Matching Composition and Efficient Weight Reduction in Dynamic Matching[*]

Aaron Bernstein[†] Jiale Chen[‡] Aditi Dudeja[§] Zachary Langley[¶] Aaron Sidford[‖]
Ta-Wei Tu[**]

Abstract

We consider the foundational problem of maintaining a $(1-\varepsilon)$-approximate maximum weight matching (MWM) in an n-node dynamic graph undergoing edge insertions and deletions. We provide a general reduction that reduces the problem on graphs with a weight range of $\text{poly}(n)$ to $\text{poly}(1/\varepsilon)$ at the cost of just an additive $\text{poly}(1/\varepsilon)$ in update time. This improves upon the prior reduction of Gupta-Peng (FOCS 2013) which reduces the problem to a weight range of $\varepsilon^{-O(1/\varepsilon)}$ with a multiplicative cost of $O(\log n)$.

When combined with a reduction of Bernstein-Dudeja-Langley (STOC 2021) this yields a reduction from dynamic $(1-\varepsilon)$-approximate MWM in bipartite graphs with a weight range of $\text{poly}(n)$ to dynamic $(1-\varepsilon)$-approximate maximum cardinality matching in bipartite graphs at the cost of a multiplicative $\text{poly}(1/\varepsilon)$ in update time, thereby resolving an open problem in [GP'13; BDL'21]. Additionally, we show that our approach is amenable to MWM problems in streaming, shared-memory work-depth, and massively parallel computation models. We also apply our techniques to obtain an efficient dynamic algorithm for rounding weighted fractional matchings in general graphs. Underlying our framework is a new structural result about MWM that we call the "matching composition lemma" and new dynamic matching subroutines that may be of independent interest.

1 Introduction

The *maximum matching problem* is foundational in graph algorithms and has numerous applications. A *matching* is a set of vertex-disjoint edges in an (undirected) graph. In *unweighted graphs*, $G = (V, E)$, the problem, known as *maximum cardinality matching (MCM)*, is to find a matching with the maximum number of edges (also known as the matching's *size*). More generally, in *weighted graphs*, $G = (V, E, w)$, where each $e \in E$ has weight $w(e) > 0$, the problem, known as *maximum weighted matching (MWM)*, is to find a matching M of maximum *weight*, i.e., $\sum_{e \in M} w(e)$. For simplicity, throughout the paper, we assume that each $w(e) \in [1, W]$ for $W = \text{poly}(n)$.

In the standard *static* or *offline* version of the maximum matching problem, it was recently shown how to compute maximum matchings in unweighted and weighted bipartite graphs in almost-linear time [23, 39] (when the edge weights are integer). Though this almost resolves the complexity of the problem in the standard static, full-memory, sequential model of computation, the complexities of the problem in alternative models of computation such as dynamic, streaming, and parallel models are yet to be determined. The problem has been studied extensively in these models and there are conditional lower bounds that rule out efficient algorithms for exact maximum matching in certain settings (see, for example, [31] for dynamic and [30] for streaming).

Consequently, there has been work on efficiently computing *approximately* maximum matchings. There is a range of approximation quality versus efficiency trade-offs studied (see e.g., [13, 14, 27, 40, 16]). We focus

---

[*]The full version of the paper can be accessed at https://arxiv.org/abs/2410.18936

[†]New York University, bernstei@gmail.com. Supported by Sloan Fellowship, Google Research Fellowship, NSF Grant 1942010, and Charles S. Baylis endowment at NYU.

[‡]Stanford University, jialec@stanford.edu. Supported by a Lawrence Tang Graduate Fellowship, a Microsoft Research Faculty Fellowship, and NSF CAREER Award CCF-1844855.

[§]University of Salzburg, aditi.dudeja@plus.ac.at. This work is supported by the Austrian Science Fund (FWF): P 32863-N. 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 947702).

[¶]Rutgers University, zach.langley@rutgers.edu.

[‖]Stanford University, sidford@stanford.edu. Supported by a Microsoft Research Faculty Fellowship, NSF CAREER Grant CCF-1844855, NSF Grant CCF-1955039, and a PayPal research award.

[**]Stanford University, taweitu@stanford.edu. Supported by a Stanford School of Engineering Fellowship, a Microsoft Research Faculty Fellowship, and NSF CAREER Award CCF-1844855.

on the gold standard of computing (multiplicative) $(1-\varepsilon)$-approximate matching, that is a matching of size or weight that is at least $(1-\varepsilon)$ times the maximum. While there are algorithms that compute $(1-\varepsilon)$-approximate maximum matchings in many computational models, there is often a significant gap between the bounds for weighted and unweighted graphs; many state-of-the-art results are limited to unweighted graphs. However, in dynamic, streaming, and parallel computational models, there is no clear indication of a fundamental separation between the computational complexity of the two cases. It is plausible that the existing gap could be due to a relative lack of techniques for working with weighted matchings.

Closing the gap between the state of the art for weighted and unweighted matchings is an important open problem. Several works have made progress on this problem by developing meta-algorithms that convert any algorithm for unweighted matching into an algorithm for weighted matching in a black-box fashion (albeit with potential loss in approximation quality and efficiency). The central focus of this paper is to provide improved meta-algorithms and new tools that more efficiently reduce weighted matching to unweighted matching. We motivate, develop, and introduce our results through the prominent dynamic matching problem (which we introduce next), though we also obtain results for streaming and parallel settings.

Dynamic Weighted Matching In the dynamic matching problem, a graph undergoes a sequence of adversarial updates, and the algorithm must (explicitly) maintain an (approximate) maximum matching in the graph after each update.[1] The goal is to minimize the update time of the algorithm, which is the time needed to process a single update. In the most general *fully dynamic* model, each update either inserts an edge into or deletes an edge from the graph. We also consider two natural, previously studied, *partially dynamic* models, including the *incremental* model, where each update can only insert an edge, and the *decremental* model, where each update can only delete an edge.

Over the past decades, meta-algorithms for reducing dynamic matching on weighted to unweighted graphs have been developed (with different approximation and update time trade-offs). The first general reduction is by Stubbs and Williams [38], who show that any dynamic α-approximate MCM algorithm can be converted to a dynamic $(1/2-\varepsilon)\alpha$-approximate MWM algorithm with (multiplicative) $\mathrm{poly}(\log n, \varepsilon^{-1})$ overhead in the update time.

More recently, the state of the art was achieved by a result of Bernstein, Dudeja, and Langley [11]. This paper reduces the approximation error for weighted matching to $(1-\varepsilon)\alpha$ in bipartite graphs and $(2/3-\varepsilon)\alpha$ in non-bipartite graphs with $\varepsilon^{-\Theta(1/\varepsilon)}\log n$ multiplicative overhead in the update time. [11] crucially relies on different general reduction of Gupta and Peng [29], which reduces weighted matching in a general (not necessarily bipartite) graph with large weights to one with small weights—concretely, from real values in $[1, W]$ to integers in $\{1, \ldots, \varepsilon^{-O(1/\varepsilon)}\}$—at the cost of an extra $(1-\varepsilon)$-approximation factor and $O(\log n)$ multiplicative overhead.

All of these reductions mentioned incur a multiplicative overhead of only $O_\varepsilon(\mathrm{poly}(\log n))$ to the update time, where we use $O_\varepsilon(\cdot)$ to hide factors depending on ε. However, the dependence of $1/\varepsilon$ in update time overhead in previous reductions for $(1-\varepsilon)$-approximate MWM [29, 11] are all exponential. Consequently, even for $\varepsilon = 1/O(\log n)$, an accuracy decaying slowly with increases in the graph's size, the algorithms may no longer achieve non-trivial update times.

Our Contribution Our first contribution is the following weighted-to-unweighted reduction in bipartite graphs, which settles the open problem of [29, 11] for bipartite graphs. Interestingly, this reduction, along with the prior reductions in [29, 38, 11], are *partially dynamic preserving*, i.e., if the input unweighted matching algorithm is incremental or decremental, then the resulting weighted matching algorithm is also incremental or decremental respectively.

