Mathematical Programming
https://doi.org/10.1007/s10107-024-02066-3

FULL LENGTH PAPER

Series B

Multiplicative auction algorithm for approximate maximum weight bipartite matching

Da Wei Zheng[1] · Monika Henzinger[2]

Received: 6 August 2023 / Accepted: 19 January 2024

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 $O(m\varepsilon^{-1})$, beating the running time of the fastest known approximation algorithm of Duan and Pettie [JACM '14] that runs in $O(m\varepsilon^{-1}\log\varepsilon^{-1})$. 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 $O(m\varepsilon^{-1})$, where m is the sum of the number of initially existing and inserted edges.

Mathematics Subject Classification 68R10

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

An earlier version of this paper appeared in the 24th International Conference on Integer Programming and Combinatorial Optimization (IPCO 2023) with a slightly slower algorithm running in $O(m\varepsilon^{-1}\log\varepsilon^{-1})$ time.

✉ Da Wei Zheng
dwzheng2@illinois.edu

✉ Monika Henzinger
monika.henzinger@ista.ac.at

1 University of Illinois Urbana-Champaign, Urbana, IL, USA

2 Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Published online: 06 March 2024

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 [18] 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 [21] 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, Chen et al. [11] 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 prior best approximate algorithm is by Duan and Pettie [13] 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 $O(m\varepsilon^{-1})$ for bipartite graphs. This is a modest improvement of a $\log\varepsilon^{-1}$ factor over the prior algorithm of Duan and Pettie [13] which works in general graphs. 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. No algorithm that allows both these operations with running time faster than recomputation from scratch was known prior.

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 [15] for a survey of the current state of the art for the fully dynamic setting. 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 [16].

Dynamic $(1-\varepsilon)$*-approximate matchings* For *general weighted* graphs Gupta and Peng [14] 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 $[1, w_{max}]$. There are also some results for bipartite graphs in *partially dynamic* settings. In the *incremental* setting, edges are only inserted, and *decremental* setting, edges are only deleted. For unweighted bipartite

graphs, the fastest known decremental algorithm is by Bernstein, Probst Gutenberg, and Saranurak [4] achieves update times of $O(\varepsilon^{-4}\log^3(n))$ per edge deletion. For incremental algorithms Blikstad and Kiss [8] achieve update times of $O(\varepsilon^{-6})$ time per edge insertion. These results can be made to work in weighted graphs by a meta theorem of Bernstein, Dudeja, and Langley [5]. Their theorem states that any dynamic algorithm on an unweighted bipartite graph can be transformed into a dynamic algorithm on weighted bipartite graph at the expense of an extra $(1/\varepsilon)^{O(1/\varepsilon)}\log n$ factor.
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. [9] 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. [19] 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 $O(m\varepsilon^{-1})$, where m is the sum of the number of initially existing and inserted 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 [23] that the below LP is integral, that is the optimal solution has all variables $x_{uv}\in\{0,1\}$.

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

We can also consider the dual problem of weighted vertex cover 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, 10, 17, 22, 24, 25]. Typically the running times for solving these LPs have a dependence on ε of ε^{-2}, e.g. the algorithm of Koufogiannakis and Young [17] 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 [7] 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. Using similar ideas with MWUs, Assadi [2] recently derived a simple semi-streaming algorithm for bipartite matchings. However as packing LPs are more general than MWM, these algorithms are significantly more complicated and are slower by $\log n$ factors (and sometimes worse dependence on ε e.g. in [7]) when compared to our 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 find a welfare-maximizing allocation of goods to buyers. The algorithm is attributed to Bertsekas [6], as well as to Demange, Gale, and Sotomayor [12].

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 $\text{util}(uv) = w(uv) - y_u$. The auction algorithm proceeds by asking an unallocated buyer $v \in V$ for the good they desire that maximizes their utility, i.e. for $u_v = \arg\max_{u\in N(v)} \text{util}(uv)$. If $\text{util}(u_v v) < 0$, the buyer remains unallocated. Otherwise the algorithm allocates u_v to v, then increases the price y_u to $y_u + \varepsilon$. The

[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.

algorithm terminates when all buyers are either allocated or for every unallocated buyer v, it holds that $\text{util}(u_v v) < 0$. 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 $n\varepsilon$.

There have been a recent resurgence in interest in auction algorithms. Assadi, Liu, and Tarjan [3] used the auction algorithm for matchings in unweighted graphs in semi-streaming and massively parallel computing (MPC) settings. This work was generalized for weighted bipartite graphs in the same settings by Liu, Ke, and Kholler [20].

