Chapter 15

Multiplicative Weight Update III: Width Reduction Techniques

In the last chapter, we saw how to use the Multiplicative Weight Update (MWU) technique to approximate the fractional matching LP in a rather generic way. We now show how one can use more structure about the specific problem at hand to improve upon the vanilla application of the MWU technique.

15.1 A Faster MWU Algorithm for Bipartite Matching

While in the previous chapter we focused on fractional matching in arbitrary graphs, we switch to the **bipartite matching** problem for this chapter. Recall the bipartite matching LP and corresponding oracle LP (for any choices of weights $w_v \geqslant 0$ for $v \in V$ and $W := \sum_v w_v$):

Bipartite Matching LP (LP):

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

Oracle LP (OLP):

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

As before, we let opt_{LP} be the optimal value of this LP and opt_{OLP} be the optimal value of the oracle LP. Moreover, by Proposition 6.5, the integrality gap of this LP is one on bipartite graphs and thus we can assume there is a matching M_{opt} of size opt_{LP} in the graph G.

By Lemma 14.8, to run Algorithm 14.4 on this LP, we need to repeatedly solve the OLP for

$$T = \frac{8\rho \cdot \ln n}{\varepsilon^2}$$

iterations to get feasible solutions $x^{(1)}, x^{(2)}, \ldots, x^{(T)}$ (to OLP) for

$$\rho \geqslant \max_{v \in V} \max_{1 \leqslant t \leqslant T} \sum_{e \ni v} x_e^{(t)}. \tag{15.1}$$

We then set $\bar{x} := \frac{1}{T} \cdot \sum_{t=1}^{T} x^{(t)}$ and have

$$\begin{aligned} &\sum_{e \in E} \bar{x}_e \geqslant \min_{1 \leqslant t \leqslant T} \sum_{e \in E} x_e^{(t)}, \\ &\sum_{e \ni v} \bar{x}_e \leqslant (1+\varepsilon) \qquad \text{for all } v \in V. \end{aligned} \tag{15.2}$$

In the previous chapter, we used this strategy as follows: in each iteration $t \in [T]$, we find the edge $e = (u, v) \in E$ with minimum $w_e^{(t)} := w_u^{(t)} + w_v^{(t)}$; then, we set $x_e^{(t)}$ to be $\min\left(n, W/w_e^{(t)}\right)$ and set $x_f^{(t)} = 0$ for all $f \neq e$. This implies that for all $t \in [T]$, $\sum_{e \in E} x_e^{(t)} \geqslant \text{opt}_{\text{LP}}$. Thus, we also have $\sum_{e \in E} \bar{x}_e \geqslant \text{opt}_{\text{LP}}$ by Eq (15.2). Moreover, the parameter $\rho \leqslant n$, and so the algorithm 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 $\text{opt}_{\text{LP}}/(1+\varepsilon)$. Each iteration of the algorithm can be implemented in $O(n \log n)$ time also, leading to an overall $O(n^2 \log^2(n)/\varepsilon^2)$ time for the entire algorithm.

In this chapter, we see how to reduce the number of iterations and speedup this algorithm.

15.1.1 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 to achieve a solution which is (nearly) as good as the *original* LP (but is not necessarily optimal for the oracle LP). This ensures our output will also be (nearly) optimal;
- We would like to reduce the **width parameter** ρ (defined in Eq (15.1)) 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.