Theorem 1.1. (Informal version of Theorem 5.8) *Given any dynamic $(1-\varepsilon)$-approximate MCM algorithm in n-node m-edge* ***bipartite*** *graphs with update time $\mathcal{U}(n, m, \varepsilon)$, there is a transformation which produces a dynamic $(1-O(\varepsilon))$-approximate MWM algorithm for n-node bipartite graphs with amortized update time $\mathcal{U}(\mathrm{poly}(1/\varepsilon)\cdot n, \mathrm{poly}(1/\varepsilon)\cdot m, \varepsilon)\cdot \mathrm{poly}(1/\varepsilon)$. This transformation is partially dynamic preserving. In non-bipartite graphs, the approximation ratio for weighted matching becomes $2/3 - O(\varepsilon)$. Moreover, if the unweighted algorithm is deterministic, then so is the weighted algorithm.*

[1] Some papers instead maintain a data structure that can answer queries about a maximum matching, e.g., [21]. The low-recourse transformation we propose in Section 5.4 can convert certain algorithms that maintain the matchings implicitly to ones that maintain them explicitly. For the simplicity of the statement, unless stated otherwise, the dynamic matching algorithms in the paper should maintain a matching explicitly.

Beyond improving the exponential dependence on $1/\varepsilon$ to polynomial, Theorem 1.1 also eliminates the $O(\log n)$ factors in the update-time of [11]. Therefore, Theorem 1.1 implies that in dynamic bipartite graphs, $(1-\varepsilon)$-approximate MWM shares the same complexity as $(1-\varepsilon)$-approximate MCM, up to $\text{poly}(1/\varepsilon)$ factors.

In the case of general graphs, we make substantial progress towards reducing weighted matching to unweighted matching. Indeed, a crucial ingredient of our algorithm is a reduction from large weights to small ones, which applies to non-bipartite graphs as well.

Theorem 1.2. (Informal version of Theorem 3.2) *Given any dynamic $(1-\varepsilon)$-approximate MWM algorithm in n-node m-edge general (possibly non-bipartite) graphs with weights in $[1, W]$ with update time $\mathcal{U}(n, m, W, \varepsilon)$, there is a transformation which produces a dynamic $(1 - O(\varepsilon))$-approximate MWM algorithm for n-node general graphs with amortized update time $\mathcal{U}(\text{poly}(1/\varepsilon) \cdot n, \text{poly}(1/\varepsilon) \cdot m, \text{poly}(1/\varepsilon), \varepsilon) \cdot \text{poly}(\log(1/\varepsilon)) + \text{poly}(1/\varepsilon)$. This transformation is partially dynamic preserving.*

Theorem 1.2 removes the exponential dependence on $1/\varepsilon$ in [29] and again incurs no $\log n$ factors in the update-time overhead (whereas [29] incurs $\log n$).

Additionally, our framework leads to a dynamic weighted *rounding* algorithm with a polylogarithmic dependence on W, improving over that of [22] (which depends linearly on W). Here, a weighted rounding algorithm maintains an *integral* matching supported on a dynamically changing *fractional* one while approximately preserving its weight (see Definition 5.6). This shows that dynamic *integral* matching, weighted or not, is equivalent to dynamic *fractional* matching (up to $\text{poly}(\log n, 1/\varepsilon)$ terms). For example, as discussed in [22], this leads to a decremental $(1-\varepsilon)$-approximate MWM algorithm in weighted general graphs with update time $\text{poly}(\log n, 1/\varepsilon)$. Previously, rounding algorithms with $\text{poly}(\log n, 1/\varepsilon)$ update times were only known for *unweighted graphs*, bipartite [5, 40, 15, 19] and general [22, 25].

Theorem 1.3. (Informal version of Theorem 5.6) *There is a dynamic weighted rounding algorithm with $\widetilde{O}(\text{poly}(1/\varepsilon))$ update time.*

Techniques Our key technical contribution is Theorem 1.2, which removes the $\varepsilon^{-O(1/\varepsilon)}$ factor in an analogous result of [29]. To reduce edge weights in weighted graph $G = (V, E, w)$ with vertices V, edges E, and edge weights w, both our reduction and the one in [29] define edge sets $E_i \subseteq E$, where each edge set is defined solely as a function of weights. The algorithm then computes an *arbitrary* $(1-\varepsilon)$-approximate MWM M_i in each E_i and shows that the M_i can be combined to compute an approximate MWM M for the entire graph. Consequently, the edge sets, E_i, are chosen to satisfy the following two properties:

1. Within each E_i, the ratio ρ of the maximum to minimum edge weight is small (we call this the *width* of the interval). This is the crux of our weight-reduction because a simple scaling and rounding approach yields the requested M_i using only an algorithm for integer weights in $\{1, \ldots, \lceil \rho/\varepsilon \rceil\}$.
2. $\mu_w(M_1 \cup ... \cup M_k) \geq (1-\varepsilon)\mu_w(G)$, where $\mu_w(\cdot)$ is the weight of the MWM in the input graph or edge set. This property is to ensure that the M_i can be combined to obtain a $(1-\varepsilon)$-approximate matching.

In the algorithm of [29], the intervals were disjoint; in fact, they were well-separated. This made it easy to prove property 2 above via a simple greedy combination. We show, however, that any set of disjoint intervals E_i that satisfies property 2 must have, in the worst case, width $\rho \geq \exp(1/\varepsilon)$. (See Section B for more details.)

To bypass this barrier, we allow for overlapping intervals. This forces us to use a different and more involved analysis, as the analysis of [29] crucially relied on disjointedness. At the outset, it is not clear that this is even enough. In fact, as we discuss below, it is only narrowly suffices in that our analysis crucially relies on the matchings M_i being $(1-\varepsilon)$-approximate, rather than α-approximate for some constant $\alpha < 1$.

Our key technical contribution consists of two new structural properties of weighted matching, which we call matching *composition* and *substitution* lemmas (see Lemmas 3.1 and 3.2). On a high level, these two lemmas show that as long as the weight classes E_i overlap slightly, a width of $\text{poly}(1/\varepsilon)$ is sufficient to ensure the second property above.

Using the above idea inside the framework of [11] with a simple greedy aggregation algorithm in [29] immediately yields a weaker version of Theorem 1.1 (with $\text{poly}(1/\varepsilon) \cdot O(\log n)$ multiplicative overhead and a slightly worse dependence on $1/\varepsilon$). We further optimize the greedy aggregation in [29] to remove the $\log n$ factor, improve the analysis of [11], and propose a low-recourse transformation to remove several $\text{poly}(1/\varepsilon)$ factors.

A Limitation of Our Reductions Although our results successfully remove the exponential dependence on $1/\varepsilon$ in prior work, they also have a limitation. The work of [29] reduces $\alpha(1-\varepsilon)$-approximate matching with general weights to α-approximate matching with small edges weights; crucially, it works for any $\alpha \le 1$. By contrast, our Theorem 1.2 requires $\alpha = (1-\varepsilon)$. As a result, our Theorem 1.1 also requires $\alpha = (1-\varepsilon)$, while the analogous result of [11] works for any α. Consequently, there are several applications related to smaller α, for example, $\alpha = 1/(2-\sqrt{2})$ (see [10, 18, 9]) or $\alpha = 2/3$ (see [13, 14]) that benefit from [11] but not from our reduction. Notably, there are also conditional lower bounds ruling out efficient $(1-\varepsilon)$-approximate matching algorithms in various models, specifically the fully dynamic [36] and the single-pass streaming model [34]. Nonetheless, $(1-\varepsilon)$-approximate matching is a well-studied regime, and, as we show in Section 6, our reduction improves the state of the art in multiple computational models.

Perhaps surprisingly, this limitation is inherent to the general framework discussed above. More precisely, consider a scheme which picks edge sets $E_i = \{e \mid \ell_i \le w(e) \le r_i\}$, computes an arbitrary α-approximate matching in each M_i, and then shows that $\mu_w(M_1 \cup \ldots \cup M_k) \ge (\alpha - \varepsilon) \cdot \mu_w(G)$. Our key contribution is to show that for $\alpha = (1 - O(\varepsilon))$, there exist suitable edge sets with width $\mathrm{poly}(1/\varepsilon)$. However, our sets do not necessarily work for a fixed constant $\alpha < 1$; in fact, we show that for $\alpha < 1$, there are graphs for which *any* suitable edge sets necessarily have width $\exp(1/\varepsilon)$ (as what was done in [29]). Generalizing our result to all work for all α would thus require a different approach. See Section C for more details.

Overview of the Paper In the remainder of the introduction, in Section 1.1 we give an informal overview of our results in other computational models and in Section 1.2 we discuss concrete applications. Thereafter, we state preliminaries in Section 2. In Section 3, we give a technical overview including our main structural lemma that implies our reduction framework. We then show our main technical lemma in Section 4 which helps to prove the main structural lemma, and state our framework in Section 5 in detail. We state applications of our framework in Section 6. We conclude with open problems in Section 7.

