Multiplicative Auction Algorithm for Approximate Maximum Weight Bipartite Matching

Da Wei Zheng
University of Illinois Urbana-Champaign
Urbana, IL, USA

Monika Henzinger
University of Vienna
Vienna, Austria

Abstract

We present an *auction algorithm* using multiplicative instead of constant weight updates to compute a $(1-\varepsilon)$-approximate maximum weight matching (MWM) in a bipartite graph with n vertices and m edges in time $O(m\varepsilon^{-1}\log(\varepsilon^{-1}))$, matching the running time of the linear-time approximation algorithm of Duan and Pettie [JACM '14]. Our algorithm is very simple and it can be extended to give a dynamic data structure that maintains a $(1-\varepsilon)$-approximate maximum weight matching under (1) one-sided vertex deletions (with incident edges) and (2) one-sided vertex insertions (with incident edges sorted by weight) to the other side. The total time time used is $O(m\varepsilon^{-1}\log(\varepsilon^{-1}))$, where m is the sum of the number of initially existing and inserted edges.

1 Introduction

Let $G = (U \cup V, E)$ be an edge-weighted bipartite graph with $n = |U \cup V|$ vertices and $m = |E|$ edges where each edge $uv \in E$ with $u \in U$ and $v \in V$ has a non-negative weight $w(uv)$.

The *maximum weight matching* (MWM) problem asks for a matching $M \subseteq E$ that attains the largest possible weight $w(M) = \sum_{uv\in M} w(uv)$. This paper will focus on approximate solutions to the MWM problem. More specifically, if we let M^* denote a maximum weight matching of G, our goal is to find a matching M such that $w(M) \geq (1-\varepsilon)w(M^*)$ for any small constant $\varepsilon > 0$.

Matchings are a very well studied problem in combinatorial optimization. Kuhn [13] in 1955 published a paper that started algorithmic work in matchings, and presented what he called the "Hungarian algorithm" which he attributed the work to Kőnig and Egerváry. Munkres [15] showed that this algorithm runs in $O(n^4)$ time. The running time for computing the exact MWM has been improved many times since then. Recently this year, Chen et al. [6] showed that it was possible to solve the more general problem of max flow in $O(m^{1+o(1)})$ time.

For $(1-\varepsilon)$-approximation algorithms for MWM in bipartite graphs, Gabow and Tarjan in 1989 showed an $O(m\sqrt{n}\log(n/\varepsilon))$ algorithm. Since then there were a number of results for different running times and different approximation ratios. The current best approximate algorithm is by Duan and Pettie [8] which computes a $(1-\varepsilon)$-approximate maximum weight matching in $O(m\varepsilon^{-1}\log(\varepsilon^{-1}))$ time with a scaling algorithm. We defer to their work for a more thorough survey of the history on the MWM problem.

We show in our work that the auction algorithm for matchings using multiplicative weights can give a $(1-\varepsilon)$-approximate maximum weight matching with a running time of $O(m\varepsilon^{-1}\log(\varepsilon^{-1}))$ for bipartite graphs. This matches the best known running time of Duan and Pettie [8]. However, in comparison to their rather involved algorithm, our algorithm is simple and only uses elementary data structures. Furthermore, we are able to use properties of the algorithm to support two dynamic operations, namely one where vertices are deleted from one side and one where vertices of the other side of the bipartite graph are inserted together with their incident edges.

1.1 Dynamic matching algorithms.

Dynamic weighted matching. There has been a large body of work on dynamic matching and many variants of the problem have been studied, e.g, the maximum, maximal, as well as α-approximate setting for a variety of values of α, both in the weighted as well as in the unweighted setting. See [10] for a survey of the current state of the art for the fully dynamic setting. We just mention here a few of the most relevant prior works. For any constant $\delta > 0$ there is a conditional lower bound based on the OMv conjecture that shows that any dynamic algorithm that returns the *exact* value of a maximum cardinality matching in a bipartite graph with polynomial preprocessing time cannot take time $O(m^{1-\delta})$ per query and $O(m^{1/2-\delta})$ per edge update operation [11]. For *general weighted* graphs Gupta and Peng [9] gave the first algorithm in the *fully dynamic* setting with edge insertions and deletions to maintain a $(1-\varepsilon)$-approximate matching in $O(\varepsilon^{-1}\sqrt{m}\log w_{max})$ time, where the edges fall into the range $[1, w_{max}]$.