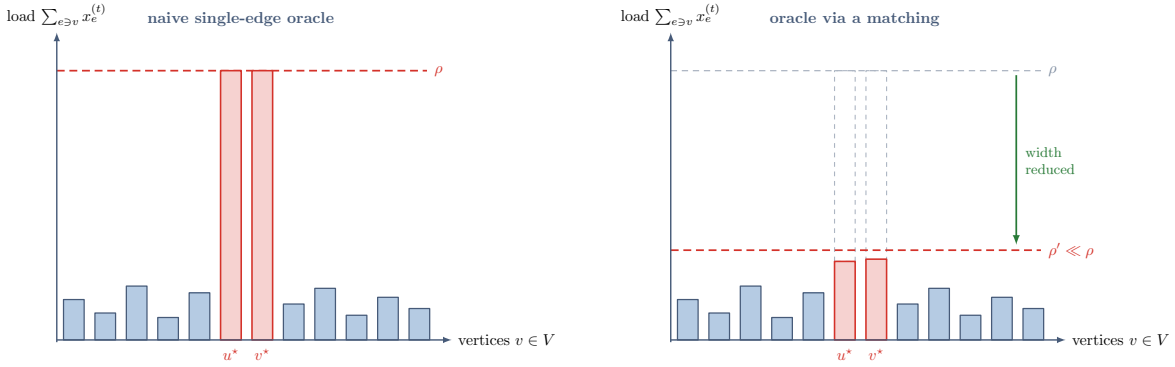
Note that we can state the Oracle LP alternatively as

$$\begin{aligned} \max_{x \in \mathbb{R}^E} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \sum_{e \in E} x_e \cdot w_e \leqslant W \\ & x_e \geqslant 0 \quad \forall e \in E. \end{aligned} \tag{15.3}$$

Thus, 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 $x_{e^*} = \text{opt}_{\text{LP}}$ and $x_e = 0$ for all $e \neq e^*$ (we actually put a different value because we do not know opt_{LP}, but you can see that *had* we known opt_{LP}, using this choice would have worked as well). We can think of this strategy as follows: while M_{opt} has opt_{LP} edges with weight at most W, we can find a single edge with weight at most W/opt_{LP} – 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 (see Figure 15.1a).

But, now suppose we could pick k edges with total weight $k \cdot W/\text{opt}_{\text{LP}}$ for some $k > 1$. Can we again compete favorably with the optimum solution by setting $x_{e^*} = \text{opt}_{\text{LP}}/k$ for picked edges? *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_{LP}/k also by putting an x-value of opt_{LP}/k over each of these edges (see Figure 15.1b).



(a) The naive single-edge oracle puts all the LP mass on one cheapest edge $e^\star = (u^\star, v^\star)$, so its two endpoints' load spikes and the width ρ (dashed line) becomes as large as $\approx n$.

(b) Spreading the same mass over a matching M (an x-value of $\text{opt}_{\text{LP}}/|M|$ per edge) caps every vertex's load considerably lower.

Figure 15.1: Reducing the width of the bipartite-matching oracle LP. Since the number of iterations $T = 8\rho \ln n/\varepsilon^2$ scales with the width, collapsing the spikes from $\rho \approx n$ to smaller ρ significantly reduces the number of iterations.

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

Lemma 15.1. *Let $\delta \in (0, 1/2)$ be any parameter and M be any matching in G with*

$$\frac{w(M)}{|M|} \leqslant (1+\delta) \cdot \frac{W}{\text{opt}_{\text{LP}}};$$

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

Proof. For the feasibility, we have,

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

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

$$\sum_{e \in E} x_e = \sum_{e \in M} \frac{\text{opt}_{\text{LP}}}{(1+\delta) \cdot |M|} = |M| \cdot \frac{\text{opt}_{\text{LP}}}{(1+\delta) \cdot |M|} = \frac{\text{opt}_{\text{LP}}}{(1+\delta)}.$$

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

$$\sum_{e \ni v} x_e \leqslant \frac{\text{opt}_{\text{LP}}}{(1+\delta) \cdot |M|} \leqslant \frac{\text{opt}_{\text{LP}}}{|M|},$$

concluding the proof. □

Again, the interpretation of Lemma 15.1 is that as long as we can find a matching such that on average "spends" less weight per edge than M_{opt} does (i.e., bang for the buck competes with M_{opt}), we can turn it into a feasible solution for the Oracle LP with a competitive objective (we would like to set δ to be at most $\Theta(\varepsilon)$ so that the final solution is still a $(1-\Theta(\varepsilon))$-approximation after scaling $\bar{x}$ by a $(1+\Theta(\varepsilon))$ 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 15.1.