1.1 Additional Computational Models

Our Results Although the above theorems were written for dynamic models, our techniques are general and apply to a variety of different models. In Section 6 we state formal reductions in a variety of models; here, we simply state the main takeaways. We show that analogs of the results in Theorem 1.1 and Theorem 1.2 also apply to the semi-streaming, massively parallel computing (MPC) with $O(n \log n)$ space per machine, and the parallel work-depth models. We suspect they apply to other models as well, but in this paper, we focus on these four.

In the case of semi-streaming and MPC, the reduction when applied to existing algorithms, leaves the number of passes/rounds the same (up to a constant factor), but increases the space requirement by $\log n \cdot \mathrm{poly}(1/\varepsilon)$. On the other hand, in the parallel work-depth model, the work increases by a factor of $\log n \cdot \mathrm{poly}(1/\varepsilon)$, and the depth increases by an additive $\log^2 n$ factor. Despite these overheads, we can improve many of the state of the arts in these models.

Later in this section, we discuss these improvements and our contributions in more detail.

Contrast to Previous Work As in the case of dynamic algorithms, when we turn to other models our Theorem 1.1 and Theorem 1.2 achieve the same reductions as [11] and [29] respectively, except that we reduce their multiplicative overhead of $\varepsilon^{-O(1/\varepsilon)}$ to $\mathrm{poly}(1/\varepsilon)$. There is also a different reduction of Gamlath, Kale, Mitrovic, and Svensson [26], which works in both the streaming and MPC models, but not in the dynamic model. [26] has the advantage of reducing the most general case of weighted non-bipartite matching to the simplest case of unweighted bipartite matching, but it has exponential dependence on $1/\varepsilon$ and it increases the number of *passes/rounds* by a $\varepsilon^{-O(1/\varepsilon)}$ factor (which is generally considered a bigger drawback than the space increase).

Similar to the dynamic models, our reductions have a somewhat narrower range of application than those of [11] and [29] even in MPC, parallel, and streaming models because ours do not work for general approximation factors: they only reduce a $(1-\varepsilon)$-approximation to a $(1-\Theta(\varepsilon))$-approximation. There is also a second, more minor drawback, which is that our reduction in Theorem 1.2 works in a slightly narrower range of models than the corresponding reduction of [29]. For example, the reduction of [29] applies to algorithms that only maintain the approximate *size* of the maximum matching (see [18, 10]), whereas our reduction only applies to algorithms that maintain the actual matching. But for the most part, our reduction and that of [29] apply to the same set of models.

1.2 Applications In this subsection, we give an informal overview of some of the implications of our reductions. For a more formal statement of the results, we refer the reader to Section 6.

Applications to Bipartite Graphs Since our reduction from weighted to unweighted matching is black-box, it immediately improves upon the state of the art for weighted matching in a wide variety of computational models. Many of these results which achieve the state of the art were obtained by plugging existing unweighted algorithms into the reduction of [11], and hence incur a multiplicative overhead of $\varepsilon^{-O(1/\varepsilon)}$. Plugging in our Theorem 5.8 reduces the multiplicative overhead to $\mathrm{poly}(1/\varepsilon)$. In particular, our reduction obtains weighted analogs of the following *unweighted bipartite* results.

1. A fully dynamic algorithm for maintaining a $(1-\varepsilon)$-approximate MCM in $O(\sqrt{m}\,\mathrm{poly}(1/\varepsilon))$ time per update [29].

2. A decremental $(1-\varepsilon)$-approximate MCM algorithm with update time $\mathrm{poly}(\log n, \varepsilon^{-1})$ [12, 32].

3. A fully dynamic algorithm for maintaining a $(1-\varepsilon)$-approximate MCM with update time $O(\mathrm{poly}(\varepsilon^{-1}) \cdot \frac{n}{2^{\Omega(\sqrt{\log n})}})$ [36].

4. A fully dynamic *offline* algorithm for maintaining a $(1-\varepsilon)$-approximate MCM with update time $O(n^{0.58}\,\mathrm{poly}(\varepsilon^{-1}))$; in the offline model, the entire sequence is known to the algorithm in advance [36].

5. An incremental $(1-\varepsilon)$-approximate MCM algorithm with update time $\mathrm{poly}(\varepsilon^{-1})$ [20].

6. An $O(\varepsilon^{-2})$-pass, $O(n)$ space streaming algorithm for $(1-\varepsilon)$-approximate MCM [8]

7. An $O(\varepsilon^{-2} \cdot \log\log n)$-round, $O(n)$ space per machine, MPC algorithm for $(1-\varepsilon)$-approximate MCM [8].

Our reduction extends all of the above results to work in weighted graphs: the multiplicative overhead is only $\mathrm{poly}(1/\varepsilon)$ in the dynamic model, as well as a $O(\log n)$ factor in some of the other models. Before our work, the weighted versions of 1, 3, and 4 had a multiplicative overhead of $\varepsilon^{-O(1/\varepsilon)} \cdot \log W$. For others mentioned on the list, separate weighted versions were known (see [17, 35]), but had worse dependence on either $\log n$ factors or ε^{-1} factors. Our main contribution here is to remove these overheads and, equally importantly, to streamline existing research by removing the need for a separate weighted algorithm.

Note that there are additional results on streaming approximate matching algorithm which obtain improved pass dependencies on ε at the cost of $\mathrm{poly}(\log n)$ factors [1, 3, 7, 6]. Our reduction does not improve the state of the art here for such methods. Therefore, we focus on algorithms that have pass complexities that only depend on ε.

Applications to Non-Bipartite Graphs Similar to Theorem 5.8, our aspect-ratio reduction in Theorem 3.2 also works as a black-box. Each of the results below was initially obtained by applying the reduction of [29] to a weighted matching algorithm with a large dependence on W. By plugging in our reduction, we reduce the ε-dependence in all of them from $\varepsilon^{-O(1/\varepsilon)}$ to $\mathrm{poly}(1/\varepsilon)$.

1. A fully dynamic algorithm for maintaining a $(1-\varepsilon)$-approximate MWM in general graphs in $\sqrt{m} \cdot \varepsilon^{-O(1/\varepsilon)} \cdot \log W$ time [29].

2. A decremental algorithm for maintaining a $(1-\varepsilon)$-approximate MWM in general graphs in $\mathrm{poly}(\log n) \cdot \varepsilon^{-O(1/\varepsilon)}$ update time [22].

3. A $\mathrm{poly}(\log n) \cdot \varepsilon^{-O(1/\varepsilon)}$ update time algorithm for rounding $(1-\varepsilon)$-approximate weighted fractional matchings in general graphs to $(1-\Theta(\varepsilon))$-approximate integral matchings [22].

2 Preliminaries

General Notation For positive integer k, we let $[k] \stackrel{\text{def}}{=} \{1, \ldots, k\}$. For sets S and T, we let $S \oplus T \stackrel{\text{def}}{=} (S \setminus T) \cup (T \setminus S)$ denote their symmetric difference.

Graphs and Matchings Throughout this work, $G = (V, E)$ denotes an undirected graph, and $w : E \to \mathbb{R}_{>0}$ is an edge weight function. The *weight ratio* of G is $\max_e w(e)/\min_f w(f)$. A *matching* $M \subseteq E$ is a set of vertex-disjoint edges. The weight of a matching M, denoted $w(M)$, is the sum of the weights of the edges in the matching: $w(M) \stackrel{\text{def}}{=} \sum_{e \in M} w(e)$. We denote the maximum value of $w(M)$ achieved by any matching M on G by $\mu_w(G)$. For $\alpha \in [0, 1]$, an α-approximate MWM of G is a matching M such that $w(M) \geq \alpha \cdot \mu_w(G)$. The following result states that we can compute an $(1 - \varepsilon)$-approximate MWM very efficiently.

THEOREM 2.1. ([24, THEOREM 3.12]) *On an m-edge general weighted graph, a $(1 - \varepsilon)$-approximate MWM can be computed in time $O(m \log(\varepsilon^{-1})\varepsilon^{-1})$.*

Weight Intervals For $I \subseteq \mathbb{R}$, we denote by G_I the subgraph of G restricted to edges e such that $w(e) \in I$. A set of (disjoint) weight intervals $[\ell_1, r_1), \dots, [\ell_k, r_k) \subseteq \mathbb{R}$ has *weight gap* δ if $\ell_{i+1} \geq \delta \cdot r_i$ for all $i \in [k - 1]$ and we call such a set of weight classes *δ-spread*. We also say that the set of intervals is *δ-wide* if $r_i \geq \delta \cdot \ell_i$ for all $i \in [k]$. If $\ell_{i+1} = r_i$ and the intervals cover $[1, W]$, then we say the intervals are a *weight partition*.

