as well.

While the above interpretation is perhaps more natural from a physical point of view, we will use an alternative definition of electrical flows as a purely optimization problem. We need this equivalent interpretation of electrical flows for this lecture. We will not prove the equivalence of these two definitions in this course and instead refer the interested reader to the monograph by [Vis13].

Definition 3. Given any undirected graph $G = (V, E)$ with resistors $r : E \to \mathbb{R}_{\geqslant 0}$, the **energy** of a s-t flow $f : E \to \mathbb{R}$ (that satisfies preservation of flow but not necessarily capacity constraint) is

$$\text{Energy}(f) := \sum_{e \in E} r(e) \cdot f_e^2.$$

An **electrical flow** of value F is the unique flow f with value F from s to t that *minimizes* $\text{Energy}(f)$.

For us, the important fact is that (approximate) electrical flows (or rather approximate energy-minimizing flows) can be found efficiently using the seminal result of [ST14] on Laplacian solvers.

Proposition 4 (cf. [CKM+11, ST14])**.** *There exists an algorithm that given any undirected graph $G = (V, E)$ with resistors $r : E \to \mathbb{R}_{\geqslant 0}$ and parameter $\delta > 0$, outputs a s-t flow f such that*

$$\text{Energy}(f) \leqslant (1 + \delta) \cdot \text{Energy}(\tilde{f}),$$

where $\tilde{f}$ is the energy-minimizing s-t flow in G (i.e., $\tilde{f}$ is the electrical flow). The algorithm runs in $\widetilde{O}(\frac{m \log (R+1)}{\delta})$ time where R is the ratio of the largest to smallest resistance.

2.2 Back to the Oracle

We are going to use Proposition 4 as our oracle. In particular, given the weights $\{w_e \mid e \in E\}$, we are going to define the resistors

$$r(e) = w_e + \frac{\varepsilon}{m} \cdot W,$$

for each edge $e \in E$ where $W = \sum_{e \in E} w_e$ is the total weight defined earlier. The reason for adding the extra additive term is to ensure that none of the resistances become too small (we clarify this further on). We then compute an approximate energy-minimizing flow f with parameter $\delta = \varepsilon$ on this network using Proposition 4. We first argue that this flow *approximately* preserves the oracle condition (not exactly, but by the moreover part of Theorem 1 this is enough).

Lemma 5. *As long as (P) is feasible, the returned flow f by the oracle satisfies*

$$\sum_{e \in E} w_e \cdot f_e \leqslant (1 + 3\varepsilon) \cdot W;$$

in other words, $f/(1 + 3\varepsilon)$, satisfies the oracle condition and has value at least $F/(1 + 3\varepsilon)$.

Proof. The first step is to bound the energy of f using its approximate energy-minimizing property.

Claim 6. $\text{Energy}(f) \leqslant (1 + 3\varepsilon) \cdot W$.

Proof of Claim 6. Let $\tilde{f}$ be the electrical flow and f^* be the optimal s-t flow with unit-capacity in G. If (P) is feasible, we have that value of f^* is at least F, thus f^* is also a feasible solution for the energy-minimization flow problem of $\tilde{f}$, and as such $\text{Energy}(\tilde{f}) \leqslant \text{Energy}(f^*)$. In f^*, the flow over each edge is at most 1 as it satisfies capacity constraints, thus,

$$\text{Energy}(f^*) = \sum_{e \in E} r(e) \cdot {f_e^*}^2 \leqslant \sum_{e \in E} r(e) = \sum_{e \in E} (w_e + \frac{\varepsilon}{m} \cdot W) = (\sum_{e \in E} w_e) + \varepsilon \cdot W = (1 + \varepsilon) \cdot W,$$

Thus, we have,

$$\text{Energy}(f) \leqslant (1 + \varepsilon) \cdot \text{Energy}(\tilde{f}) \leqslant (1 + \varepsilon) \cdot \text{Energy}(f^*) \leqslant (1 + \varepsilon) \cdot (1 + \varepsilon) \cdot W \leqslant (1 + 3\varepsilon) \cdot W,$$

concluding the proof. □

We continue with the proof of Lemma 5. Notice that by Claim 6 we managed to bound a (weighted) variant of ℓ_2-norm of f and we now would like to use that to bound its ℓ_1-norm (with the same weights) to prove the lemma. We have,