1.5 Our contribution

We present the following modification of the auction algorithm:

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

Note that this decreases $\text{util}(v)$ by at least a factor of $(1 - \varepsilon)$ as well as increases y_u by at least a factor of $(1 + \varepsilon)$. 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 $O(m\varepsilon^{-1})$, faster than the prior approximate matching algorithm of Duan and Pettie [13] that ran in $O(m\varepsilon^{-1} \log \varepsilon^{-1})$. While the algorithm of [13] 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 and ε be a value such that $1 > \varepsilon > 0$. There is a multiplicative auction algorithm running in time $O(m\varepsilon^{-1})$ 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 $O(\log \log n)$ to sort the log of the weights of 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 with all its incident edges sorted by weight*

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

2 The static algorithm

2.1 A slower algorithm

For sake of exposition we will first present a slower algorithm that runs in near-linear time in the number of edges that will use the following update rule:

$$\textit{When } u \textit{ is allocated to } v, \quad \textit{increase } y_u \textit{ to } y_u + \varepsilon \cdot \text{util}(uv)$$

We assume that the algorithm is given as input some fixed $0 < \varepsilon' < 1$, and the goal is to find a $(1-\varepsilon')$-approximate MWM. We will also assume that $m = \Omega(n)$, as a graph with m edges has at most $2m$ vertices that have at least one incident edge. If $2m < n$, then we may discard the isolated vertices and reduce n.

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 $\varepsilon < \varepsilon'/2$. We define $\text{ILOG}(x) = \lfloor \log_{1+\varepsilon}(x) \rfloor$, and we will only care about weights after applying this operation.

Let $k_{max} = \text{ILOG}(w_{max}) = \text{ILOG}(n/\varepsilon') = O(\varepsilon^{-1}\log(n/\varepsilon))$. 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 $-k_{min} \leq i \leq j_{uv} = \text{ILOG}(w_{uv})$ and then sorts Q_v by decreasing value of i. After, it calls the function $\text{MATCHR}(v)$ on every $v \in V$. The function $\text{MATCHR}(v)$ 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, $\text{MATCHR}(v')$ is called immediately after $\text{MATCHR}(v)$.

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

$M \leftarrow \emptyset$.
$y_u \leftarrow 0$ for all $u \in U$.
$y_v \leftarrow 0$ for all $v \in V$ # This is only used in the analysis
$j_v \leftarrow k_{max}$ 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 pairs $(i, uv) \in Q_v$ so the pairs are in non-increasing order of i.

For $v \in V$:

3. MATCHR(v).

Return M.

MATCHR(v)

While Q_v is not empty:

1. $(j, uv) \leftarrow$ the first element of 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) $< (1+\varepsilon)^j$:
   - Remove (j, uv) from Q_v.
5. Else:
   (a) $y_v \leftarrow$ util(uv) # This is only used in the analysis
   (b) $y_u \leftarrow y_u + \varepsilon \cdot$ (util(uv)) # Update rule
   (c) If u was matched to v' so that $uv' \in M$:
       - Remove (u, v') from M
       - $y_{v'} \leftarrow 0$ # This is only used in the analysis
       - Add (u, v) to M
       - MATCHR(v')
       - Return
   (d) 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 we keep for each vertex v a value j_v corresponding to the highest weight threshold $(1+\varepsilon)^{j_v}$ that we will consider. We also

keep a value y_v which corresponds to the utility v receives *before* we update the price y_u when v is matched to u. Note that j_v and y_v are 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. The algorithm maintains five different invariants.

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

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

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

As the algorithm proceeds, $\text{util}(uv)$ 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 $\text{util}(uv) \geq (1 + \varepsilon)^{j+1}$. It follows that when j_v decreases to j for all $u \in N(v)$ it holds that $\text{util}(uv) < (1+\varepsilon)^{j_v+1}$. □

Invariant 2 *For all $u \in U$ $y_u \geq 0$ and y_u never decreases over the course of the algorithm. Furthermore, if $u \in U$ is not matched, then $y_u = 0$.*

Proof We initialize y_u to 0. If u is never matched, we never change y_u, so it stays 0. Throughout the algorithm, we only ever increase y_u. □

Invariant 3 *For all $v \in V$ for which* MATCHR(v) *was called at least once, if v is unmatched, then $y_v = 0$ and Q_v is empty. Furthermore, for all $u \in N(v)$ we have that $y_u + y_v = y_u > (1 - \varepsilon) \cdot w(uv)$.*

Proof MATCHR(v) terminates (i) after it matches v and recurses or (ii) if Q_v is empty. Initially v is unmatched and y_v is set to 0. If v is matched, it is possible that for some $v' \in V$, $v' \neq v$, that v becomes temporarily unmatched during MATCHR(v') and y_v is set to 0, but MATCHR(v) will be immediately called again. Thus, whenever v is unmatched, $y_v = 0$.