Computational Model We work in the standard Word-RAM model in which arithmetic operations over $\Theta(\log n)$-bit words can be performed in constant time.

3 Technical Overview

Here we give an overview of our framework. For comparison and motivation, in Section 3.1 we first introduce a result of [29], which provides a deterministic framework that is partially dynamic preserving for dynamic approximate MWM algorithms to reduce the weight range to $\varepsilon^{-O(1/\varepsilon)}$ with an overhead of $O(\log W)$. In Section 3.2, we explain the difficulty of reducing to $\text{poly}(1/\varepsilon)$ weight range using [29]. Motivated by this, in Section 3.3, we introduce our key technical innovation, the matching composition lemma (Lemma 3.1), that allows us to bypass the barrier. We then show that this lemma naturally induces our algorithmic framework that reduces the weight range down to $\text{poly}(1/\varepsilon)$. Finally, in Section 3.4, we overview several further improvements that we made to shave off $\log n$ and $1/\varepsilon$ factors from the running time.

3.1 Weight Reduction Framework of Gupta–Peng The reduction framework of [29] works as follows. First, it groups edges geometrically by their weights so that the weight ratio of each group is $\Theta(1/\varepsilon)$. It then deletes one group for every $\Theta(1/\varepsilon)$ consecutive groups and merges the remaining consecutive groups. We refer to the merged groups as *weight classes*; note that they are $\Theta(1/\varepsilon)$-spread and have weight ratio $\varepsilon^{-\Theta(1/\varepsilon)}$. For each of these weight classes, a $(1 - \varepsilon)$-approximate MWM is maintained. Because the weight classes are $\Theta(1/\varepsilon)$-spread, a simple greedy aggregation [4, 29, 38] of the $(1 - \varepsilon)$-approximate MWMs on each weight class leads to a $(1 - O(\varepsilon))$-approximation of $\mu_w(G)$. As a result, this reduction reduces general approximate MWM to the problem of maintaining an approximate MWM inside a weight class with weight ratio $\varepsilon^{-\Theta(1/\varepsilon)}$.

More formally, the algorithm of [29] assigns all edges e with weight $w_e \in [\varepsilon^{-i}, \varepsilon^{-(i+1)})$ to the ith group. Let $G^{(j)}$ denote the graph obtained by deleting all groups i such that $i \equiv j \bmod \lceil \varepsilon^{-1} \rceil$. An averaging argument shows that $\max_j \mu_w(G^{(j)}) \geq (1 - O(\varepsilon))\mu_w(G)$. So it suffices to maintain a $(1 - O(\varepsilon))$-approximate MWM on each $G^{(j)}$ and return the one with maximum weight.

To do so, [29] merges all groups between neighboring deletions to form weight classes in $G^{(j)}$. Those weight classes have weight ratio $\varepsilon^{-\Theta(1/\varepsilon)}$ and are $\Theta(1/\varepsilon)$-spread. [29] proceed by maintaining a $(1-\varepsilon)$-approximate MWM $M_k^{(j)}$ on each weight class $[\ell_k^{(j)}, r_k^{(j)})$; by scaling down appropriately, maintaining each $M_k^{(j)}$ requires maintaining a $(1 - \varepsilon)$-approximate MWM in a graph with edges in range $[1, \varepsilon^{-\Theta(1/\varepsilon)}]$, as desired. The authors of [29] then greedily aggregate the M_i into a single matching M by checking the M_i in descending order of weight range and including in M any edge that is not adjacent to existing edges in M. The $\Theta(1/\varepsilon)$ weight gap between weight classes ensures that for each edge e in the final matching M, the total weight of edges in $\bigcup M_i$ that were not included in M because of e is at most $O(\varepsilon) \cdot w_e$. Since $(1 - O(\varepsilon))\mu_w(G) \leq (1 - O(\varepsilon))\mu_w(G^{(j)}) \leq \sum_i w(M_i)$, the greedy aggregation keeps a $1 - O(\varepsilon)$ fraction of the weight in $\sum_i w(M_i)$ thus is a $(1 - O(\varepsilon))$-approximate MWM.

3.2 Disjoint Weight Classes Require Exponential Width The $\Theta(1/\varepsilon)$ weight gap plays an important role in [29] because it ensures that $\mu_w(\bigcup M_i) \geq (1 - O(\varepsilon))\mu_w(G)$, while also allowing for efficient greedy aggregation. But as long as we try to maintain weight classes that are $1/\varepsilon$-spread, it seems hard to reduce the weight ratio all the way down to $\text{poly}(1/\varepsilon)$. This is because it would require deleting a constant fraction of the initial weight groups (the ones of weight ratio $\Theta(1/\varepsilon)$), so the MWM on the remaining graph $G^{(j)}$ would have a much smaller weight than $\mu_w(G)$. Indeed, we rule out the possibility of a broader family of methods that works with non-overlapping weight classes (which includes all methods that create weight gaps) by answering the following question in the negative.

QUESTION 3.1. *For any graph G, is there a weight partition $[\ell_1, r_1), \ldots, [\ell_k, r_k)$ such that $r_i/\ell_i \leq \text{poly}(1/\varepsilon)$ holds for all i and given any set of $(1-\varepsilon)$-approximate MWM M_i on each $G_{[\ell_i, r_i)}$ we have*

$$\mu_w(M_1 \cup M_2 \cup \cdots \cup M_k) \geq (1 - O(\varepsilon)) \cdot \mu_w(G)?$$

To see why methods creating $\Theta(1/\varepsilon)$ weight gaps are a special case of the weight partition allowed in Question 3.1, note that given any partition with gaps, we can naturally define a weight partition by letting each weight gap be its own weight class; if a large matching exists after deleting the edges in the gaps, it still exists when we keep those edges.

We give a counterexample (see Claim B.1) that answers Question 3.1 in the negative, even when the weight partition can be chosen depending on the input graph (recall that [29] chose the weight partition up front, oblivious to the structure of the input graph). To see why this is the case, consider the gadget shown in Figure 1 below. Fix a partition $\mathcal{P}$ of $[1, W]$ into weight classes. Observe that if this partition "separates" the gadget, i.e., some weight class i in $\mathcal{P}$ contains only the weight 1 but not 1.5 (and the other weight class j contains 1.5), then it leads to an overall loss larger than $(1-\varepsilon)$. More concretely, if the matching M_i in class i restricted to the gadget contains the edge bc (and not ab), then $\text{MWM}(M_i \cup M_j) = 1.5$ while the entire gadget contains a matching $\{ab, cd\}$ of weight 2.5. The final counterexample we construct then contains multiple copies of the gadget with different weights and argues that any weight partition $\mathcal{P}$ with $r_i/\ell_i \leq \text{poly}(1/\varepsilon)$ for all i must "separate" sufficiently many gadgets. Consequently, it cannot preserve $(1-\varepsilon)$-approximation.

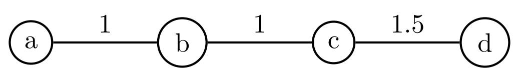

Figure 1: Gadget for answering Question 3.1 in the negative

3.3 Leveraging Weight Overlaps: the Matching Composition Lemma How can we bypass this barrier? Let us take a closer look at Figure 1. In the gadget above, there are two possible M_i for the weight class i that contains the weight 1: either it contains the edge ab or bc. As we have discussed above, the "bad" case is when M_i contains bc instead of ab, in which case the edge bc will be "kicked out" by the edge cd in M_j and results in a weight loss. Notice also that the weight loss affects the final approximation ratio *when the weights of bc and cd are relatively close* (as depicted in Figure 1)—if instead of 1.5, the weight of cd is changed to at least $1/\varepsilon$, then the weights of ab and bc are negligible compared to cd (up to an ε fraction), and it is okay if somehow M_i contains bc and it is "kicked out" by cd. Therefore, to fix the issue, for any two weight values w_1 and w_2 that are close enough (in particular, $\varepsilon \lesssim w_1/w_2 \lesssim 1/\varepsilon$), we should have a weight class that contains both of them. On the other hand, it is fine for w_1 and w_2 to not be in any weight class together if they are far apart.