Vertex updates. By vertex update we refer to updates that are vertex insertion (resp. deletion) that also inserts (resp. deletes) all edges incident to the vertex. There is no prior work on maintaining matchings in weighted graphs under vertex updates. However, vertex updates in the *unweighted bipartite* setting has been studied. Bosek et al. [4] gave an algorithm that maintains the $(1-\varepsilon)$-approximate matching when vertices of one side are deleted in $O(\varepsilon^{-1})$ amortized time per changed edge. The algorithm can be adjusted to the setting where vertices of one side are inserted in the same running time, but it cannot handle both vertex insertions and deletions. Le et al. [14] gave an algorithm for maintaining a *maximal* matching under vertex updates in constant amortized time per changed edge. They also presented an $e/(e-1) \approx 1.58$ approximate algorithm for maximum matchings in an unweighted graph when vertex updates are only allowed on one side of a bipartite graph.

We give the first algorithm to maintain a $(1-\varepsilon)$-approximate maximum weight matching where vertices can undergo vertex deletions on one side *and* vertex insertions on the other side in total time $O(m\varepsilon^{-1}\log(\varepsilon^{-1}))$, where m is the sum of the number of initially existing, inserted, and deleted edges. It assumes that the edges incident to an inserted vertex are given in sorted order by weight, otherwise, the running time increases by $O(\log n)$ per inserted edge.

1.2 Linear Program for MWM

The MWM problem can be expressed as the following *linear program* (LP) where the variable x_{uv} denotes whether the edge uv is in the matching. It is well known [17] that the below LP is integral, that is the optimal solution has all variables $x_{uv} \in \{0,1\}$.

$$\begin{aligned}
\max \quad & \sum_{uv \in E} w(uv) x_{uv} & \\
s.t. \quad & \sum_{v \in N(u)} x_{uv} \leq 1 & \forall u \in U \\
& \sum_{u \in N(v)} x_{uv} \leq 1 & \forall v \in V \\
& x_{uv} \geq 0 & \forall uv \in E
\end{aligned}$$

We can also consider the dual problem that aims to find dual weights y_u and y_v for every vertex $u \in U$ and $v \in V$ respectively.

$$\begin{aligned}
\min \quad & \sum_{u \in U} y_u + \sum_{v \in V} y_v & \\
s.t. \quad & y_u + y_v \geq w(uv) & \forall uv \in E \\
& y_u \geq 0 & \forall u \in U \\
& y_v \geq 0 & \forall v \in V
\end{aligned}$$

1.3 Multiplicative weight updates for packing LPs

Packing LPs are LPs of the form $\max\{c^T x \mid Ax \leq b\}$ for $c \in \mathbb{R}^n_{\geq 0}$, $b \in \mathbb{R}^m_{\geq 0}$ and $A \in \mathbb{R}^{n \times m}_{\geq 0}$. The LP for MWM is a classical example of a packing LP. The *multiplicative weight update method* (MWU) has been investigated extensively to provide faster algorithms for finding approximate solutions[1] to packing LPs [1, 5, 12, 16, 18, 19]. Typically the running times for solving these LPs have a dependence on ε of ε^{-2}, e.g. the algorithm of Koufogiannakis and Young [12] would obtain a running time of $O(m\varepsilon^{-2} \log n)$ when applied to the matching LP.

The fastest multiplicative weight update algorithm for solving packing LPs by Allen-Zhu and Orecchia [1] would obtain an $O(m\varepsilon^{-1} \log n)$ running time for MWM. Very recently, work by Battacharya, Kiss, and Saranurak [3] extended the MWU for packing LPs to the *partially dynamic setting*. When restricted to the MWM problem means the weight of edges either only increase or only decrease. However as packing LPs are more general than MWM, these algorithms are significantly more complicated and are slower by $\log n$ factors (and worse dependence on ε for [3]) when compared to our static and dynamic algorithms.

We remark that our algorithm, while it uses multiplicative weight updates, is unlike typical MWU algorithms as it has an additional monotonicity property. We only increase dual variables on one side of the matching.

1.4 Auction Algorithms

Auction algorithms are a class of primal dual algorithms for solving the MWM problem that view U as a set of *goods* to be sold, V as a set of *buyers*. The goal of the auction algorithm is to

[1]By *approximate solution* we mean a possibly fractional assignments of variables that obtains an approximately good LP objective. If we find such an approximate solution to MWM, fractional solutions need to be rounded to obtain an actual matching.

find a welfare-maximizing allocation of goods to buyers. The algorithm is commonly attributed Bertsekas [2], as well as to Demange, Gale, and Sotomayor [7].

An auction algorithm initializes the prices of all the goods $u \in U$ with a price $y_u = 0$ (our choice of y_u is intentional, as prices correspond directly to dual variables), and has buyers initially *unallocated*. For each buyer $v \in V$, the *utility* of that buyer upon being allocated $u \in U$ is $util(uv) = w(uv) - y_u$. The auction algorithm proceeds by asking an unallocated buyer $v \in V$ for the good they desire that maximizes their utility, i.e. for $u_v = \arg\max_{u \in N(v)} util(uv)$. If $util(u_v v) < 0$, the buyer remains unallocated. Otherwise the algorithm allocates u_v to v, then increases the price y_u to $y_u + \varepsilon$. The algorithm terminates when all buyers are either allocated or for every unallocated buyer v, it holds that $util(u_v v) < 0$. If the maximum weight among all the edges is w_{max}, then the auction algorithm terminates after $O(n\varepsilon^{-1} w_{max})$ rounds and outputs a matching that differs from the optimal by an additive factor of at most $n\varepsilon$.