Hence, if the last call to MATCHR(v) does not result in v being matched, then this means that Q_v must be empty and $y_v = 0$. Since Q_v is empty, then for all $u \in N(v)$, we must have $\text{util}(uv) < (1+\varepsilon)^{-k_{min}} \leq \varepsilon$. Since we rescaled weights so that $w(uv) \geq 1$, we know that $\text{util}(uv) < \varepsilon \leq \varepsilon \cdot w(uv)$. Note that $\text{util}(uv) = w(uv) - y_u^*$ where y_u^* denotes the value of y_u before u was matched, and $y_u \geq y_u^*$. Thus,

$$y_u \geq y_u^* = w(uv) - \text{util}(uv) > (1 - \varepsilon) \cdot w(uv).$$

□

Invariant 4 *If $v \in V$ is matched to $u \in U$, then for all $u' \in N(v)$, $y_{u'} + y_v \geq (1-\varepsilon) \cdot w(u'v)$ for as long as v stays matched.*

Proof Note that y_v doesn't change as long as v stays matched, and for all $u' \in N(v)$, $y_{u'}$ can only increase by Invariant 2, so it suffices to prove $y_{u'} + y_v \geq (1-\varepsilon) \cdot w(u'v)$ right after v was matched to u.

Let y_u^* be the value of y_u right before v was matched to u. Note that $y_v = w(uv) - y_u^*$. For $u' = u$, we know that $y_v + y_u^* = w(uv)$, and, by Invariant 2, $y_u^* \leq y_u$ so $y_v + y_u \geq w(uv)$.

For all other $u' \in U$, right before we updated y_u, we had that $(1+\varepsilon)^{j_v} \leq \text{util}(uv)$ and, by Invariant 1, $\text{util}(u'v) \leq (1+\varepsilon)^{j_v+1}$. Thus, $(1+\varepsilon) \cdot \text{util}(uv) \geq \text{util}(u'v)$, so that $\text{util}(uv) = w(uv) - y_u^* = y_v \geq (1+\varepsilon)^{-1} \cdot (w(u'v) - y_{u'})$. As $y_{u'} \geq 0$ by Invariant 2, it follows that:

$$y_v + y_{u'} \geq (1+\varepsilon)^{-1} \cdot w(u'v) + (1 - (1+\varepsilon)^{-1}) \cdot y_{u'} \geq (1-\varepsilon) \cdot w(u'v),$$

as $1 > \varepsilon > 0$. □

Invariant 5 *If $v \in V$ is matched to $u \in U$, then $y_u + y_v \leq (1+\varepsilon) \cdot w(uv)$ for as long as v remains matched to u.*

Proof Note that y_u and y_v don't change as long as v remains matched to u. Let y_u^* denote the value of y_u right before the update rule of line 5(b) in MATCHR. Then observe $y_u = y_u^* + \varepsilon \cdot (w(uv) - y_u^*)$, and $y_v = w(uv) - y_u^*$. Thus,

$$\begin{aligned} y_u + y_v &= y_u^* + \varepsilon \cdot (w(uv) - y_u^*) + w(uv) - y_u^* \\ &= (1+\varepsilon) \cdot w(uv) - \varepsilon \cdot y_u^* \\ &\leq (1+\varepsilon) \cdot w(uv). \end{aligned}$$

□

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. 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 used by Duan and Pettie [13] (the original lemma was for general matchings, we have specialized it here to bipartite matchings).

Lemma 2.1 (Lemma 2.3 of [13]) *Let M be a matching and let y be an assignment of the dual variables. Suppose y is a complementary solution to M in the following approximate sense:*

(i) *For all $uv \in E$, $y_u + y_v \geq (1-\varepsilon_0) \cdot w(uv)$,*
(ii) *For all $e \in M$, $y_u + y_v \leq (1+\varepsilon_1) \cdot w(uv)$,*
(iii) *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.