Based on the observation, we see that if the weight classes are disjoint, then there will always be two close-enough weight values that are separated by the partition. As a result, instead of creating weight gaps, we should leverage *overlaps* between adjacent weight classes. More concretely, we compute the approximate MWM for each weight class based on the information within the class and *also* edges of neighboring weight classes. In other words, we enlarge the intervals in which we compute the matchings slightly. Perhaps surprisingly, we show that having an overlap of $\text{poly}(1/\varepsilon)$ allows us to bypass the above barrier completely, which we prove the following key technical lemma.

LEMMA 3.1. (MATCHING COMPOSITION LEMMA) *Let $\varepsilon \leq 1/6$ and G be a graph, and consider a δ-wide weight partition $[\ell_1, r_1), [\ell_2, r_2), \ldots, [\ell_k, r_k)$. If M_i is an arbitrary $(1-\varepsilon)$-approximate MWM on $G_{[\varepsilon\ell_i, r_i\varepsilon^{-1})}$ for all $i \in [k]$, then*

$$\mu_w(M_1 \cup M_2 \cup \cdots \cup M_k) \geq (1 - O(\varepsilon \cdot \log_\delta(1/\varepsilon))) \cdot \mu_w(G).$$

On a high level, the matching composition lemma states that if we "pad" the weight classes $[\ell_i, r_i)$ slightly by a factor of $1/\varepsilon$ in both directions, causing overlap, then we can effectively "sparsify" the graph by only considering MWM's on each $G_{[\varepsilon\ell_i, r_i\varepsilon^{-1})}$. This readily leads to an algorithmic framework for weight reduction: fix a δ-wide weight partition of the graph, maintain a $(1-\varepsilon)$-approximate MWM M_i on each "padded" weight class using the given dynamic algorithm, and then somehow aggregate them together to form the final output matching (i.e., maintain a $(1-\varepsilon)$-approximate MWM on the union of M_i's). As long as the aggregation can be done efficiently, the output matching can be as well. We give a more detailed overview in Section 3.3.1.

The Matching Substitution Lemma The moment we introduce weight overlaps, we need a completely different analysis from that of [29] to prove that $\mu_w(\bigcup M_i) \geq (1-O(\varepsilon))\mu_w(G)$. By forcing disjoint (and in fact spread) weighted classes, [29] ensured that any conflict between matchings M_i and M_j could always be resolved in favor of the higher weight class (hence greedy aggregation). But once there is weight overlap, it is not clear how conflicts should be resolved. Our new analysis thus requires a new structural understanding of weighted matching.

In particular, the matching composition lemma is proved via a structural *matching substitution lemma* formally stated below. It asserts that one can effectively "substitute" parts of a matching S with matchings $T_1, \ldots, T_k$ that come from certain weight classes.

Lemma 3.2. (Matching Substitution Lemma) *Let G be a graph and $[\ell_1, r_1), \ldots, [\ell_k, r_k) \subseteq \mathbb{R}$ be $(1/\varepsilon)$-spread. For $\varepsilon \leq 1/2$, given any matching $S \subseteq G$, and a batch of target matchings $\{T_i \subseteq G_{[\varepsilon\ell_i, r_i\varepsilon^{-1})} \mid i \in [k]\}$, there exists a matching $M \subseteq S \cup T_1 \cup \cdots \cup T_k$ of weight*

$$w(M) \geq (1-4\varepsilon)w(S) - \sum_{i\in[k]} \left(\mu_w(G_{[\varepsilon\ell_i, r_i\varepsilon^{-1})}) - w(T_i)\right)$$

such that $M \cap G_{[\ell_i, r_i)} \subseteq T_i$ for all $i \in [k]$.

The matching substitution lemma starts with an arbitrary source matching S, and a set of target matchings $T_1, \ldots, T_k$ on the "padded" weight classes $[\varepsilon\ell_1, r_1/\varepsilon), \ldots, [\varepsilon\ell_k, r_k/\varepsilon)$. It allows us to build a matching starting from S and substitute all edges of S in each weight class $[\ell_i, r_i)$ with edges contained T_i; this incurs some additive approximation error, but it is easy to check that the error is small as long as each T_i is a near-maximum matching for the corresponding padded weight class. For example, if we take S to be a maximum weight matching on G and set T_i to be the $(1-\varepsilon)$-approximate MWM M_i on $G_{[\varepsilon\ell_i, r_i\varepsilon^{-1})}$ from Lemma 3.1, then we can substitute each weight range of S with edges from M_i at minimal loss. We defer the full proof to Section 4.

Arbitrary Approximation Ratio As discussed in the introduction, our reductions only work for $(1-\varepsilon)$-approximations, and not for arbitrary α-approximations. In particular, in the matching composition lemma, if each M_i is instead an arbitrary α-approximate MWM (for some fixed $\alpha < 1$), then it is ***not*** the case that $\mu_w(M_1 \cup \ldots \cup M_k) \geq (\alpha - O(\varepsilon))\mu_w(G)$. Somewhat surprisingly, this limitation is not an artifact of our particular choice of weight classes, and turns out to be inherent to the general framework of composing approximate matchings between weight classes: for such a framework to work with any α-approxmation (as does [29]), $\exp(1/\varepsilon)$-wide weight classes are required. See Section C for more details.

3.3.1 Algorithmic Framework The matching composition lemma suggests the following algorithmic framework:

1. Fix a δ-wide weight partition of the input graph and maintain a $(1-\varepsilon)$-approximate MWM M_i on each padded weight class.

2. Maintain a $(1-O(\varepsilon))$-approximate MWM on the union of M_i's as the output matching. By the matching composition lemma this is $(1-O(\varepsilon\log_\delta(1/\varepsilon)))$-approximate in the input graph.

Scaling ε down by a factor of $O(\log_\delta(1/\varepsilon))$ thus ensures that the matching we output is $(1-\varepsilon)$-approximate in the input graph. We now describe how we implement Step 2 efficiently. For this we set $\delta = \Theta(\varepsilon^{-3})$. With this choice of δ, even though the neighboring "padded" weight classes overlap, the sets of "odd" and "even" intervals are still each $\Theta(1/\varepsilon)$-spread.

(2.1) As such, similar to [29], these matchings can be separately aggregated using a greedy algorithm in $O(\log n)$ update time. More specifically, let M_i be the matching in the i-th weight class. Then, we compute a $(1-\varepsilon)$-approximate MWM M_{odd} (respectively, M_{even}) on the union $M_1 \cup M_3 \cup \cdots$ (respectively, $M_2 \cup M_4 \cup \cdots$) greedily.

(2.2) At this point, we are left with two matchings M_{odd} and M_{even} that we need to combine together. This can be relatively easily handled in $O(1/\varepsilon)$ time per change to M_{odd} and M_{even} since the union of these two matchings consists of only paths and cycles, and MWM can be computed and maintained very efficiently on them by splitting each connected component into paths of length $O(1/\varepsilon)$ and solving each path individually via a dynamic program.

As a result, with this choice of δ we arrive at a deterministic framework that reduces the aspect ratio from $\text{poly}(n)$ to $\Theta(\varepsilon^{-5})$ for any dynamic algorithm (note that the $\Theta(\varepsilon^{-5})$ term comes from padding the $\Theta(\varepsilon^{-3})$-wide intervals in each direction).

THEOREM 3.1. *Given a dynamic $(1-\varepsilon)$-approximate MWM algorithm in general (possibly non-bipartite) graphs with maximum weight in $[1, \text{poly}(1/\varepsilon)]$, there is a transformation that produces a dynamic $(1-O(\varepsilon))$-approximate MWM algorithm in graphs with maximum weight $[1, W]$. The reduction is partially dynamic preserving and has a multiplicative update time overhead of $\log n \cdot \text{poly}(1/\varepsilon)$. The new weighted algorithm is deterministic if the initial algorithm is deterministic.*

The weight reduction framework described above works for both bipartite and non-bipartite graphs. Moreover, combined with the unfolding framework of [11], it reduces weighted matching algorithms directly to unweighted matching algorithms in bipartite graphs with $\log n \cdot \text{poly}(1/\varepsilon)$ multiplicative overhead.

3.4 Further Improvements On top of the framework Theorem 3.1, we made the following additional improvements to decrease $\log n$ and $1/\varepsilon$ factors in the final running time which may be of independent interest.

More Efficient Aggregation of Spread Matchings In Step (2.1) of our framework described above, we need to maintain a $(1-\varepsilon)$-approximate MWM over matchings $M_1, \ldots, M_k$ whose weights are sufficiently spread apart (by a gap of $\Theta(1/\varepsilon)$).