$$\begin{aligned} \sum_{e \in E} w_e \cdot f_e &= \sum_{e \in E} \sqrt{w_e} \cdot \sqrt{w_e \cdot f_e^2} \\ &\leqslant \sqrt{\sum_{e \in E} \sqrt{w_e}^2} \cdot \sqrt{\sum_{e \in E} \sqrt{w_e \cdot f_e^2}^2} && \text{(by Cauchy-Schwartz inequality)} \end{aligned}$$

$$
\begin{aligned}
&= \sqrt{\sum_{e\in E} w_e} \cdot \sqrt{\sum_{e\in E} w_e \cdot f_e^2} \\
&\leqslant \sqrt{W} \cdot \sqrt{\text{Energy}(f)} && \text{(as } w_e \leqslant r(e) \text{ and by the definition of Energy}(f)) \\
&\leqslant \sqrt{W \cdot (1+3\varepsilon) \cdot W} \leqslant (1+3\varepsilon) \cdot W, && \text{(by Claim 6)}
\end{aligned}
$$

finalizing the proof of Lemma 5. □

We now bound the width of the resulting (approximate) oracle solution, crucially using the fact that we ensured each resistance is at least $(\varepsilon/m) \cdot W$ and the bound on the energy of f.

Lemma 7. *The returned flow f by the oracle satisfies that for all $e \in E$,*

$$f_e \leqslant \sqrt{2m/\varepsilon}.$$

Proof. Suppose towards a contradiction that there exists an edge $e' \in E$ with $f_{e'} > \sqrt{2m/\varepsilon}$. Then, we have,

$$\text{Energy}(f) = \sum_{e\in E} r(e) \cdot f_e^2 \geqslant r(e') \cdot f_{e'}^2 \geqslant \frac{\varepsilon}{m} \cdot W \cdot \frac{2m}{\varepsilon} = 2W.$$

This contradicts Claim 6 that bounds $\text{Energy}(f) \leqslant (1+3\varepsilon)$ (when $\varepsilon < 1/3$, which we can assume w.l.o.g.). □

To conclude, the runtime for each oracle is $\widetilde{O}(m/\varepsilon)$ by Proposition 4. By Theorem 1, after running this oracle for $\widetilde{O}(\sqrt{m/\varepsilon} \cdot \frac{1}{\varepsilon^2})$ iterations (given that $\rho = \sqrt{2m/\varepsilon}$), we obtain a unit-capacity flow of value $(1 - \Theta(\varepsilon)) \cdot F$ (or the oracle fails at some step and we can safely declare maximum flow of G is less than F). By scaling ε by a constant factor and running a binary search over the choice of F, we can use this to obtain an approximate maximum flow algorithm, proving the following theorem.

Theorem 8 (Weak version of [CKM+11])**.** *There is an algorithm that given any undirected unit-capacity graph G and vertices s and t, finds a $(1-\varepsilon)$-approximate maximum s-t flow in G in $\widetilde{O}(\varepsilon^{-2.5} \cdot m^{1.5})$ time.*

While this is now an improvement over Ford-Fulkerson and some "more classical" maximum flow algorithms, it is still not good enough. Already in 70's, we knew $\widetilde{O}(m^{1.5})$ time algorithms (due to Karzanov and independently Even and Tarjan) for solving directed capacitated maximum flow exactly. Thus, Theorem 8 still did not provide any improvements over previously known algorithms. But, it sets the stage for the very final algorithm which is just a simple, yet quite clever, modification of this algorithm. We postpone this result to Appendix A for the interested reader and conclude the "official" study of the MWU in our course by the following remark.

Remark. In the context of LPs and many other optimization problems, the MWU method can also be seen alternatively as a **boosting framework**: one can obtain a good approximation for a problem by designing a "much weaker approximation" for a more general variant of the problem (weighted by the MWU weight), where this weak approximation notion can be interpreted quite broadly[a] – this weak approximation ratio then translates to the number of iterations of the MWU method.

[a] E.g., in the context of maximum flow, this was a flow that did not violate capacity constraints by more than $O(\sqrt{m})$. In the previous lecture, we saw two another version, one simply being finding any constant approximation to matching, and the more efficient one, being solving a certain weighted matching problem.