Proof Let M^* be the maximum weight matching.

$$\begin{aligned} w(M) &= \sum_{uv \in M} w(uv) \\ &\geq \sum_{uv \in M} (1+\varepsilon_1)^{-1} \cdot (y_u + y_v) && \text{by (ii)} \\ &= (1+\varepsilon_1)^{-1} \left(\sum_{u \in U} y_u + \sum_{v \in V} y_v \right) && \text{by (iii)} \\ &\geq (1+\varepsilon_1)^{-1} \sum_{uv \in M^*} (y_u + y_v) && \text{as } M^* \text{ is a matching} \\ &\geq (1+\varepsilon_1)^{-1} \sum_{uv \in M^*} (1-\varepsilon_0) \cdot w(uv) && \text{by (i)} \\ &\geq (1-\varepsilon_0)(1+\varepsilon_1)^{-1} w(M^*) \end{aligned}$$

□

This lemma along with our invariants 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 $v \in V$ as exactly the values used by the algorithm at termination. It remains to verify that we satisfy the conditions of Lemma 2.1. Property (i) is satisfied by Invariant 3 or Invariant 4 (depending on whether v is matched or not) for $\varepsilon_0 = \varepsilon$. Property (ii) is satisfied by Invariant 5 for $\varepsilon_1 = \varepsilon$. Property (iii) is satisfied by Invariant 2 and Invariant 3. Thus we have satisfied Lemma 2.1 with $\varepsilon_0 = \varepsilon$ and $\varepsilon_1 = \varepsilon$. Setting $\varepsilon = \varepsilon'/2$ gives us a $(1-\varepsilon')$-approximate maximum weight matching. □

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 Sect. 2.2.

Theorem 2.1 *Let* $G = (U \cup V, E)$ *be a weighted biparitite graph, and* ε *be a value such that* $1 > \varepsilon > 0$. *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.

2.2 Improving the running time

Variations to the update rule We remark that there is some flexibility in choosing the update rule in line 5(a) of MATCHR.

Observation 2.1 *To compute an* $(1-\varepsilon')$-*approximate maximum weight matching the update rule in line 5(a) of* MATCHR *can be any of the following:*

(1) $y_u \leftarrow y_u + \delta \cdot (\text{util}(uv))$, *with* $\delta \leq \varepsilon'/2$,
(2) $y_u \leftarrow y_u + \delta \cdot (y_u)$, *with* $\delta \leq \varepsilon'/2$,
(3) $y_u \leftarrow y_u + \delta \cdot (w(uv))$, *with* $\delta \leq \varepsilon'/4$.

Proof It suffices to verify that all invariants hold for different update rules. Invariant 1, 2, 3, and 4 all hold regardless of the update rule, as they only use the fact that y_u is non-decreasing throughout the algorithm, so we will only focus on Invariant 5.

We proved that update rule (1) works in Sect. 2.1 for $\delta = \varepsilon = \varepsilon'/2$. Note that if we chose an $\delta < \varepsilon$, we would still have $y_u \leq y_u^* + \varepsilon(w(uv) - y_u^*)$, and Invariant 5 holds.

To prove update rule (2) works for $\delta \leq \varepsilon = \varepsilon'/2$, let $v \in V$ be matched to $u \in U$, and y_u^* be the value of y_u right before the update rule. Observe that $y_u = (1+\delta) \cdot y_u^* \leq (1+\varepsilon) \cdot y_u^*$ and $y_v = w(uv) - y_u^*$. Furthermore $y_u^* \leq w(uv)$ as otherwise $\text{util}(uv) < 0$ and v would not be trying to match to u. Thus,

$$y_u + y_v \leq (1+\varepsilon) \cdot y_u^* + w(uv) - y_u^* = w(uv) + \varepsilon \cdot y_u^* \leq (1+\varepsilon) \cdot w(uv).$$

Since we have shown that either update rule (1) or (2) work, we can choose the larger of the two update rules, i.e. the update of adding $\delta \cdot \max\{util(uv), y_u\}$ is also a valid update rule. However, as $\text{util}(uv) + y_u = w(uv)$, this means that $\delta \cdot (w(uv)) \leq \varepsilon'/4 \cdot (w(uv))$, so (3) is also a valid update rule.

