CS 466/666: Algorithm Design and Analysis **University of Waterloo: Fall 2024**

Lecture 18

November 12, 2024

Instructor: Sepehr Assadi

Topics of this Lecture

1 A Faster MWU Algorithm for Bipartite Matching

In the last lecture, we saw how to use the Multiplicative Weight Update (MWU) technique to approximate the fractional matching LP. As we also discussed, almost everything covered in the previous lecture can be readily generalized to solving other LPs if our goal is to find an approximately feasible solution (although not necessary a feasible one with approximate objective). We now show how one can use more structure about the problem at hand to improve upon the vanilla application of the MWU technique. Specifically, we focus on the **bipartite matching** problem for this lecture.

1.1 Recap

Recall that the LP we would like to solve is the following:

$$\begin{aligned} \max_{x \in \mathbb{R}^E} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \sum_{e \ni v} x_e \leqslant 1 \quad \forall v \in V \\ & x_e \geqslant 0 \quad \forall e \in E; \end{aligned} \tag{1}$$

we know that this LP has an integral optimal solution (given the bipartiteness of the graph; see Lecture 8).

Based on this LP, and for every weight function w that assigns $w_v > 0$ to each vertex $v \in V$, we define the following **oracle LP** (here $W := \sum_{v \in V} w_v$):

$$\begin{aligned} \max_{x \in \mathbb{R}^E} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \sum_{v \in V} w_v \sum_{e \ni v} x_e \leqslant W \end{aligned}$$

$$x_e \geqslant 0 \quad \forall e \in E. \tag{2}$$

Notice that this LP is simpler than the original LP in that instead of the n "hard" constraints of the original LP—one for each vertex of the graph—we now only have a single constraint for their convex combination.

As we saw, the power of the MWU technique was that as long as we could solve the oracle LP efficiently, we could also approximately solve the original LP, as captured by the following algorithm.

Algorithm 1. A MWU algorithm for the Matching LP.

1. For every vertex $v \in V$, let $w_v^{(1)} = 1$.
2. For $t = 1$ to T iterations:
   (a) Let $x^{(t)}$ be any feasible solution to the oracle LP with weights $w^{(t)}$ according to LP (2).
   (b) For every vertex $v \in V$, update:
$$w_v^{(t+1)} = \left(1 + \eta \cdot \sum_{e \ni v} x_e^{(t)}\right) \cdot w_v^{(t)}.$$
3. Return the final solution
$$\bar{x} := \frac{1}{T} \cdot \sum_{t=1}^{T} x^{(t)}.$$

The following lemma, proved in the previous lecture, captures the main property of this algorithm.

Lemma 1. *For any $\varepsilon \in (0, 1/2)$, if we run Algorithm 1 for $T > \frac{8\rho \cdot \ln n}{\varepsilon^2}$ iterations for*

$$\rho \geqslant \max_{v \in V} \max_{t \geqslant 1} \sum_{e \ni v} x_e^{(t)},$$

with parameter $\eta = \frac{\varepsilon}{2\rho}$, then the output solution $\bar{x}$ satisfies

$$\sum_{e \in v} \bar{x}_e \leqslant 1 + \varepsilon \qquad \text{for all } v \in V.$$

The way we used Algorithm 1 and Lemma 1 in the previous lecture was the following: in each iteration, we find the edge $e = (u, v)$ with minimum weight $w_e^{(t)} := w_u^{(t)} + w_v^{(t)}$; then, we let $x_e^{(t)}$ to be $\min(n, W/w_e)$ and $x_f^{(t)} = 0$ for all $e \neq f$. As we argued, this implies that for all $t \in [T]$, $\sum_{e \in E} x_e^{(t)} \geqslant \mathsf{OPT}$, where OPT is the optimum solution of LP (1). Thus, we also have $\sum_{e \in E} \bar{x}_e \geqslant \mathsf{OPT}$ (since $\bar{x}$ is the average of $x^{(t)}$'s). Moreover, by Lemma 1, for $\rho = n$, we have that Algorithm 1 finishes in $O(n \ln n/\varepsilon^2)$ iterations and finds a solution $\bar{x}$ that satisfies every constraint up to a $(1+\varepsilon)$ factor; finally, rescaling $\bar{x}$ with a $(1+\varepsilon)$ factor, leads to a fractional matching of size at least $\mathsf{OPT}/(1+\varepsilon)$. Each iteration of the algorithm can be implemented in $O(n)$ time also, leading to an overall $O(n^2 \log n/\varepsilon^2)$ time for the entire algorithm.