PROBLEM 5.1. ($(1-\varepsilon)$-APPROXIMATE MWM OVER MATCHINGS IN $(1/\varepsilon)$-SPREAD WEIGHT CLASSES) *Given a set of $(1/\varepsilon)$-spread weight classes $[\ell_1, r_1), \ldots, [\ell_k, r_k) \subseteq \mathbb{R}$, and a set of k matchings $M_1, \ldots, M_k \subseteq G$ undergoing adversarial edge deletions/insertions satisfying $M_i \subseteq G_{[\ell_i, r_i)}$ for all $i \in [k]$. The task is to dynamically maintain a matching M satisfying*

$$w(M) \geq (1 - O(\varepsilon)) \sum_{i \in [k]} w(M_i).$$

The work of [29] solved Problem 5.1 with update time $O(k)$ using a *greedy census matching* algorithm that was also used in [4, 38]. To improve upon this, we propose a different notion of *locally* greedy census matching. We show that the new notion suffices for maintaining a $(1-\varepsilon)$-approximation and since, on a high level, the local version allows us to consider fewer edges in each update, we get a faster update time of $O(k/\log n)$. Note that for Step (2.1), the value of k is[2] $O(\log n)$ and thus this shaves off the $O(\log n)$ factor in the update time that would have been there if we used the subroutine of [29]. See Section 5.1 for more details.

Low-Recourse Transformation Note that the overall update time of our framework also depends on the *recourse* σ of the given dynamic algorithm $\mathcal{A}$, i.e., the number of changes to the matching M_i that it generates per update to the input graph. This is because each such changes propagate to the internal dynamic subroutines, and for our case it will first correspond to an update to our algorithm for Problem 5.1, and then be propagated to Step (2.2) which has an update time of $O(1/\varepsilon)$. Our overall update time is thus $\mathcal{U} + \sigma/\varepsilon$, where $\mathcal{U}$ is the update time of the dynamic algorithm $\mathcal{A}$. Similar scenarios also occur in previous reductions of [29, 11], and they both implicitly used the fact that $\sigma \leq \mathcal{U}$ (this is for algorithms that explicitly maintain a matching) and therefore their reductions incur a *multiplicative* overhead in the update time of $\mathcal{A}$.

[2] Note that this is because we assume the input graph has weights in $[1, \ldots, \text{poly}(n)]$.

However, the output recourse can be much smaller than the update time. For instance, for unweighted dynamic matching algorithms, the recourse can always be made $O(1/\varepsilon)$ by a simple lazy update trick (the work of [37] further achieved a *worst-case* recourse bound by "smoothing" the lazy update), while all known dynamic matching algorithms have update time much larger than this. To address this disparity and make the overhead of our reduction *additive*, we design a generic low-recourse transformation that converts, in a black-box way, *any* $(1-\varepsilon)$-approximate dynamic MWM algorithm to one with amortized recourse $O(\text{poly}(\log W)/\varepsilon)$. This improves the naïve lazy update approach that would have a recourse bound of $O(W/\varepsilon)$. As our framework reduces the weight range to $W = \Theta(\varepsilon^{-5})$, we use this new low-recourse transformation on the input algorithm $\mathcal{A}$ to decrease the additive overhead from $O(\varepsilon^{-6})$ (with the naïve lazy update) to $O(\log(\varepsilon^{-1})/\varepsilon)$.

To improve the aspect ratio further than $\Theta(\varepsilon^{-5})$, we continue to apply the framework on each $\Theta(\varepsilon^{-5})$ intervals. Combined with the low-recourse transformation, we provide a trade-off between the aspect ratio and the efficiency of aggregation (see Corollary 5.1). We use it to improve the fully dynamic low-degree algorithm in [29] which then serves as another aggregation method that finally allows us to reduce the aspect ratio to $\Theta(\varepsilon^{-2})$, the best we can get using Lemma 3.1. See Section 5 for more details.

The Final Transformation In the end, applying the improvements we discussed in this section, we obtain our final main theorem.

Theorem 3.2. *Given a dynamic $(1-\varepsilon)$-approximate MWM algorithm $\mathcal{A}$ that, on input n-vertex m-edge graph with aspect ratio W, has initialization time $\mathcal{I}(n,m,W,\varepsilon)$, and update time $\mathcal{U}(n,m,W,\varepsilon)$, there is a transformation which produces a dynamic $(1-\varepsilon\log(\varepsilon^{-1}))$-approximate MWM dynamic algorithm that has initialization time*

$$\log(\varepsilon^{-1}) \cdot O(\mathcal{I}(n,m,\Theta(\varepsilon^{-2}),\Theta(\varepsilon)) + m\varepsilon^{-1}),$$

amortized update time

$$\text{poly}(\log(\varepsilon^{-1})) \cdot O(\mathcal{U}(n,m,\Theta(\varepsilon^{-2}),\Theta(\varepsilon)) + \varepsilon^{-5}),$$

and amortized recourse

$$\text{poly}(\log(\varepsilon^{-1})) \cdot O(\varepsilon^{-5}).$$

The transformation is partially dynamic preserving.

4 Matching Composition and Substitution Lemmas

We now turn to the proof of our key technical lemmas, the matching composition and substitution lemmas. We first prove the matching substitution lemma, and then use it to deduce the matching composition lemma that ultimately leads to our algorithmic framework.

In the matching substitution lemma, we are given a source matching S and target matchings $T_1,\dots,T_k$ on padded versions of $(1/\varepsilon)$-spread intervals $[\ell_1,r_1),\dots,[\ell_k,r_k)$; more precisely, each T_i is a matching on $[\varepsilon\ell_i, r_i\varepsilon^{-1})$. The lemma states that we can find a new matching M with $w(M)\approx w(S)$—assuming the matchings T_i are large—such that M only uses edges of T_i on the weight interval $[\ell_i,r_i)$ for each $i\in[k]$. The key idea in the proof is to identify a set of edges D with small total weight (relative to S) to delete such that the edges of every component in $(S\cup T_1\cup\dots\cup T_k)\setminus D$ are confined to a single weight class.

Lemma 3.2. (Matching Substitution Lemma) *Let G be a graph and $[\ell_1,r_1),\dots,[\ell_k,r_k)\subseteq\mathbb{R}$ be $(1/\varepsilon)$-spread. For $\varepsilon\le 1/2$, given any matching $S\subseteq G$, and a batch of target matchings $\{T_i\subseteq G_{[\varepsilon\ell_i,r_i\varepsilon^{-1})}\mid i\in[k]\}$, there exists a matching $M\subseteq S\cup T_1\cup\dots\cup T_k$ of weight*

$$w(M)\ge(1-4\varepsilon)w(S)-\sum_{i\in[k]}\left(\mu_w(G_{[\varepsilon\ell_i,r_i\varepsilon^{-1})})-w(T_i)\right)$$

such that $M\cap G_{[\ell_i,r_i)}\subseteq T_i$ for all $i\in[k]$.

Proof. We construct a sequence of matchings $M_0,M_1,\dots,M_k$, such that $M_0=S$ is the source matching, M_i is constructed from $M_{i-1}\cup T_i$, and $M_k=M$ is the desired matching in the lemma.

We first describe how to construct M_i for $i\ge1$. The components of $M_{i-1}\oplus T_i$ are only paths and cycles. Construct a set $D_i\subseteq M_{i-1}$ as follows: For each path or cycle $P\subseteq M_{i-1}\oplus T_i$ and $e\in P\cap M_{i-1}$ such that

$w(e) \geq r_i\varepsilon^{-1}$, in both directions of P, add the closest edges in $P \cap M_{i-1}$ of weight at most r_i into D_i. For each $e \in P \cap M_{i-1}$ such that $\ell_i \leq w(e) < r_i\varepsilon^{-1}$, in both directions of P, add the closest edges in $P \cap M_{i-1}$ of weight less than $\varepsilon\ell_i$ into D_i.

Now let $\widetilde{M}_{i-1} \overset{\text{def}}{=} M_{i-1} \setminus D_i$ and again consider $\widetilde{M}_{i-1} \oplus T_i$ and any path or cycle $P \subseteq \widetilde{M}_{i-1} \oplus T_i$. Notice that if there is an $e \in P \cap \widetilde{M}_{i-1}$ such that $w(e) \in [\ell_i, r_i)$, then it must be the case that $P \subseteq G_{[\varepsilon\ell_i, r_i\varepsilon^{-1})}$. Let L_i be the collection of such paths and cycles. We then construct $M_i \overset{\text{def}}{=} \widetilde{M}_{i-1} \oplus L_i$, and thus $M_i \subseteq \widetilde{M}_{i-1} \cup T_i$ and $M_i \cap G_{[\ell_i, r_i)} \subseteq T_i$. It follows by induction that $M_k \subseteq M_{k-1} \cup T_k \subseteq \cdots \subseteq S \cup T_1 \cup \cdots \cup T_k$ and $M_k \cap G_{[\ell_i, r_i)} \subseteq T_i$ for all i.