Remarks. Update rule (2) offers an alternative way to implement the algorithm with a running time of $O(m\varepsilon^{-1} \log(n\varepsilon^{-1}))$. Update rule (3) shows that v can update the value of y_u at most $O(\varepsilon^{-1})$ times before $\text{util}(uv)$ becomes non-positive, so using update rule (3) results in at most $O(m\varepsilon^{-1})$ total updates. Furthermore, a careful reader may have noticed that Invariant 3 only requires for an edge uv that $\text{util}(uv) \leq \varepsilon \cdot w(uv)$ when we stop considering that edge, so it suffices to only consider edges in multiples of $\varepsilon/2$ and stop considering an edge when it falls below a value of $\varepsilon \cdot w(uv)$.
Improved algorithm For simplicity of the exposition we will assume $2\varepsilon^{-1}$ is a positive integer (otherwise we can choose a slightly smaller ε). To improve the running time to $O(m\varepsilon^{-1})$, we use the observation in the above remark that every edge only needs to be updated $O(\varepsilon^{-1})$ times if we use update rule (3) and we only need to consider edges in multiples of $\varepsilon/2$. Thus it suffices if we change line 1.(b) in MULTIPLICATIVEAUCTION to insert copies of an edge uv if it has weight of the form $j \cdot w(uv)$ for some $j = \varepsilon, 3\varepsilon/2, 2\varepsilon, \ldots, 1$, after rounding down to the nearest power of $(1+\varepsilon)$. This change implies that we insert $O(\varepsilon^{-1}|N(v)|)$ items into Q_v for every $v \in V$. However, sorting Q_v for every vertex individually, would be too slow.

We will instead sort on all the rounded edge weights at once, as we have $O(m\varepsilon^{-1})$ total copies of the edges that can take on values of integers between $-k_{min}$ and k_{max}. As $k_{max} + k_{min} = O(n\varepsilon^{-1} \log \varepsilon^{-1}) = \text{poly}(n)$, we can actually use radix sort to sort all the edges in linear time. Afterwards, we can 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 $v \in V$.
$L \leftarrow \emptyset$
For $uv \in E$:

1. For j from ε to 1 in multiples of $\varepsilon/2$:
   (a) $i \leftarrow$ ILOG($j \cdot w(uv)$)
   (b) Insert the pair (i, uv) into L.

Sort L in decreasing order with radix sort.
For (i, uv) in L:

2. Insert (i, uv) to the back of Q_v.

For $v \in V$:

3. MATCHR(v).

Return M.

New runtime. Radix sorting all $O(m\varepsilon^{-1})$ pairs and initializing the sorted Q_v for all $v \in V$ takes linear time in the number of pairs. The total amount of work done in MATCHR(v) for a vertex $v \in V$ is $O(|N(v)|\varepsilon^{-1})$ which also sums to $O(m\varepsilon^{-1})$. 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 and ε be a value such that $1 > \varepsilon > 0$. There is a multiplicative auction algorithm running in time $O(m\varepsilon^{-1})$ 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 with all its incident edges sorted by weight*

in total time $O(m\varepsilon^{-1})$, 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 3 for the unmatched v. To restore this invariant we simply call MATCHR(v).
Type (2) operations: Adding a new vertex to V with all incident edges. To add a new vertex v to V with ℓ incident edges to $u_1 v, \ldots, u_\ell v$ with $w(u_1 v) > \cdots > w(u_\ell v)$, we can create the queue Q_v by inserting the $O(\varepsilon^{-1})$ 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.

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

Funding *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.

Declarations

Conflict of interest There are no conflicts of interests or competing interests.

References

1. Allen-Zhu, Z., Orecchia, L.: Nearly linear-time packing and covering LP solvers - achieving width-independence and -convergence. Math. Program. **175**(1–2), 307–353 (2019). https://doi.org/10.1007/s10107-018-1244-x
2. Assadi, S.: A simple (1-ϵ)-approximation semi-streaming algorithm for maximum (weighted) matching (2023) https://doi.org/10.48550/arXiv.2307.02968
3. Assadi, S., Liu, S.C., Tarjan, R.E.: An auction algorithm for bipartite matching in streaming and massively parallel computation models. In: Le, H.V., King, V. (eds.), 4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11–12, 2021. SIAM, pp. 165–171 (2021) https://doi.org/10.1137/1.9781611976496.18
4. Bernstein, A., Gutenberg, M.P., Saranurak, T.: Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In: Irani, S. (ed.) 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16–19, 2020. IEEE, pp. 1123–1134 (2020) https://doi.org/10.1109/FOCS46700.2020.00108
5. Bernstein, A., Dudeja, A., Langley, Z.: A framework for dynamic matching in weighted graphs. In: Khuller, S., Williams, V.V. (eds.) STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21–25, 2021. ACM, pp. 668–681 (2021). https://doi.org/10.1145/3406325.3451113
6. Bertsekas, D.P.: A new algorithm for the assignment problem. Math. Program. **21**(1), 152–171 (1981). https://doi.org/10.1007/BF01584237
7. Bhattacharya, S., Kiss, P., Saranurak, T.: Dynamic algorithms for packing-covering lps via multiplicative weight updates (2022) https://doi.org/10.48550/arXiv.2207.07519
8. Blikstad, J., Kiss, P.: Incremental (1-ϵ)-approximate dynamic matching in o(poly(1/ϵ)) update time (2023) https://doi.org/10.48550/arXiv.2302.08432
9. Bosek, B., Leniowski, D., Sankowski, P., et al.: 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. IEEE Computer Society, pp. 384–393 (2014). https://doi.org/10.1109/FOCS.2014.48