1.2 A Strategy for Improved Algorithms

As we saw, *"all"* we need to be able to use MWU *efficiently* is the following:

- We should be able to solve the oracle LP (2) to achieve a solution which is (nearly) as good as the *original* LP (1) (but not necessarily optimally for LP (2) itself). This ensures our output will also be (nearly) optimal;

- We would like to reduce the **width parameter** ρ (defined in Lemma 1) to be as small as possible – this ensures the number of iterations will be as small as possible.

Let us now see how we can reduce the width for our particular bipartite matching LP.

As we proved earlier in the course (in Lecture 8), the integrality gap of LP (1) on bipartite graphs is one. Thus, we can assume that there is always a matching M_{OPT} of size OPT in the in the input graph G. On the other hand, recall that Oracle LP (2), when stated with the weights of edges as sum of weight of its vertices, is the following:

$$\begin{aligned} \max_{x \in \mathbb{R}^E} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \sum_{e \in E} x_e \cdot w_e \leqslant W \\ & x_e \geqslant 0 \quad \forall e \in E. \end{aligned} \tag{3}$$

Thus, we are saying that there is a matching M_{OPT} in the graph G with the total weight at most W (no matter what weights we are choosing for the vertices and thus, in turn, the weight of edges).

Now, consider our previous approach of picking the edge e^* with minimum weight and setting $x_{e^*} = \mathsf{OPT}$ and $x_e = 0$ for all $e \neq e^*$ (we actually put a different value because we do not know OPT, but you can see that *had* we known OPT, using this choice would have worked as well). We can think of this strategy as follows: while M_{OPT} has OPT edges with weight at most W, we can find a single edge with weight at most W/OPT – thus, on this single edge, we are *competing* quite favorably with the optimum in terms of "bang for the buck": how much we contribute to objective value versus the main constraint of the Oracle LP. The problem with this approach was that we needed to put a very large value on x_{e^*} and thus get a large width ρ, which in turn led to a large number of iterations in the algorithm.

But, now suppose could pick k edges with total weight W/k for some $k > 1$. Can we again compete favorably with the optimum solution? *Yes*; do we get a lower width? *Not necessarily* because the width is determined by the largest x-value we put on a single *vertex* and not an edge. Thus, just picking k edges is not enough. But what if these k edges form a matching? Then, indeed we can reduce the width to OPT/k also by putting an x-value of OPT/k over each of these edges.

We use the following (slightly more general) lemma to formalize this.

Lemma 2. *Let $\delta \in (0, 1/2)$ be any parameter and M be any matching in G with*

$$\frac{w(M)}{|M|} \leqslant (1+\delta) \cdot \frac{W}{\mathsf{OPT}};$$

then, the solution $x \in \mathbb{R}^E$ with $x_e = \mathsf{OPT}/(1+\delta)|M|$ for $e \in M$ and 0 outside M is feasible for the Oracle LP (2) with objective value $\mathsf{OPT}/(1+\delta)$ and width (in this iteration), at most $\mathsf{OPT}/|M|$.

Proof. For the feasibility, we have,

$$\sum_{e \in E} x_e \cdot w_e = \sum_{e \in M} \frac{\mathsf{OPT}}{(1+\delta) \cdot |M|} \cdot w_e = \frac{\mathsf{OPT}}{(1+\delta)} \cdot \frac{w(M)}{|M|} \leqslant \frac{\mathsf{OPT}}{(1+\delta)} \cdot (1+\delta) \cdot \frac{W}{\mathsf{OPT}} = W,$$

and thus the solution is feasible. For the objective value,

$$\sum_{e \in E} x_e = \sum_{e \in M} \frac{\mathsf{OPT}}{(1+\delta) \cdot |M|} = |M| \cdot \frac{\mathsf{OPT}}{(1+\delta) \cdot |M|} = \frac{\mathsf{OPT}}{(1+\delta)}.$$

And finally, since M is a matching, each vertex $v \in V$ is incident on at most one edge of M and thus

$$\sum_{e \ni v} x_e \leqslant \frac{\mathsf{OPT}}{(1+\delta) \cdot |M|} \leqslant \frac{\mathsf{OPT}}{|M|},$$

concluding the proof. □

Again, the interpretation of Lemma 2 is that as long as we can find a matching such that on average, it "spends" less weight per each than M_{OPT} does (i.e., bang per buck competes with M_{OPT}), we can turn it into a feasible solution for the Oracle LP (2) with a competitive objective (we would like to set δ to be at most $\Theta(\varepsilon)$ so that the final solution is still a $(1-O(\varepsilon))$ approximation after scaling $\bar{x}$ by a $(1+\varepsilon)$ factor). Moreover, the width of this approach will depend on how large we can make $|M|$ (the larger the better).