References

[CKL+22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 612–623. IEEE, 2022. 13

[CKM+11] Paul F. 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 Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 273–282. ACM, 2011. 2, 4, 6, 7, 10, 12, 13

[Kar98] David R. Karger. Better random sampling algorithms for flows in undirected graphs. In Howard J. Karloff, editor, *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 25-27 January 1998, San Francisco, California, USA*, pages 490–499. ACM/SIAM, 1998. 13

[ST14] Daniel A. Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM J. Matrix Anal. Appl.*, 35(3):835–885, 2014. 6

[Vis13] Nisheeth K. Vishnoi. Lx = b. *Found. Trends Theor. Comput. Sci.*, 8(1-2):1–141, 2013. 5

A Optional Topic: An Even Stronger Oracle for Maximum Flow

We now see our final algorithm. The idea is to *explicitly* maintain an upper bound on the width of the oracle. Let $\rho^\star$ be a parameter to be determined later. We are going to ensure that in every iteration of Algorithm 1, the flow f returned by the oracle satisfies $f_e \leqslant \rho^\star$ for all $e \in E$. This definitely implies the width parameter $\rho \leqslant \rho^\star$ also. But how do we ensure this bound?

The new algorithm. We run the same exact oracle as before (by computing electrical flows via Proposition 4 using the same resistances defined based on weights) and look at the returned flow. If all edges satisfy $f_e \leqslant \rho^\star$, we pass this solution to Algorithm 1 and go to the next iteration. Otherwise, if there is an edge e with $f_e > \rho^\star$, we will delete e from G (for this iteration and in the future), and recompute the oracle. This way, *by design*, the width of the oracle is at most $\rho^\star$. This concludes the description of the new algorithm.

Of course we now need to take care of the following. Define D as the number of deleted edges and recall that T is the number of iterations of Algorithm 1:

- We need to ensure $D \leqslant \varepsilon \cdot F$ as otherwise, we may delete so many edges from G that it no longer can even support a unit-capacity $(1-\varepsilon) \cdot F$-flow (even if it could originally supported a unit-capacity F-flow). This means MWU has no chance of returning a $(1-\varepsilon)$-approximate solution.
- The runtime of the new algorithm is now $\widetilde{O}(m/\varepsilon) \cdot (D+T)$, because each time we run Proposition 4, we either move to the next iteration of the MWU algorithm in Algorithm 1 (and account for it in T), or we delete one more edge from the graph (and so account for it in D).

The remainder of the analysis shows the effect of the choice of $\rho^\star$ on the parameter D. We need yet another quick detour.

A.1 Detour: More on Electrical Flows

For a graph $G = (V, E)$ and integer $F \geqslant 1$, define $\text{Energy}_F(G)$ as $\text{Energy}(f)$ where f is the s-t electrical F-flow in G (with a certain resistance over its edges). In other words, $\text{Energy}_F(G)$ is the minimum energy possible for any F-flow in G between s and t. We note that $\text{Energy}_F(G)$ is closely related to the notion of *effective resistance* between s and t (it is precisely the effective resistance times F^2); however, it will be easier to simply work with energy of the G in the remainder of this lecture.

We first have that energy of G is a monotone function of its resistances.

Fact 9. *Let G be an undirected graph with resistances $r : E \to \mathbb{R}_{\geqslant 0}$. Suppose we increase resistance of some edge $e \in E$ to $r'(e) \geqslant r(e)$ and let G' be the new graph. Then, for all $F \geqslant 1$,*

$$\text{Energy}_F(G') \geqslant \text{Energy}_F(G).$$

Proof. Any F-flow f in G' is also an F-flow in G and $\text{Energy}(f)$ is a monotone function of resistances. □

A more interesting property is that deleting a "high energy" edge from G increases the total energy of the graph considerably.

Proposition 10. *Let G be an undirected graph with resistances $r : E \to \mathbb{R}_{\geqslant 0}$. Suppose the edge $e \in E$ contributes $r(e) \cdot f_e^2 \geqslant \beta \cdot \text{Energy}_F(G)$ to the energy in the electrical F-flow for some $\beta \in (0,1)$. Then, if we increase $r'(e) \leftarrow \infty$ (i.e., delete this edge from G) to obtain G', we have,*

$$\text{Energy}_F(G') \geqslant \frac{1}{1-\beta} \cdot \text{Energy}_F(G).$$

Intuitively, Proposition 10 says the following: while the electrical flow f was trying to minimize the energy, it still had to "route β-fraction of its energy" through the edge e; this means that the remainder of the graph, can only "support" $(1-\beta)$-fraction of a F-flow in an "energy minimizing way". But, now that we deleted the edge e, the reminder of the F-flow needs to go through that part of the graph which increases the total energy by a $1/(1-\beta) \approx (1+\beta)$ factor. That being said, making this intuition formal, requires a couple of lines of proofs, which we omit here; but you can find a formal proof in [CKM+11, Lemma 2.6].