1.5 Our contribution

We present the following modification of the auction algorithm:

When v is allocated u, increase y_u to $y_u + \varepsilon \cdot util(uv)$ instead of $y_u + \varepsilon$.

Note that this decreases $util(v)$ by a factor of $(1 - \varepsilon)$ and, thus, we will call algorithms with this modification *multiplicative auction algorithms*. Surprisingly, we were not able to find any literature on this simple modification. Changing the constant additive weight update to a multiplicative weight update has the effect of taking much larger steps when the weights are large, and so we are able to show that the algorithm can have no dependence on the size of the weights. In fact, we are able to improve the running time to $O(m\varepsilon^{-1} \log(\varepsilon^{-1}))$, the same as the fastest known matching algorithm of Duan and Pettie [8]. While the algorithm of [8] has the advantage that it works for general graphs and ours is limited to bipartite graphs, our algorithm is simpler as it avoids the scaling algorithm framework and is easier to implement.

Theorem 1.1. *Let $G = (U \cup V, E)$ be a weighted biparitite graph. There is a multiplicative auction algorithm running in time $O(m\varepsilon^{-1} \log(\varepsilon^{-1}))$ that finds a $(1 - \varepsilon)$-approximate maximum weight matching of G.*

Furthermore, it is straightforward to extend our algorithm to a setting where *vertices on one side are deleted* and *vertices on the other side are added with all incident edges given in sorted order of weight*. When the inserted edges are not sorted by weight, the running time per inserted edge increases by an additive term of $O(\log n)$ to sort all incident inserted edges.

Theorem 1.2. *Let $G = (U \cup V, E)$ be a weighted bipartite graph. There exists a dynamic data structure that maintains a $(1 - \varepsilon)$-approximate maximum weight matching of G and supports any arbitrary sequence of the following operations*

(1) Deleting a vertex in U

(2) Adding a new vertex into V along with all its incident edges sorted by weight

in total time $O(m\varepsilon^{-1} \log(\varepsilon^{-1}))$, where m is sum of the number of initially existing, and inserted edges.

2 The static algorithm

We assume that the algorithm is given as input some fixed $0 < \varepsilon' < 1$.

Notation For sake of notation let $N(u) = \{v \in V \mid uv \in E\}$ be the set of neighbors of $u \in U$ in G, and similarly for $N(v)$ for $v \in V$.

Preprocessing of the weights. Let $w_{max} > 0$ be the maximum weight edge of E. For our static auction algorithm we may ignore any edge $uv \in E$ of weight less than $\varepsilon' \cdot w_{max}/n$ as $w(M^*) \geq w_{max}$ as taking n of these small weight edges would not even contribute $\varepsilon' \cdot w(M^*)$ to the matching. Thus, we only consider edges of weight at least $\varepsilon' \cdot w_{max}/n$, which allows us to rescale all edge weights by dividing them by $\varepsilon' \cdot w_{max}/n$. As a result we can assume (by slight abuse of notation) in the following that the minimum edge weight is 1 and the largest edge weight w_{max} equals n/ε'. Furthermore, since we only care about approximations, we will also round down all edge weights to the nearest power of $(1+\varepsilon)$ for some $\varepsilon < \varepsilon'/2$ and, again by slight abuse of notation, we will use w to denote these edge weights. Formally to round, we define $\textsc{iLog}(x) = \lfloor \log_{1+\varepsilon}(x) \rfloor$ and $\textsc{Round}(x) = (1+\varepsilon)^{\textsc{iLog}(x)}$.

Let $k_{max} = \textsc{iLog}(w_{max}) = \textsc{iLog}(n/\varepsilon') = O(\varepsilon^{-1}\log(n/\varepsilon))$. Let k_{min} be the smallest integer such that $(1+\varepsilon)^{-k_{min}} \leq \varepsilon$. Observe that as $\log(1+\varepsilon) \leq \varepsilon$ for $0 \leq \varepsilon \leq 1$ it holds that

$$k_{min} \geq \frac{\log(\varepsilon^{-1})}{\log(1+\varepsilon)} \geq \varepsilon^{-1}\log(\varepsilon^{-1}).$$

Thus we see that $k_{min} = \Theta(\varepsilon^{-1}\log(\varepsilon^{-1}))$.