15.1.2 Reducing the Width: Approach One

Our first approach finds a matching M of size at least $\varepsilon/4 \cdot \text{opt}_{\text{LP}}$ which satisfies the premise of Lemma 15.1 for the parameter $\delta = \varepsilon$. A key observation toward this algorithm is the following.

Claim 15.2. *There are at least $\varepsilon/2 \cdot \text{opt}_{\text{LP}}$ edges e in M_{opt} such that $w_e \leqslant (1+\varepsilon) \cdot W/\text{opt}_{\text{LP}}$.*

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

$$(1-\varepsilon/2) \cdot \text{opt}_{\text{LP}} \cdot (1+\varepsilon) \cdot \frac{W}{\text{opt}_{\text{LP}}} > W,$$

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

Using this claim, we can do the following when solving the Oracle LP in Algorithm 14.4[1]: we consider all edges of G whose weight currently is at most $(1+\varepsilon) \cdot W/\text{opt}_{\text{LP}}$; then, we run the greedy algorithm on this subgraph of G to find a matching M. As we have proven earlier in the book (Lemma 9.2), the size of M is at least half of the maximum matching in the subgraph, which itself is of size at least $\varepsilon/2 \cdot \text{opt}_{\text{LP}}$ by Claim 15.2.

To analyze the algorithm, we have that the matching M returned above always satisfies the requirement of Lemma 15.1 for $\delta = \varepsilon$ and thus, using that lemma, we can return a feasible solution $x^{(t)}$ for the Oracle LP in each iteration $t \in [T]$ of Algorithm 14.4 such that its objective value is at least $\text{opt}_{\text{LP}}/(1+\varepsilon)$ and the resulting width will be

$$\rho \leqslant \frac{\text{opt}_{\text{LP}}}{|M|} \leqslant \frac{\text{opt}_{\text{LP}}}{\varepsilon/4 \cdot \text{opt}_{\text{LP}}} = \frac{4}{\varepsilon};$$

thus, we have reduced our width from n all the way down to $O(1/\varepsilon)$ using this new subroutine for solving the Oracle LP. If we use this approach in our strategy, we will get an algorithm that converges in only $O(\log(n)/\varepsilon^3)$ iterations, each taking $O(m)$ time for finding the desired subgraph and running the greedy matching algorithm over it. Finally, the solution $\bar{x}$ satisfies

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

Hence, by rounding down $\bar{x}$ by a factor of $(1+\varepsilon)$, we obtain a feasible solution which is a $(1-3\varepsilon)$-approximation (to obtain a $(1-\varepsilon)$-approximation exactly, run the algorithm with a smaller value of ε by a constant factor).

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

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

[1] This step requires assuming that we know the value of opt_{LP}. One way of handling this is to binary search for opt_{LP}; we do not cover this idea in more detail and leave it as an exercise because in the next subsection, we are going to prove an improved way of solving the Oracle LP which does not require this assumption.

[2] We can round this fractional matching into an integral one given the input graph is bipartite (Proposition 6.5).

iterations of the MWU algorithm (Algorithm 14.4) and total runtime of

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

15.1.3 Reducing the Width: Approach Two

We now switch to an even more improved strategy for this problem. We design an algorithm that solves the Oracle LP with a width of only 2, still in $O(m)$ time.

Algorithm 15.3.

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

Lemma 15.4. *The size of M in Algorithm 15.3 is at least $\text{opt}_{\text{LP}}/2$ for any choices of weights.*

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

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

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 less than or equal to the weight any other edge in the graph and thus in particular is half of $w(o_1) + w(o_2)$.