We now show two different strategies for obtaining better algorithms (with smaller width) using Lemma 2.

1.3 Reducing the Width: Approach One

Our first approach finds a matching M of size at least $\varepsilon/4 \cdot \mathsf{OPT}$ which satisfies the premise of Lemma 2 for the parameter $\delta = \varepsilon$. A key observation toward this algorithm is the following.

Claim 3. *There are at least $\varepsilon/2 \cdot \mathsf{OPT}$ edges e in M_{OPT} such that $w_e \leqslant (1+\varepsilon) \cdot W/\mathsf{OPT}$.*

Proof. Suppose not; then, the weight of the remaining edges is at least

$$(1-\varepsilon/2) \cdot \mathsf{OPT} \cdot (1+\varepsilon) \cdot \frac{W}{\mathsf{OPT}} > W,$$

contradicting the fact that $w(M_{\mathsf{OPT}}) \leqslant W$ (as argued earlier). □

Using this claim, we can do the following[1] in Line (2a) of Algorithm 1: we consider all edges of G whose weight currently is at most $(1+\varepsilon) \cdot W/\mathsf{OPT}$; then, we run the greedy algorithm on this subgraph of G to find a matching M. As we have proven earlier in the course, size of M is at least half of the maximum matching in the subgraph, which itself, is of size at least $\varepsilon/2 \cdot \mathsf{OPT}$ by Claim 3.

To analyze the algorithm, we have that the matching M returned above always satisfy the requirement of Lemma 2 for $\delta = \varepsilon$ and thus, using that lemma, we can return a feasible solution $x^{(t)}$ for the Oracle LP (2) in each iteration $t \in [T]$ of Algorithm 1 such that its objective value is at least $\mathsf{OPT}/(1+\varepsilon)$ and the resulting width will be

$$\rho \leqslant \frac{\mathsf{OPT}}{|M|} \leqslant \frac{\mathsf{OPT}}{\varepsilon/4 \cdot \mathsf{OPT}} = \frac{4}{\varepsilon};$$

thus, we have reduced our width from n all the way down to $O(1/\varepsilon)$ using this new algorithm for Line (2a) of Algorithm 1. If we use this approach instead in Lemma 1, we will get an algorithm that converges in only $O(\log(n)/\varepsilon^3)$ iterations, and each iteration takes $O(m)$ time for finding the desired subgraph and running the greedy matching algorithm over that. Finally, the solution $\bar{x}$ we find satisfies

$$\sum_{e \in E} \bar{x}_e \geqslant \frac{\mathsf{OPT}}{(1+\varepsilon)} \quad \text{and} \quad \sum_{e \ni v} \bar{x}_e \leqslant (1+\varepsilon) \quad \text{for all } v \in V.$$

Hence, by rounding down $\bar{x}$ by a factor of $(1+\varepsilon)$, we obtain a feasible solution which is a $(1-O(\varepsilon))$ approximation (to obtain a $(1-\varepsilon)$-approximation exactly, simply run the algorithm with a smaller value of ε by a constant factor).

All in all, this gives us an algorithm for finding a $(1-\varepsilon)$-approximate fractional matching[2] in

$$O(\frac{\log(n)}{\varepsilon^3})$$

iterations of the MWU algorithm and

$$O(\frac{m \log n}{\varepsilon^3})$$

time in total.

We now switch to an even more improved strategy for this problem.

[1] This step requires assuming that we actually know the value of OPT. One way of handling this is to binary search for OPT as well; we do not cover this part in more detail as in the next part, we are going to prove an improved way of solving the oracle LP which does not require this assumption anyway.

[2] We can also round this fractional matching into an integral one given the input graph is bipartite, but we skip that step.

1.4 Reducing the Width: Approach Two

We now design an algorithm that solves the Oracle LP (2) with a width of only 2 (!), still in $O(m)$ time.

Algorithm 2. A Better Oracle for the Bipartite Matching LP.

1. Sort the edges in the *increasing* order of their weights.
2. Let $M = \emptyset$. While $w(M) \leqslant W/2$:
   - Pick the lowest weight available edge e in M if both its endpoints are unmatched, and otherwise skip this edge.
3. Return M as the final matching.