A.2 Back to the Oracle

We can now analyze the new algorithm. In the following, we are going to **assume** that $D \leqslant \varepsilon \cdot F$, i.e., the number of deletions never become more than $\varepsilon \cdot F$. We will then show how we can justify this assumption by picking a "right" choice for $\rho^\star$ and D.

We first have an analogue of Lemma 5 even under the edge deletions imposed in the new oracle.

Lemma 11. *As long as (P) is feasible and $D \leqslant \varepsilon \cdot F$, the returned flow f by the oracle in each step satisfies*

$$\sum_{e \in E} w_e \cdot f_e \leqslant (1+6\varepsilon) \cdot W;$$

in other words, $f/(1+6\varepsilon)$, satisfies the oracle condition and has value at least $F/(1+6\varepsilon)$.

Proof. Exactly as in Lemma 5, we start by bounding the energy of f using its approximate energy-minimizing property.

Claim 12. $\text{Energy}(f) \leqslant (1+6\varepsilon) \cdot W$.

Proof of Claim 12. Let $\tilde{f}$ be the electrical flow and f^* be the optimal s-t flow with unit-capacity in G', the current graph obtained from G after the deletions done so far. We have that value of f^* is at least $(1-\varepsilon) \cdot F$, thus $f^*/(1-\varepsilon)$ is also a feasible solution for the energy-minimization flow problem of $\tilde{f}$, and as such $\text{Energy}(\tilde{f}) \leqslant \text{Energy}(f^*/(1-\varepsilon))$. In f^*, the flow over each edge is at most 1 as it satisfies capacity constraints, thus, in $f^*/(1-\varepsilon)$, we have,

$$\text{Energy}(f^*/(1-\varepsilon)) = \sum_{e \in E} r(e) \cdot (f^*_e/(1-\varepsilon))^2 \leqslant (1+3\varepsilon) \cdot \sum_{e \in E} r(e) \leqslant \sum_{e \in E} (w_e + \frac{\varepsilon}{m} \cdot W) = (1+4\varepsilon) \cdot W,$$

Thus, we have,

$$\text{Energy}(f) \leqslant (1+\varepsilon) \cdot \text{Energy}(\tilde{f}) \leqslant (1+\varepsilon) \cdot \text{Energy}(f^*/(1-\varepsilon)) \leqslant (1+\varepsilon) \cdot (1+4\varepsilon) \cdot W \leqslant (1+6\varepsilon) \cdot W,$$

concluding the proof. □

The rest of the proof is identical to Lemma 5 and is simply repeated here.

$$\begin{aligned}
\sum_{e \in E} w_e \cdot f_e &= \sum_{e \in E} \sqrt{w_e} \cdot \sqrt{w_e \cdot f_e^2} \\
&\leqslant \sqrt{\sum_{e \in E} \sqrt{w_e}^2} \cdot \sqrt{\sum_{e \in E} \sqrt{w_e \cdot f_e^2}^2} && \text{(by Cauchy-Schwartz inequality)} \\
&= \sqrt{\sum_{e \in E} w_e} \cdot \sqrt{\sum_{e \in E} w_e \cdot f_e^2} \\
&\leqslant \sqrt{W} \cdot \sqrt{\text{Energy}(f)} && \text{(as } w_e \leqslant r(e) \text{ and by the definition of Energy}(f)) \\
&\leqslant \sqrt{W \cdot (1+6\varepsilon) \cdot W} \leqslant (1+6\varepsilon) \cdot W, && \text{(by Claim 12)}
\end{aligned}$$

finalizing the proof of Lemma 11. □

Lemma 11 means that we can continue to use f as an oracle solution in Algorithm 1. We will also use the bound of Claim 12 (even in a weaker form, namely, by upper bounding $\text{Energy}(f) \leqslant (1+6\varepsilon)W \leqslant 2W$) for upper bounding the value of D. In particular, notice that since we are only increasing the weights and by extension resistances during the iterations of the algorithm (and deleting edges which is the same as increasing their resistances to ∞), we can Fact 9 to say that $\text{Energy}_F(G)$ is keep increasing in every step. We now use this as a potential function argument (this is actually quite similar to the proof of correctness of the MWU framework itself).