10. Chekuri, C., Quanrud, K.: Randomized MWU for positive LPs. In: Czumaj, A. (ed.) Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7–10, 2018. SIAM, pp. 358–377, (2018) https://doi.org/10.1137/1.9781611975031.25
11. Chen, L., Kyng, R., Liu, Y.P., et al.: Maximum flow and minimum-cost flow in almost-linear time (2022) https://doi.org/10.48550/arXiv.2203.00671
12. Demange, G., Gale, D., Sotomayor, M.: Multi-item auctions. J. Polit. Econ. **94**(4), 863–872 (1986)
13. Duan, R., Pettie, S.: Linear-time approximation for maximum weight matching. J. ACM **61**(1), 1:1-1:23 (2014). https://doi.org/10.1145/2529989
14. Gupta, M., Peng, R.: Fully dynamic $(1+\epsilon)$-approximate matchings. In: 54th Symposium on Foundations of Computer Science, FOCS. IEEE Computer Society, pp. 548–557 (2013). https://doi.org/10.1109/FOCS.2013.65
15. Hanauer, K., Henzinger, M., Schulz, C.: Recent advances in fully dynamic graph algorithms (invited talk). In: Aspnes, J., Michail, O. (eds.) 1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022, March 28–30, 2022, Virtual Conference, LIPIcs, vol 221. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, pp. 1:1–1:47 (2022). https://doi.org/10.4230/LIPIcs.SAND.2022.1
16. Henzinger, M., Krinninger, S., Nanongkai, D., et al.: 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, pp. 21–30 (2015). https://doi.org/10.1145/2746539.2746609
17. Koufogiannakis, C., Young, N.E.: A nearly linear-time PTAS for explicit fractional packing and covering linear programs. Algorithmica **70**(4), 648–674 (2014). https://doi.org/10.1007/s00453-013-9771-6
18. Kuhn, H.W.: The Hungarian method for the assignment problem. Naval Res. Logist. Quart. **2**(1–2), 83–97 (1955)
19. Le, H., Milenkovic, L., Solomon, S. et al.: Dynamic matching algorithms under vertex updates. In: Braverman, M. (ed.) 13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA, LIPIcs, vol 215. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, pp. 96:1–96:24, (2022) https://doi.org/10.4230/LIPIcs.ITCS.2022.96
20. Liu, Q.C., Ke, Y., Khuller, S.: Scalable auction algorithms for bipartite maximum matching problems (2023) https://doi.org/10.48550/arXiv.2307.08979
21. Munkres, J.: Algorithms for the assignment and transportation problems. J. Soc. Ind. Appl. Math. **5**(1), 32–38 (1957)
22. Quanrud, K.: Nearly linear time approximations for mixed packing and covering problems without data structures or randomization. In: Farach-Colton, M., Gørtz, I.L. (eds.) 3rd Symposium on Simplicity in Algorithms, SOSA 2020, Salt Lake City, UT, USA, January 6–7, 2020. SIAM, pp. 69–80 (2020) https://doi.org/10.1137/1.9781611976014.11
23. Schrijver, A., et al.: Combinatorial Optimization: Polyhedra and Efficiency, vol. 24. Springer, Berlin (2003)
24. Wang, D., Rao, S., Mahoney, M.W.: Unified acceleration method for packing and covering problems via diameter reduction. In: Chatzigiannakis, I., Mitzenmacher, M., Rabani, Y., et al. (eds.) 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11–15, 2016, Rome, Italy, LIPIcs, vol 55. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 50:1–50:13, (2016). https://doi.org/10.4230/LIPIcs.ICALP.2016.50
25. Young, N.E.: Nearly linear-time approximation schemes for mixed packing/covering and facility-location linear programs (2014) arxiv:1407.3015