We now analyze $w(M_k)$. Starting with M_0, two kinds of changes happened to the matching. The first one is the edge deletion $D_1 \cup \cdots \cup D_k$, and the second one is the edge substitution through $L_1 \cup \cdots \cup L_k$. We analyze the total weight loss in each part respectively.

1. Since $\ell_i \geq r_{i-1}\varepsilon^{-1}$, only edges in S cause deletion. For any edge $e \in S$, it could cause at most 2 edges deletions with respect to every weight class $[\varepsilon\ell_i, r_i\varepsilon^{-1})$. The weight of the deleted edges in the ith weight class would be at least ε smaller than w_e and at most r_i. Since $r_i \geq \ell_i \geq r_{i-1}\varepsilon^{-1}$, the total weight of those deleted edges would be at most $w(e) \cdot \left(2\varepsilon + 2\varepsilon^2 + \cdots\right) \leq 4\varepsilon \cdot w(e)$. Thus
$$w(D_1 \cup \cdots \cup D_k) \leq 4\varepsilon \cdot w(M).$$

2. For each of the substitution induced by L_i, notice that $L_i \subseteq G_{[\varepsilon\ell_i, r_i\varepsilon^{-1})}$, thus
$$\sum_{P \in L_i} w(P \cap \widetilde{M}_{i-1}) - w(P \cap T_i) \leq \mu_w(G_{[\varepsilon\ell_i, r_i\varepsilon^{-1})}) - w(T_i).$$
Therefore, the second part leads to a total weight loss of at most
$$\sum_{i \in [k]} \left(\mu_w(G_{[\varepsilon\ell_i, r_i\varepsilon^{-1})}) - w(T_i)\right).$$

□

We also need the following helper lemma.

Lemma 4.1. *For $\varepsilon \leq 1/6$, any graph G and any set of $(1/\varepsilon)$-spread weight classes $[\ell_1, r_1), \ldots, [\ell_k, r_k) \subseteq \mathbb{R}$, we have*
$$\sum_{i \in [k]} \mu_w(G_{[\ell_i, r_i)}) \leq (1 + 4\varepsilon) \cdot \mu_w(G).$$

Proof. Suppose that $\ell_1 < \ell_2 < \cdots < \ell_k$. Let M_i be a MWM on $G_{[\ell_i, r_i)}$, and let $H \overset{\text{def}}{=} M_1 \cup \cdots \cup M_k$. Let M be the matching obtained by the following greedy process: While H is non-empty, we pick an edge e in H with the maximum weight and include it into M. Then, to ensure that the next edge we pick from H still forms a matching with M, we remove all edges in H that are adjacent to e (including e itself). Observe that if an edge f is removed from H by e, then we must have $w(f) \leq w(e)$. Let i_e be such that $e \in M_{i_e}$. We also have that for each $j < i_e$, at most two edges from M_j will be removed by e (the two matched edges in M_j for the endpoints of e). As the weight classes are $(1/\varepsilon)$-spread, we have
$$\sum_{f \text{ removed by } e} w(f) \leq w(e) \cdot (1 + 2 \cdot (\varepsilon + \varepsilon^2 + \cdots)) \leq (1 + 4\varepsilon) \cdot w(e).$$
At the end of the process, H will become empty. In other words, each edge in H is removed by some edge in M. This shows that
$$\sum_{i \in [k]} \mu_w(G_{[\ell_i, r_i)}) = \sum_{f \in H} w(f) \leq (1 + 4\varepsilon) \cdot w(M) \leq (1 + 4\varepsilon) \cdot \mu_w(G).$$

□

We can now prove the matching composition lemma.

Lemma 3.1. (Matching Composition Lemma) *Let $\varepsilon \le 1/6$ and G be a graph, and consider a δ-wide weight partition $[\ell_1, r_1), [\ell_2, r_2), \dots, [\ell_k, r_k)$. If M_i is an arbitrary $(1-\varepsilon)$-approximate MWM on $G_{[\varepsilon \ell_i, r_i \varepsilon^{-1})}$ for all $i \in [k]$, then*

$$\mu_w(M_1 \cup M_2 \cup \cdots \cup M_k) \ge (1 - O(\varepsilon \cdot \log_\delta(1/\varepsilon))) \cdot \mu_w(G).$$

Proof. Let $g = \lceil \log_\delta(1/\varepsilon^3) \rceil + 1$, and for all $j \in \{0, \dots, g-1\}$, let $I_j = \{i \in [k] : i \equiv j \bmod g\}$. For the weight classes in each I_j, the weight gap between neighboring weight classes is at least $\delta^{g-1} \ge 1/\varepsilon^3$. The set of weight classes $\{[\ell_i, r_i) : i \in I_j\}$ is $(1/\varepsilon^3)$-spread, and thus the set of padded weight classes $\{[\varepsilon \ell_i, r_i \varepsilon^{-1}) : i \in I_j\}$ is $(1/\varepsilon)$-spread. Consider any exact MWM M^* on G. We will start with the initial source matching $S_0 = M^*$, and for $j = 0, 1, \dots, g-1$, sequentially apply Lemma 3.2 on the source matching S_j and target matchings $\{M_i \mid i \in I_j\}$ and get S_{j+1}. For a fixed j, since M_i is a $(1-\varepsilon)$-approximate MWM on $G_{[\varepsilon \ell_i, r_i \varepsilon^{-1})}$ for $i \in I_j$, Lemmas 3.2 and 4.1 give us a matching $S_{j+1} \subseteq S_j \cup (\bigcup_{i \in I_j} M_i)$ that satisfies

$$\begin{aligned} w(S_{j+1}) &\ge (1-4\varepsilon) w(S_j) - \varepsilon \cdot \sum_{i \in I_j} \mu_w(G_{[\varepsilon \ell_i, r_i \varepsilon^{-1})}) \\ &\ge (1-4\varepsilon) w(S_j) - \varepsilon(1+4\varepsilon) \cdot \mu_w(G) \ge (1-4\varepsilon) w(S_j) - 3\varepsilon \cdot \mu_w(G), \end{aligned}$$

and that $S_{j+1} \cap G_{[\ell_i, r_i)} \subseteq M_i$ for all $i \in I_j$. By induction, we have

$$w(S_{j+1}) \ge (1 - 4(j+1)\varepsilon) \cdot w(S_0) - 3(j+1)\varepsilon \cdot \mu_w(G) \ge (1 - 7(j+1)\varepsilon)\mu_w(G),$$

and that

$$S_{j+1} \cap G_{[\ell_i, r_i)} \subseteq \left(S_j \cap G_{[\ell_i, r_i)}\right) \cup \left(\bigcup_{t \in I_j} M_t \right) \subseteq \cdots \subseteq \bigcup_{l=j'}^{j} \bigcup_{t \in I_l} M_t$$

hold for all $j' \le j$ and $i \in I_{j'}$. Thus, we have

$$w(S_g) \ge (1 - O(g \cdot \varepsilon))\mu_w(G) \ge (1 - O(\log_\delta(1/\varepsilon) \cdot \varepsilon))\mu_w(G),$$

and

$$S_g \cap G_{[\ell_j, r_j)} \subseteq \bigcup_{i \in [k]} M_i$$

for all $j \in [k]$. Therefore, the matching S_g is contained in the union of all M_i's and consequently

$$\mu_w(M_1 \cup M_2 \cup \cdots \cup M_k) \ge w(S_g) \ge (1 - O(\log_\delta(1/\varepsilon) \cdot \varepsilon))\mu_w(G).$$

□

5 Framework

In this section, we will describe our framework in detail. As suggested by Lemma 3.1, we first fix a $\Theta(\varepsilon^{-3})$-wide weight partition, and compute a $(1-\varepsilon)$-approximate MWM on each "padded" weight classes with aspect ratio $\Theta(\varepsilon^{-5})$. The choice of width ensures that the set of odd "padded" weight classes has $\Theta(1/\varepsilon)$ weight gaps and so does the set of even ones. We use a subroutine Algorithm 1 in Section 5.1 to aggregate odd matchings and even matchings, and maintain a $(1-\varepsilon)$-approximate MWM on the union of them using the second subroutine Algorithm 2 in Section 5.2. In Section 5.3 we give the complete framework to reduce the aspect ratio with multiplicative $\mathrm{poly}(1/\varepsilon)$ overhead. In Section 5.4, we introduce a low-recourse transformation for $(1-\varepsilon)$-approximate dynamic MWM to change the multiplicative $\mathrm{poly}(1/\varepsilon)$ overhead to an additive $\mathrm{poly}(1/\varepsilon)$ overhead. Finally, in Section 5.5, we use the low-recourse transformation to obtain an efficient fully dynamic algorithm on low-degree graphs, which leads to an efficient weighted rounding algorithm and could also serve as an efficient aggregation that allows us to reduce the aspect ratio to $O(\varepsilon^{-2})$, which is the best we can hope for based on Lemma 3.1. Also, combined with [11] we achieve a $\mathrm{poly}(1/\varepsilon)$ multiplicative overhead reduction from weighted matching algorithms to unweighted ones in bipartite graphs.