Lemma 4. *Size of M in Algorithm 2 is at least $\mathsf{OPT}/2$ for any choices of weights on the edges.*

Proof. Let $e_1, e_2, \ldots, e_{\mathsf{OPT}/2}$ be the first $\mathsf{OPT}/2$ edges of M in the order added to M and $o_1, o_2, \ldots, o_{\mathsf{OPT}}$ be the edges of M_{OPT} sorted in the increasing order of their weight. We claim inductively that for every $i \in [\mathsf{OPT}/2]$,

$$\underbrace{\sum_{j=1}^{i} w(e_j)}_{w(e_{\leqslant i})} \leqslant \frac{1}{2} \cdot \underbrace{\sum_{j=1}^{2i} w(o_j)}_{w(o_{\leqslant 2i})};$$

in words, the total weight of the first i edges in M, denoted by $w(e_{\leqslant i})$ is at most half the total weight of the first $2i$ edges in M_{OPT}, denoted by $w(o_{\leqslant 2i})$.

The base case for $i = 1$ holds because $w(e_1)$ is smaller than all other edges in the graph and thus in particular is half of the $w(o_1) + w(o_2)$.

For induction case, suppose we are adding the edge e_i in this iteration. Since $e_1, \ldots, e_{i-1}$ is a matching, each of these edges can be incident on at most two edges in M_{OPT}. Thus, by the pigeonhole principle, among the edges $o_1, o_2, \ldots, o_{2i-1}, o_{2i}$, there are at least two edges that are not incident on any of these edges. Hence, when picking e_i, we could have alternatively picked any of these two edges, which means that weight of e_i is at most equal to the weight of each of these two edges. In particular, we definitely have,

$$w(e_i) \leqslant \frac{1}{2} \cdot \left(w(o_{2i-1}) + w(o_{2i})\right).$$

Moreover, by induction, we have that

$$w(e_{\leqslant i-1}) \leqslant \frac{1}{2} \cdot w(o_{\leqslant 2i-2}).$$

Combining these two implies the induction hypothesis.

The lemma now follows from this because we have the weight of the first $\mathsf{OPT}/2$ edges of M is at most $W/2$ and thus the algorithm picks at least $\mathsf{OPT}/2$ edges before terminating. □

We can now use Algorithm 2 in Line (2a) of Algorithm 1. The matching M it returns satisfies the premise of Lemma 2 for $\delta = 0$ (since it has at least $\mathsf{OPT}/2$ edges with weight at most $W/2$ and thus average weight of an edge is at most W/OPT), and thus we get a feasible Oracle LP (2) solution with objective value at least OPT and width $\mathsf{OPT}/|M| \leqslant \mathsf{OPT}/(\mathsf{OPT}/2) \leqslant 2$.

This way, by Lemma 1, we obtain a $(1+\varepsilon)$-feasible solution $\bar{x}$ to LP (1) with objective value at least OPT in only

$$O(\frac{\ln n}{\varepsilon^2})$$

iterations (thus $\Theta(1/\varepsilon)$ fewer iterations than the previous algorithm). Each iteration also takes $O(m \log m)$ time if we sort the edges directly using any sorting algorithm. We can in fact implement each iteration in $O(m)$ time also because the weight of an edge is simply determined by the number of times each of its endpoints is matched so far, which is an integer; so, we can simply use Radix sort (or any other fast integer sorting algorithm) to sort their weight in $O(m)$ time. All in all, this gives us an algorithm with

$$O\left(m \cdot \frac{\ln n}{\varepsilon^2}\right)$$

runtime for returning a $(1-\varepsilon)$-approximate fractional matching.

1.5 An Even (Slightly) Better Algorithm for Additive Approximation

Finally, to show case an important feature (or rather "weakness" in the way we analyzed Algorithm 1), let us show that if our goal is to settle for a slightly weaker approximation guarantee of outputting a matching of size $\mathsf{OPT} - \varepsilon \cdot n$, instead of $(1-\varepsilon) \cdot \mathsf{OPT}$, we can reduce the number of iterations to be independent of n, i.e., remove the $\ln n$-term[3].