Claim 13. *Let G_{start} be the original graph with the original resistances $r_1(e) = (1+\varepsilon)$ and G_{final} be the final graph and its resistances. Then,*

$$\text{Energy}_F(G_{start}) \geqslant \left(\frac{F}{m}\right)^2 \qquad \textit{and} \qquad \text{Energy}_F(G_{final}) \leqslant m^{O(1/\varepsilon)}.$$

Proof. For the first bound, since we are routing an electrical F-flow f over m edges, at least one edge needs to receive a flow of F/m. That edge alone contributes at least $(F/m)^2$ to $\text{Energy}(f)$ which means $\text{Energy}_F(G_{start}) = \text{Energy}(f) \geqslant (F/m)^2$ as desired.

For the second bound, by Claim 12, we know that $\text{Energy}_F(G_{final}) \leqslant 2 \cdot W^{(T)}$. Thus, we need to bound $W^{(T)}$ only (this is the step which is already done in the analysis of MWU and the proof of Theorem 1). For every $t \geqslant 1$, we have,

$$\begin{aligned}
W^{(t+1)} &= \sum_{e\in E} w_e^{(t+1)} && \text{(by the definition of } W^{(t+1)}\text{)}\\
&= \sum_{e\in E} \left(1 + \frac{\varepsilon}{\rho}\cdot f_e^{(t)}\right)\cdot w_e^{(t)} && \text{(by the update rule of Algorithm 1)}\\
&= \sum_{e\in E} w_e^{(t)} + \frac{\varepsilon}{\rho}\cdot\left(\sum_{e\in E} w_e^{(t)}\cdot f_e^{(t)}\right) && \text{(by rearranging the terms)}\\
&= W^{(t)} + \frac{\varepsilon}{\rho}\cdot\left(\sum_{e\in E} w_e^{(t)}\cdot f_e^{(t)}\right) && \text{(by the definition of } W^{(t)}\text{)}\\
&\leqslant W^{(t)} + \frac{\varepsilon}{\rho}\cdot(1+6\varepsilon)\cdot W^{(t)}\\
&\qquad \text{(by the approximate feasibility of } f^{(t)} \text{ in the oracle LP (O) in Lemma 11)}\\
&\leqslant (1+\frac{2\varepsilon}{\rho})\cdot W^{(t)}.
\end{aligned}$$

Given that the total number of iterations of Algorithm 1 is $T = O(\rho\cdot\log(m)/\varepsilon^2)$ by Theorem 1 and the original value of $W^{(1)} = m$, we have,

$$W^{(T)} \leqslant (1+\frac{2\varepsilon}{\rho})^T \cdot m \leqslant \exp\left(\frac{2\varepsilon}{\rho}\cdot T + \log m\right) = \exp\left(O(\log(m)/\varepsilon)\right) = m^{O(1/\varepsilon)},$$

where in the second inequality we used $(1+x) \leqslant e^x$ for all x. □

Finally, we can bound D using these bounds and Proposition 10 – whenever we delete an edge, we increase energy of the graph dramatically and at the same time, we know that this energy cannot get too large also. In the following, we are going to use the bound of $\rho^\star = o(\sqrt{m/\varepsilon})$; we already know that $\rho^\star$ we are going to set should satisfy this bound as this is the width of the previous oracle and if we cannot set $\rho^\star$ considerably less than this, we will not get any improvement using the new algorithm.

Lemma 14. *As long as (P) is feasible and $D \leqslant \varepsilon\cdot F$, and assuming $\rho^\star = o(\sqrt{m/\varepsilon})$, we have*

$$D = O(\frac{m\cdot\log m}{\varepsilon^2\cdot\rho^{\star 2}}).$$

Proof. Consider a step wherein we delete an edge e because $f_e > \rho^\star$. Firstly, by Claim 12, we know that

$$\text{Energy}_F(G) \leqslant 2W,$$

at this point. At the same time, since we deleted the edge e and since $r(e) \geqslant (\varepsilon/m) \cdot W$, we have that

$$r_e \cdot f_e^2 \geqslant \frac{\varepsilon}{m} \cdot W \cdot \rho^{\star 2};$$

thus, if we set

$$\beta = \frac{\varepsilon \cdot \rho^{\star 2}}{2m},$$

we get that e was contributing β fraction of $\text{Energy}_F(G)$ to $\text{Energy}(f)$.

We would have ideally liked to apply Proposition 10 right away here but note that here f is *not* an electrical F-flow, but rather a $(1+\varepsilon)$-approximation to it obtained from Proposition 4. However, even though we did not state this explicitly, Proposition 4 actually provides a stronger guarantee the energy contribution of each edge in f is within an additive $\Theta(\varepsilon/m) \cdot \text{Energy}(f)$ factor of the energy contribution in the true electrical flow (see [CKM+11, Theorem 2.3]). Given the bound of $\rho^\star = o(\sqrt{m/\varepsilon})$, we obtain that the contribution of the energy of e in the electrical flow is also at least $\Omega(\beta)$ fraction of the total energy.