Algorithm. The algorithm first builds for every $v \in V$ a list Q_v of pairs (i, uv) for each edge uv and each value i with $-k_{min} \leq i \leq j_{uv} = \textsc{iLog}(w_{uv})$ and then sorts Q_v by decreasing value of i. After, it calls the function $\textsc{MatchR}(v)$ on every $v \in V$. The function $\textsc{MatchR}(v)$ matches v to the item that maximizes its utility and updates the price y_u according to our multiplicative update rule. While matching v, another vertex v' originally matched to v may become unmatched. If this happens, $\textsc{MatchR}(v')$ is called immediately after $\textsc{MatchR}(v)$.

```
Algorithm 2.1: MultiplicativeAuction($G = (U \cup V, E)$)

$M \leftarrow \emptyset$.
$y_u \leftarrow 0$ for all $u \in U$.
$j_v \leftarrow k_{max}$ for all $v \in V$                # This is only used in the analysis
$Q_v \leftarrow \emptyset$ for all $v \in V$.
For $v \in V$:
    1. For $u \in N(v)$:
        (a) $j_{uv} \leftarrow$ iLog($w(uv)$)
        (b) For $i$ from $j_{uv}$ to $-k_{min}$:
            i. Insert the pair $(i, uv)$ into $Q_v$.
    2. Sort all $(i, uv) \in Q_v$ so elements are in non-increasing order of $i$.
For $v \in V$:
    1. MatchR($v$).
Return $M$.
```

```
MatchR($v$)

While $Q_v$ is not empty:
    1. $(j, uv) \leftarrow$ the first element of $Q_v$, and remove it from $Q_v$.
    2. $j_v \leftarrow j$                                  # This is only used in the analysis
    3. $util(uv) \leftarrow w(uv) - y_u$
    4. If $util(uv) \geq (1+\varepsilon)^j$:
        (a) $y_u \leftarrow y_u + \varepsilon \cdot (util(uv))$      # $util(uv) \leftarrow (1-\varepsilon) \cdot util(uv)$
        (b) If $u$ was matched to $v'$ in $M$:
            • Remove $(u, v')$ from $M$
            • Add $(u, v)$ to $M$
            • MatchR($v'$)
        (c) Else:
            • Add $(u, v)$ to $M$
            • Return
```

Data structure. We store for each vertex $v \in V$ the list Q_v as well as its currently matched edge if it exists. In the pseudocode below we keep for each vertex v a value j_v corresponding to the highest weight threshold $(1+\varepsilon)^{j_v}$ that we will consider. This value is only needed in the analysis.

Running time. The creation and sorting of the lists Q_v takes time $O(|N(v)|(k_{max} + k_{min}))$ if we use bucket sort as there are only $k_{max} + k_{min}$ distinct weights. The running time of all calls to MATCHR(v) is dominated by the size of Q_v, as each iteration in MATCHR(v) removes an element of Q_v and takes $O(1)$ time. Thus, the total time is $O\left(\sum_{v \in V} |N(v)|(k_{max} + k_{min})\right) = O(m(k_{max} + k_{min})) = O(m\varepsilon^{-1} \log(n/\varepsilon))$.

Invariants maintained by the algorithm. Consider the following invariants maintained throughout by the algorithm:

Invariant 2.1. *For all $v \in V$, and all $u \in N(v)$, $util(uv) = w(uv) - y_u \leq (1+\varepsilon)^{j_v+1}$.*

Proof. This clearly is true at the beginning, since j_v is initialized to k_{max}, and

$$util(uv) = w(uv) < (1+\varepsilon)^{j_{uv}+1}.$$

As the algorithm proceeds, $util(uv)$ which equals $w(uv) - y_u$ only decreases as y_u only increases. Thus, we only have to make sure that the condition holds whenever j_v decreases. The value j_v only decreases from some value, say $j+1$, to a new value j, in MATCHR(v) and when this happens Q_v does not contain any pairs (j', uv) with $j' > j$ anymore. Thus, there does not exist a neighbor u of v with $util(uv) \geq (1+\varepsilon)^{j+1}$. It follows that when j_v decreases to j for all $u \in N(v)$ it holds that $util(uv) < (1+\varepsilon)^{j_v+1}$. □

Invariant 2.2. *If $uv \in M$, then for all other $u' \in N(v)$, $util(uv) \geq (1-2\varepsilon) \cdot util(u'v)$.*

Proof. When v was matched to u, right before we updated y_u, we had that $(1+\varepsilon)^{j_v} \leq util(uv)$ and, by Invariant 2.1, $util(u'v) \leq (1+\varepsilon)^{j_v+1}$. Thus, $(1+\varepsilon)util(uv) \geq util(u'v)$. The update of y_u decreases y_u by $\varepsilon \cdot util(uv)$, which decreases $util(uv)$ by a factor of $(1-\varepsilon)$, but does not affect $util(u'v)$. Thus we have now that:

$$util(uv) \geq (1-\varepsilon)(1+\varepsilon)^{-1} \cdot util(u'v) \geq (1-2\varepsilon) \cdot util(u'v).$$