So, why did we need the $\ln n$-term in the analysis of Algorithm 1 and in particular Lemma 1? This was essentially because we were comparing the weight of a single constraint v which were violated, against the entire potential function (a combination of n different weights). In other words, we should have run the algorithm long enough such that even one violated constraint could overweight all n constraints. But, what if we set our goal to make sure there are $< \varepsilon \cdot n$ violated constraints, namely,

$$\sum_{e \ni v} \bar{x}_e > (1+\varepsilon),$$

for $< \varepsilon \cdot n$ vertices $v \in V$. Why is this good enough for $\Theta(\varepsilon \cdot n)$ additive approximation? Notice that in Algorithm 2, we never violated a constraint with more than 2 so we always have

$$\sum_{e \ni v} \bar{x}_e \leqslant 2,$$

(because $\rho = 2$). Thus, after we have $< \varepsilon \cdot n$ violated constraints, we can simply remove *all* values of x incident on their edges so they become feasible. But this means that we reduce the value of $\bar{x}$ with at most $2\varepsilon \cdot n$ in total. Thus, after this modification, we obtain a solution which satisfies all constraints up to a $(1+\varepsilon)$ factor and its value is at least $\mathsf{OPT} - 2\varepsilon \cdot n$. This implies that we obtain a solution which is a $\mathsf{OPT} - 3\varepsilon \cdot n$ after re-scaling $\bar{x}$ to become $\bar{x}/(1+\varepsilon)$. Finally, to obtain an additive εn approximation (instead of $3\varepsilon n$, we can simply start this algorithm with ε replaced by $\varepsilon/3$).

The final question is how many iterations we need to reduce the number of violated constraints to $< \varepsilon \cdot n$? This is handled by the following lemma.

Lemma 5. *For any $\varepsilon \in (0, 1/2)$, if we run Algorithm 1 for $T > \frac{8\rho \cdot \ln(1/\varepsilon)}{\varepsilon^2}$ iterations for*

$$\rho \geqslant \max_{v \in V} \max_{t \geqslant 1} \sum_{e \ni v} x_e^{(t)},$$

with parameter $\eta = \frac{\varepsilon}{2\rho}$, then the output solution $\bar{x}$ only violates the constraints of Eq (2) for at most $\varepsilon \cdot n$ vertices.

Proof. In the last lecture (Claim 5), if in $\bar{x}$, a vertex v violates constraints of LP (1) by more than $(1+\varepsilon)$ at the end of Algorithm 1, then,

$$w_v^{(T+1)} \geqslant \exp\left(\eta \cdot (1 + \frac{\varepsilon}{4}) \cdot T\right).$$

[3]We do not cover this at this point in the course, but this can also be turned into a $(1-\varepsilon)$-approximation (as in, a multiplicative guarantee and not only additive) using some more matching-related ideas.

Let S be the set of all violating vertices and suppose towards a contradiction that $|S| \geqslant \varepsilon \cdot n$. We thus have,

$$\sum_{v \in S} w_v^{(T+1)} \geqslant \varepsilon \cdot n \cdot \exp\left(\eta \cdot (1 + \frac{\varepsilon}{4}) \cdot T\right) = \exp\left(\eta \cdot (1 + \frac{\varepsilon}{4}) \cdot T + \ln \varepsilon + \ln n\right).$$

On the other hand, we also proved in the last lecture (Claim 4) that the total weight at the end of the algorithm is

$$W^{(T+1)} \leqslant \exp\left(\eta \cdot T + \ln n\right).$$

Given that the LHS of the first equation is still upper bounded by the LHS of the next equation, we have

$$\eta \cdot (1 + \frac{\varepsilon}{4}) \cdot T + \ln \varepsilon + \ln n \leqslant \eta \cdot T + \ln n,$$

which implies that

$$T \leqslant \frac{4 \cdot \ln(1/\varepsilon)}{\eta \cdot \varepsilon}.$$

Replacing η with its value finalizes the proof. □

This implies that after running the algorithm for only $O(\ln(1/\varepsilon)/\varepsilon^2)$ iterations, we obtain the desired solution. In general, the ideas in this part can be very helpful for reducing the number of iterations to something independent of n. Let us mention that $\ln(1/\varepsilon)$ factor of Lemma 5 is actually not necessary for obtaining a solution of value $\mathsf{OPT} - O(\varepsilon) \cdot n$ at the end; we leave this as an exercise for the reader.

We conclude our lecture by mentioning that the approach presented in this lecture was motivated by the work of [AG11] although both our general formulation of the problem as well as our improved oracle algorithm with constant width are different from that work.

References

[AG11] Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, volume 6756 of *Lecture Notes in Computer Science*, pages 526–538. Springer, 2011. 7