Now, by applying Proposition 10, we obtain that after this step, by setting G' as the new graph,

$$\text{Energy}_F(G) \geqslant \left(1 + \Theta\left(\frac{\varepsilon \cdot \rho^{\star 2}}{m}\right)\right) \cdot \text{Energy}_F(G).$$

Using this, the monotonicity of $\text{Energy}_F(G)$ throughout the algorithm, and the first bound of Claim 13, we have that after D deletions,

$$\text{Energy}_F(G) \geqslant \left(1 + \Theta\left(\frac{\varepsilon \cdot \rho^{\star 2}}{m}\right)\right)^D \cdot \left(\frac{F}{m}\right)^2 \geqslant \exp\left(\Theta\left(\frac{D \cdot \varepsilon \cdot \rho^{\star 2}}{m}\right)\right) \cdot \left(\frac{F}{m}\right)^2,$$

where we used the inequality $1 + x \geqslant e^{x/2}$ for $x = o(1)$ which holds here because $\rho^\star = o(\sqrt{m/\varepsilon})$. Since we also have that $\text{Energy}_F(G) = m^{O(1/\varepsilon)}$ by the second bound of Claim 13, we have,

$$\Theta\left(\frac{D \cdot \varepsilon \cdot \rho^{\star 2}}{m}\right) = O\left(\frac{\log m}{\varepsilon}\right) \implies D = O\left(\frac{m \log m}{\varepsilon^2 \cdot \rho^{\star 2}}\right),$$

which concludes the proof. □

To finalize the proof, notice that by Lemma 14 under the assumption that $D \leqslant \varepsilon \cdot F$, we now have that the runtime of the algorithm is

$$\widetilde{O}(m/\varepsilon) \cdot (D + T) = \widetilde{O}(m/\varepsilon) \cdot \left(O\left(\frac{m \cdot \log m}{\varepsilon^2 \cdot \rho^{\star 2}}\right) + O\left(\frac{\rho^\star \cdot \log m}{\varepsilon^2}\right)\right);$$

this means that we should pick $\rho^\star = m^{1/3}$ and obtain that the runtime of the algorithm is now

$$\widetilde{O}(m^{4/3} \cdot \text{poly}(1/\varepsilon)),$$

as desired. The only question now is that how can we ensure the bound $D \leqslant \varepsilon \cdot F$? This can be easily done as follows. By the choice of $\rho^\star$, we would like to set $D = \widetilde{O}(m^{1/3} \cdot \text{poly}(1/\varepsilon))$. Fix this particular choice of D. Before we start running this algorithm, we first run Ford-Fulkerson algorithm with a bound of D/ε on the maximum flow value to see if G has a flow of size at least D/ε which means we can now focus only on values of F satisfying $D \leqslant \varepsilon \cdot F$. The Ford-Fulkerson step also takes only $O(m \cdot D/\varepsilon) = \widetilde{O}(m^{4/3} \cdot \text{poly}(1/\varepsilon))$ time which is within our budget. This allows us to conclude the following theorem.

Theorem 15 ([CKM+11])**.** *There is an algorithm that given any undirected unit-capacity graph G and vertices s and t, finds a $(1-\varepsilon)$-approximate maximum s-t flow in G in $\widetilde{O}(m^{4/3} \cdot \text{poly}(1/\varepsilon))$ time.*

This concludes the presentation of the main algorithm of [CKM+11] in our lecture. We note that [CKM+11] actually improve the runtime of this algorithm to $\widetilde{O}(mn^{1/3} \cdot \text{poly}(1/\varepsilon))$ using randomization as a black-box application of their result and the prior graph smoothing and sampling techniques of [Kar98].

Remark. The algorithm from [CKM+11] that we covered in this lecture can be seen as one of the earliest "modern" graph algorithms for maximum flow, which led to several subsequent breakthroughs, culminating in the $m^{1+o(1)}$ time algorithm of [CKL+22] for directed capacitated (exact) maximum flow.

Maximum flow has a remarkably rich history in theoretical computer science and optimization literature and its study has been closely intertwined with the entire field algorithm design. Reviewing this vast literature is quite beyond the scope of this lecture and we instead refer interested readers to [CKL+22] for more on the history of this problem.