For induction step, 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 either 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 $\text{opt}_{\text{LP}}/2$ edges of M is at most $W/2$ and thus the algorithm picks at least $\text{opt}_{\text{LP}}/2$ edges before terminating. □

We can use Algorithm 15.3 to solve the Oracle LP in the MWU algorithm (Algorithm 14.4). The matching M it returns satisfies the premise of Lemma 15.1 for $\delta = 0$ (since it has at least $\text{opt}_{\text{LP}}/2$ edges with weight at most $W/2$ and thus average weight of an edge is at most W/opt_{LP}), and thus we get a feasible Oracle LP solution with objective value at least opt_{LP} and width $\text{opt}_{\text{LP}}/|M| \leqslant \text{opt}_{\text{LP}}/(\text{opt}_{\text{LP}}/2) \leqslant 2$.

This way, by Lemma 14.8, we obtain a $(1+\varepsilon)$-feasible solution $\bar{x}$ to the bipartite matching LP with objective value at least opt_{LP} in only

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

iterations (thus a $\Theta(1/\varepsilon)$ factor 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 bipartite matching.

15.1.4 A (Slightly) Better Algorithm for Additive Approximation

Finally, to showcase an important feature (or rather weakness in the way we analyzed Algorithm 14.4 in Chapter 14), let us show that if our goal is to settle for a slightly weaker approximation guarantee of outputting a matching of size at least $\text{opt}_{\text{LP}} - \varepsilon \cdot n$, instead of $(1-\varepsilon) \cdot \text{opt}_{\text{LP}}$, we can reduce the number of iterations to be independent of n, i.e., remove the $\ln n$-term[3].

To begin with, why did we need the $\ln n$-term in the analysis of Algorithm 14.4 and in particular Lemma 14.8? This was essentially because we were comparing the weight of a single constraint v which was 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 at most εn violated constraints, namely,

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

for at most εn vertices $v \in V$. Why is this good enough for $\Theta(\varepsilon n)$ additive approximation? In Algorithm 15.3, we never violated a constraint by more than a factor of 2 so we have

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

for all vertices $v \in V$. Thus, after we have less than εn 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 $\bar{x}$ by at most $2\varepsilon n$ 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 $\text{opt}_{\text{LP}} - 2\varepsilon n$. This implies that we obtain a solution which has a value of at least $\text{opt}_{\text{LP}} - 3\varepsilon n$ after re-scaling

[3] 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. We postpone this to the exercises.

$\bar{x}$ 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 below $\varepsilon \cdot n$? This is handled by the following lemma.

Lemma 15.5. *For any $\varepsilon \in (0, 1/2)$, if we run Algorithm 14.4 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 the matching LP ((14.1)) for at most εn vertices.

Proof. By Claim 14.7, if in $\bar{x}$ a vertex v violates constraint of the matching LP at v by more than $(1+\varepsilon)$ at the end of Algorithm 14.4, then,

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

Let S be the set of all violating vertices and suppose towards a contradiction that $|S| \geqslant \varepsilon n$. 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 Claim 14.6 that the total weight at the end 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 second equation (the weight over S is at most the total weight), 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 15.5 is actually not necessary for obtaining a solution of value $\text{opt}_{\text{LP}} - O(\varepsilon) \cdot n$ at the end; we leave this as an exercise for the reader.

Remark. We conclude our chapter by mentioning that the MWU approach for solving bipartite matching presented here was motivated by the work of [AG11] although both our general formulation of the problem and our improved oracle algorithm with constant width are different from that work.

Exercises

Exercise 15.1. Show how to run a binary search for the value of opt_{LP} to complete the algorithm in Section 15.1.2.

Exercise 15.2 (♦). Recall our earlier goal of turning the additive guarantee of the algorithm in Section 15.1.4 to a more standard multiplicative $(1-\varepsilon)$-approximation. This can be achieved by a preprocessing step wherein we reduce the number of vertices to $O_\varepsilon(\text{opt})$ while keeping the matching size to remain $(1-\varepsilon)\cdot\text{opt}$ still. Show how this preprocessing can be implemented.

Exercise 15.3 (♦). Show how to remove the $\log(1/\varepsilon)$ term of the algorithm in Section 15.1.4, using a more careful analysis of the same algorithm.