□

Invariant 2.3. *If $u \in U$ is not matched, then $y_u = 0$. If $uv \in M$, then $y_u > 0$.*

Proof. If u is never matched, we never increment y_u, so it stays 0. The algorithm increments y_u by $\varepsilon \cdot util(uv) > 0$ when we add uv into the matching M. □

Invariant 2.4. *For all $v \in V$ for which* MATCHR(v) *was called at least once, either v is matched, or Q_v is empty.*

Proof. MATCHR(v) terminates (i) after it matches v and recurses or (ii) if Q_v is empty. It is possible that for some other $v' \in V$ with $v' \neq v$, that v becomes temporarily unmatched during MATCHR(v'), but we would immediately call MATCHR(v) to rematch v. □

Approximation factor. We will show the approximation factor of the matching M found by the algorithm by primal dual analysis. We remark that it is possible to show this result purely combinatorially as well which we include in Appendix A, as it may be of independent interest. We will show that this M and a vector y satisfy the complementary slackness condition up to a $1 \pm \varepsilon$ factor, which implies the approximation guarantee. This was proved by Duan and Pettie [8] (the original lemma was for general matchings, we have specialized it here to bipartite matchings).

Lemma 2.1 (Lemma 2.3 of [8]). *Let M be a matching and let y be an assignment of the dual variables. Suppose y is a valid solution to the LP in the following approximate sense: For all $uv \in E, y_u + y_v \geq (1-\varepsilon_0) \cdot w(uv)$ and for all $e \in M$, $y_u + y_v \leq (1+\varepsilon_1) \cdot w(uv)$. If the y-values of all unmatched vertices are zero, then M is a $\big((1+\varepsilon_1)^{-1}(1-\varepsilon_0)\big)$-approximate maximum weight matching.*

This lemma is enough for us to prove the approximation factor of our algorithm.

Lemma 2.2. MULTIPLICATIVEAUCTION$(G = (U \cup V, E))$ *outputs a $(1-\varepsilon')$-approximate maximum weight matching of the bipartite graph G.*

Proof. Let $\varepsilon > 0$ be a parameter depending on ε' that we will choose later. We begin by choosing an assignment of the dual variables y_u for $u \in U$ and y_v for $v \in V$. Let all y_u's be those obtained by the algorithm for $u \in U$. For $v \in V$, let $y_v = 0$ if v is not matched in M and $y_v = util(uv) = w(uv) - y_u$ if v is matched to u in M. By Invariant 2.3 all unmatched vertices $u \in U$ have $y_u = 0$.

Observe that for $uv \in M$ we have $y_u + y_v = util(uv)$. It remains to show that for $uv \notin M$ we have that $y_u + y_v \geq (1-\varepsilon_0)w(uv)$ for some $\varepsilon_0 > 0$. First we consider if v is unmatched, so $y_v = 0$. Since v is unmatched, by Invariant 2.4 then for all $u \in N(v)$, we must have $util(uv) < (1+\varepsilon)^{-k_{min}} \leq \varepsilon$. Since we rescaled weights so that $w(uv) \geq 1$, we know that $util(uv) < \varepsilon \leq \varepsilon \cdot w(uv)$. Furthermore, observe that as $y_u = w(uv) - util(uv)$ by definition of utility, it follows that:

$$y_u + y_v = y_u = w(uv) - util(uv) > (1-\varepsilon)w(uv). \tag{1}$$

Now we need to consider if v was matched to some vertex $u' \neq u$. To do so we use Invariant 2.2:

$$\begin{aligned} y_u + y_v &= y_u + util(u'v) && \text{By definition of } y \\ &\geq y_u + (1-2\varepsilon) \cdot util(uv) && \text{By Invariant 2.2} \\ &= y_u + (1-2\varepsilon) \cdot (w(uv) - y_u) && \text{By definition of } util \\ &\geq (1-2\varepsilon)w(uv) + 2\varepsilon \cdot y_u && \\ &\geq (1-2\varepsilon)w(uv) && \text{Since } y_u \geq 0 \end{aligned}$$

Thus we have satisfied Lemma 2.1 with $\varepsilon_0 = 2\varepsilon$ and $\varepsilon_1 = 0$. Setting $\varepsilon = \varepsilon'/2$ gives us a $(1-\varepsilon')$-approximate maximum weight matching. ☐

Thus we have shown the following result that is weaker than what we have set out to prove by a factor of $\log(n\varepsilon^{-1})$ that we will show how to get rid of in the next section.

Theorem 2.3. *Let $G = (U \cup V, E)$ be a weighted biparitite graph. There exists a multiplicative auction algorithm running in time $O(m\varepsilon^{-1}\log(n\varepsilon^{-1}))$ that finds a $(1-\varepsilon)$-approximate maximum weight matching of G.*

2.1 Improving the running time

To improve the running time to $O(m\varepsilon^{-1}\log(\varepsilon^{-1}))$, we observe that all we actually need for Lemma 2.2 in Equation (1) is that $util(uv) \leq \varepsilon \cdot w(uv)$. Recall that $j_{uv} = \textsc{iLog}(w(uv))$. Thus it suffices if we change line (b) in MultiplicativeAuction to range from j_{uv} to $j_{uv} - k_{min}$, since:

$$(1+\varepsilon)^{j_{uv}-k_{min}} = (1+\varepsilon)^{-k_{min}} \cdot (1+\varepsilon)^{j_{uv}} \leq \varepsilon \cdot w(uv).$$

This change implies that we insert $O(k_{min}|N(v)|)$ items into Q_v for every $v \in V$. However, sorting Q_v for every vertex individually, even with bucket sort, would be too slow. We will instead perform one bucket sort on all the edges, then go through the weight classes in decreasing order to insert the pairs into the corresponding Q_v. We explicitly give the pseudocode below as MultiplicativeAuction+.

```
Algorithm 2.2: MultiplicativeAuction+($G = (U \cup V, E)$)

$M \leftarrow \emptyset$.
$y_u \leftarrow 0$ for all $u \in U$.
$Q_v \leftarrow \emptyset$ for all $v \in V$.
$L_i \leftarrow \emptyset$ for all $i$ from $-k_{min}$ to $k_{max}$.
For $uv \in E$:
    1. $j_{uv} \leftarrow \textsc{iLog}(w(uv))$
    2. For $i$ from $j_{uv}$ to $j_{uv} - k_{min}$:
        (a) Insert the pair $(i, uv)$ into $L_i$.
For $i$ from $k_{max}$ to $-k_{min}$:
    1. For all $(i, uv) \in L_i$:
        (a) Insert the pair $(i, uv)$ to the back of $Q_v$.
For $v \in V$:
    1. MatchR($v$).
Return $M$.
```

New runtime. Bucket sorting all mk_{min} pairs and initializing the sorted Q_v for all $v \in V$ takes total time $O(mk_{min} + (k_{max} + k_{min})) = O(m\varepsilon^{-1}\log(\varepsilon^{-1}))$. The total amount of work done in MatchR(v) for a vertex $v \in V$ is $O(|N(v)|k_{min})$ which also sums to $O(m\varepsilon^{-1}\log(\varepsilon^{-1}))$. Thus we get our desired running time and have proven our main theorem that we restate here.

Theorem 1.1. *Let $G = (U \cup V, E)$ be a weighted biparitite graph. There is a multiplicative auction algorithm running in time $O(m\varepsilon^{-1}\log(\varepsilon^{-1}))$ that finds a $(1-\varepsilon)$-approximate maximum weight matching of G.*

3 Dynamic algorithm

There are many monotonic properties of our static algorithm. For instance, for all $u \in U$ the y_u values strictly increase. As another example, for all $v \in V$ the value of j_v strictly decreases. These monotonic properties allow us to extend MultiplicativeAuction+ to a dynamic setting with the following operations.

Theorem 1.2. *Let $G = (U \cup V, E)$ be a weighted bipartite graph. There exists a dynamic data structure that maintains a $(1-\varepsilon)$-approximate maximum weight matching of G and supports any arbitrary sequence of the following operations*

(1) Deleting a vertex in U

(2) Adding a new vertex into V along with all its incident edges sorted by weight

in total time $O(m\varepsilon^{-1}\log(\varepsilon^{-1}))$, where m is sum of the number of initially existing, and inserted edges.

Type (1) operations: Deleting a vertex in U. To delete a vertex $u \in U$, we can mark u as deleted and skip all edges uv in Q_v for any $v \in V$ in all further computation. If u were matched to some vertex $v \in V$, that is if there exists an edge $uv \in M$, we need to unmatch v and remove uv from M. All our invariants hold except Invariant 2.4 for the unmatched v. To restore this invariant we simply call MatchR(v).

Type (2) operations: Adding a new vertex to V along with all incident edges. To add a new vertex v to V with ℓ incident edges to $u_1v, ..., u_\ell v$ with $w(u_1v) > \cdots > w(u_\ell v)$, we can create the queue Q_v by inserting the $O(\varepsilon^{-1}\log(\varepsilon^{-1}))$ pairs such that it is non-increasing in the first element of the pair. Afterwards we call MatchR(v). All invariants hold after doing so.

If the edges are not given in sorted order, we can sort the ℓ edges in $O(\ell \log \ell)$ time, or in $O(\ell + \varepsilon^{-1}\log(w(u_1v)/w(u_\ell v)))$ time by bucket sort.

Acknowledgements

The first author thanks Chandra Chekuri for useful discussions about this paper.

Monika Henzinger: This work was done in part at the University of Vienna. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019564 "The Design of Modern Fully Dynamic Data Structures (MoDynStruct)" and from the Austrian Science Fund (FWF) project "Fast Algorithms for a Reactive Network Layer (ReactNet)", P 33775-N, with additional funding from the *netidee SCIENCE Stiftung*, 2020–2024.

A Combinatorial proof of Lemma 2.2

We start with a simple lemma.

Lemma A.1. *Let $G = (U \cup V, E)$ be a weighted bipartite graph. Let M be the matching found by* MultiplicativeAuction+(G) *for $\varepsilon > 0$, and M' be any other matching. Then for any alternating* *path, i.e. a set of edges of the form u_1v_1, u_2v_1, u_2v_2, ..., u_kv_k, $u_{k+1}v_k$ with all edges of $u_iv_i \in M'$ and $u_{i+1}v_i \in M$, we have that:*

$$(1-2\varepsilon) \cdot \sum_{i=1}^{k} w(u_iv_i) \leq \sum_{i=1}^{k} w(u_{i+1}v_i) + (1-2\varepsilon) \cdot y_{u_1} - y_{u_{k+1}}$$

Proof. By Invariant 2.2, for all i from 1 to k, since M matched v_{i+1} to u_i we have that:

$$(1-2\varepsilon) util(u_iv_i) \leq util(u_iv_{i+1})$$

Adding all such equations together we get

$$(1-2\varepsilon) \cdot \sum_{i=1}^{k} util(u_iv_i) \leq \sum_{i=1}^{k} util(u_iv_{i+1})$$

$$(1-2\varepsilon) \cdot \sum_{i=1}^{k} (w(u_iv_i) - y_{u_i}) \leq \sum_{i=1}^{k} \left(w(u_{i+1}v_i) - y_{u_{i+1}}\right)$$

$$(1-2\varepsilon) \cdot \left(\sum_{i=1}^{k} w(u_iv_i) - \sum_{i=1}^{k} y_{u_i} \right) \leq \sum_{i=1}^{k} w(u_{i+1}v_i) - \sum_{i=1}^{k} y_{u_{i+1}}$$

$$(1-2\varepsilon) \cdot \sum_{i=1}^{k} w(u_iv_i) \leq \sum_{i=1}^{k} w(u_{i+1}v_i) + (1-2\varepsilon) \cdot y_{u_1} - y_{u_{k+1}}$$

□

Theorem A.2. *Let $G = (U \cup V, E)$ be a weighted bipartite graph and $\varepsilon' > 0$ be an input parameter. Let M be the matching found by* MultiplicativeAuction+(G) *with $\varepsilon = \varepsilon'/2$. M is a $(1-\varepsilon')$-approximate maximum weight matching of the bipartite graph G.*

Proof. Let M^* be a maximum weight matching of G. Consider the symmetric difference of M with M^*. It consists of paths and and even cycles. It suffices to show that the weight obtained by M on the path or even cycle is at least $(1-\varepsilon)$ the weight of M^*. We consider the following cases:

1. Consider any even cycle u_1v_1, u_2v_1, u_2v_2, ..., u_kv_k, u_1v_k with $u_iv_i \in M^*$ for all $i = 1, ..., k$ and the other edges in M. Applying Lemma A.1 with $u_{k+1} = u_1$, and by Invariant 2.3 $y_{u_1} > 0$ as u_1 is matched gives:

$$(1-2\varepsilon) \cdot \sum_{i=1}^{k} w(u_iv_i) \leq \sum_{i=1}^{k} w(u_{i+1}v_i) + (1-2\varepsilon)y_{u_1} - y_{u_1} < \sum_{i=1}^{k} w(u_{i+1}v_i).$$

2a. Consider any even length path u_1v_1, u_2v_1, u_2v_2, ..., u_kv_k, $u_{k+1}v_k$ with $u_iv_i \in M^*$ for all $i = 1, ..., k$ and the other edges in M. By Invariant 2.3 u_1 is unmatched in M so $y_{u_1} = 0$, and u_{k+1} is matched so $y_{u_{k+1}} > 0$. Applying Lemma A.1 gives:

$$(1-2\varepsilon) \cdot \sum_{i=1}^{k} w(u_iv_i) \leq \sum_{i=1}^{k} w(u_{i+1}v_i) + (1-2\varepsilon) \cdot 0 - y_{u_{k+1}} < \sum_{i=1}^{k} w(u_{i+1}v_i).$$

2b. Consider any odd length path u_1v_1, u_2v_1, u_2v_2, ..., u_kv_k, $u_{k+1}v_k$, u_{k+1}, v_{k+1} with $u_iv_i \in M^*$ for all $i = 1, ..., k$ and the other edges in M. Since v_k is unmatched in M, we have that $w(u_{k+1}v_{k+1}) - y_{u_{k+1}} = util(u_{k+1}v_{k+1}) \leq \varepsilon w(u_{k+1}v_{k+1})$. Rearranging, we get that $y_{u_{k+1}} \geq (1-\varepsilon)w(u_{k+1}v_{k+1}) > (1-2\varepsilon)w(u_{k+1}v_{k+1})$. Adding this equation to Lemma A.1, and by Invariant 2.3 we have $y_{u_1} = 0$, so:

$$(1-2\varepsilon) \cdot \sum_{i=1}^{k+1} w(u_iv_i) < \sum_{i=1}^{k} w(u_{i+1}v_i) + (1-2\varepsilon) \cdot y_{u_1} = \sum_{i=1}^{k} w(u_{i+1}v_i).$$

2c. Consider any even length path u_0v_1, u_1v_1, u_2v_1, u_2v_2, ..., u_kv_k, $u_{k+1}v_k$ with $u_iv_i \in M^*$ for all $i = 1, ..., k$ and the other edges in M. By Invariant 2.2, $w(u_1v_0) - y_{u_0} \geq (1-2\varepsilon)(w(u_1v_1) - y_{u_1})$, Adding this inequality to what we get when we apply Lemma A.1 to the path starting at u_1v_1, and remarking that u_0 and u_{k+1} are matched so Invariant 2.3 applies gives:

$$(1-2\varepsilon) \cdot \sum_{i=1}^{k+1} w(u_iv_i) \leq \sum_{i=0}^{k} w(u_{i+1}v_i) - y_{u_0} - y_{u_{k+1}} < \sum_{i=1}^{k} w(u_{i+1}v_i).$$

In all cases we achieve $(1-2\varepsilon)$ the weight of M^*. We may choose ε such that $\varepsilon = \varepsilon'/2$, then the theorem holds. □

References

[1] Zeyuan Allen-Zhu and Lorenzo Orecchia. Nearly linear-time packing and covering LP solvers - achieving width-independence and -convergence. *Math. Program.*, 175(1-2):307–353, 2019.

[2] Dimitri P. Bertsekas. A new algorithm for the assignment problem. *Math. Program.*, 21(1):152–171, 1981.

[3] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic algorithms for packing-covering lps via multiplicative weight updates. *CoRR*, abs/2207.07519, 2022.

[4] Bartlomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Online bipartite matching in offline time. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 384–393. IEEE Computer Society, 2014.

[5] Chandra Chekuri and Kent Quanrud. Randomized MWU for positive lps. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 358–377. SIAM, 2018.

[6] 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. *CoRR*, abs/2203.00671, 2022.

[7] Gabrielle Demange, David Gale, and Marilda Sotomayor. Multi-item auctions. *Journal of political economy*, 94(4):863–872, 1986.

[8] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1):1:1–1:23, 2014.

[9] Manoj Gupta and Richard Peng. Fully dynamic $(1+\epsilon)$-approximate matchings. In *54th Symposium on Foundations of Computer Science, FOCS*, pages 548–557. IEEE Computer Society, 2013.

[10] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms (invited talk). In James Aspnes and Othon Michail, editors, *1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022, March 28-30, 2022, Virtual Conference*, volume 221 of *LIPIcs*, pages 1:1–1:47. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[11] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proc. of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30, 2015.

[12] Christos Koufogiannakis and Neal E. Young. A nearly linear-time PTAS for explicit fractional packing and covering linear programs. *Algorithmica*, 70(4):648–674, 2014.

[13] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[14] Hung Le, Lazar Milenkovic, Shay Solomon, and Virginia Vassilevska Williams. Dynamic matching algorithms under vertex updates. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 96:1–96:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[15] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[16] Kent Quanrud. Nearly linear time approximations for mixed packing and covering problems without data structures or randomization. In Martin Farach-Colton and Inge Li Gørtz, editors, *3rd Symposium on Simplicity in Algorithms, SOSA 2020, Salt Lake City, UT, USA, January 6-7, 2020*, pages 69–80. SIAM, 2020.

[17] Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

[18] Di Wang, Satish Rao, and Michael W. Mahoney. Unified acceleration method for packing and covering problems via diameter reduction. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 50:1–50:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[19] Neal E. Young. Nearly linear-time approximation schemes for mixed packing/covering and facility-location linear programs. *CoRR*, abs/1407.3015, 2014.