5.1 Dynamic Approximate MWM on Matchings in $(1/\varepsilon)$-Spread Weight Classes

Our first subroutine is an improved algorithm that combines matchings in weight classes that are sufficiently spread. In particular, the goal is to solve the following problem.

Problem 5.1. ($(1-\varepsilon)$-Approximate MWM over Matchings in $(1/\varepsilon)$-Spread Weight Classes) *Given a set of $(1/\varepsilon)$-spread weight classes $[\ell_1, r_1), \ldots, [\ell_k, r_k) \subseteq \mathbb{R}$, and a set of k matchings $M_1, \ldots, M_k \subseteq G$ undergoing adversarial edge deletions/insertions satisfying $M_i \subseteq G_{[\ell_i, r_i)}$ for all $i \in [k]$. The task is to dynamically maintain a matching M satisfying*

$$w(M) \geq (1 - O(\varepsilon)) \sum_{i \in [k]} w(M_i).$$

As mentioned in Section 3.4, our improvement comes from maintaining the following *locally greedy census* matchings.

Definition 5.1. (Locally Greedy Census) *Consider k matchings $M_1, M_2, \ldots, M_k$. A matching M is a* locally greedy census matching *of $M_1, M_2, \ldots, M_k \subseteq G$ if for every edge $e \in M_i \setminus M$, there exists an $f \in M_j$ such that $e \cap f \neq \emptyset$ for some $j > i$.*

The above local notion should be compared with the standard greedy census matching considered in [4, 29, 38]. In the standard notion, an edge can only be removed if it is incident to some higher-weight edge *that is included into the output matching.* In contrast to that, in our locally greedy census matching, if an edge is incident to *any* higher-weight edge, regardless of whether that edge is in the output matching we are allowed to remove it. This allows us to consider potentially much fewer edges when maintaining the local greedy census matching. Nevertheless, we show that a similar charging argument can be used to prove the following guarantee.

Lemma 5.1. *For $\varepsilon \leq 1/2$, any set of $(1/\varepsilon)$-spread weight classes $[\ell_1, r_1), \ldots, [\ell_k, r_k) \subseteq \mathbb{R}$, and matchings $M_1, \ldots, M_k \subseteq G$ satisfying $M_i \subseteq G_{[\ell_i, r_i)}$ for all $i \in [k]$, every locally greedy census matching M over the union of $M_1, \ldots, M_k$ satisfies*

$$w(M) \geq (1 - 4\varepsilon) \sum_{i \in [k]} w(M_i).$$

Proof. The proof idea is similar to Lemma 4.1. For any edge $e \in M_j$, at most two edges in each lower weight class $i < j$ are not included in M because of e, and the total weight of these edges is at most

$$\sum_{i \in [j-1]} 2 \cdot w(e) \cdot \varepsilon^{-(i-j)} \leq \frac{2\varepsilon}{1-\varepsilon} \cdot w(e).$$

Thus,

$$w(M) \geq \left(1 - \frac{2\varepsilon}{1-\varepsilon}\right) \sum_{i \in [k]} w(M_i) \geq (1 - 4\varepsilon) \sum_{i \in [k]} w(M_i).$$

□

We show that our modified notion allows us to maintain a locally greedy census matching more efficiently than what is achieved in [38] for the non-local version. Remarkably, our algorithm achieves a constant update time when there are only $O(\log n)$ matchings.

Theorem 5.1. *Algorithm 1 initializes in $O(m)$ time and solves Problem 5.1 by dynamically maintaining a locally greedy census matching with $\min\{O(\log k), O(k/\log n)\}$ worst-case update time and $O(1)$ worst-case recourse.*

Proof. For any edge uv, it is contained in the locally greedy census matching if and only if it is in the highest weight class among N_u and N_v. By definition, after the initialization, Algorithm 1 maintains a locally greedy census matching. And the initialization takes $O(m)$ time.

For an edge update uv, the only possible changes in the locally greedy census matching are in N_u and N_v. For insertion of uv, Algorithm 1 checks whether the edges related to u and v in the current matching still satisfies

Algorithm 1: Dynamic Locally Greedy Census Matching

```
function Initialize()
    for each node u ∈ G do
        Initialize its neighborhood N_u ← ∅.
    for j = k, ..., 1 do
        for uv ∈ M_j do
            if N_u = ∅ and N_v = ∅ then
                Add uv to the matching.
            Add uv to N_u and N_v.
function Insert(j, uv)
    Add uv to N_u and N_v.
    if u is matched to some vertex u', and uu' ∈ M_i such that i < j then
        Delete uu' from the matching.
    if v is matched to some vertex v', and vv' ∈ M_i such that i < j then
        Delete vv' from the matching.
    if uv is in the highest weight class among N_u and N_v then
        Add uv to the matching.
function Delete(j, uv)
    Delete uv from N_u and N_v.
    if N_u is not empty then
        uu' ← the edge in the highest weight class among N_u.
        if uu' is in the highest weight class among N_{u'}. then
            Add uu' to the matching.
    if N_v is not empty then
        vv' ← the edge in the highest weight class among N_v.
        if vv' is in the highest weight class among N_{v'}. then
            Add vv' to the matching.
```

the condition, and whether uv can be added. For deletion of uv, only the edges in the highest weight class among N_u or N_v can be added into the matching. Algorithm 1 finds those edges and checks whether the condition is met. Therefore, it maintains a locally greedy census matching and the worst-case recourse is $O(1)$.

For each node u, there is at most one edge from each weight class in N_u, i.e., $|N_u| \leq k$. To maintain the maximum element in N_u, we can use a binary search tree which runs in $O(\log k)$ time. Both `Insert` and `Delete` have $O(1)$ number of updates and queries to the binary search tree. Therefore, the update time would be $O(\log k)$.

Alternatively, we can use a packed bit-representation of the weight-class information in N_u. We set the i-th bit to be 1 if and only if there is an edge in N_u in the $(k-i)$-th weight class. Thus to find the edge in the highest weight class among N_u, it suffices to look at the lowest bit in the representation, which can be done in $O(k/\log n)$ time in the word RAM model. □

5.2 Dynamic Approximate MWM on Degree-Two Graphs After combining the odd and even matchings with our locally greedy census matching algorithm, we are left with a union of two matchings which is a graph with maximum degree at most two. That is, we need to solve the following problem.

Problem 5.2. (Fully-Dynamic $(1-\varepsilon)$-Approximate MWM on Degree-Two Graphs) *Given a graph G undergoing edge updates satisfying that its maximum degree is at most two. The task is to dynamically maintain a matching M satisfying the following condition:*

$$w(M) \geq (1 - O(\varepsilon)) \cdot \mu_w(G).$$

Observe that a degree-two graph consists of paths and cycles. Since an exact MWM on a path or cycle P can be computed in $O(|P|)$ time with dynamic programming, it suffices to maintain a collection of short paths and cycles on which a large-weight matching is supported. For this, one can delete the minimum weight edge in each $\Theta(1/\varepsilon)$-length neighborhood while keeping a $1 - O(\varepsilon)$ fraction of the total weight. We propose Algorithm 2 to solve Problem 5.2 by dynamically maintaining this $O(1/\varepsilon)$-length decomposition of the paths and cycles and computing an exact MWM on each piece.

LEMMA 5.2. (DYNAMIC PATH/CYCLE MAINTAINER) *There is a deterministic data structure $\mathcal{D}$ that maintains a set of dynamic paths or cycles $\{P_i\}$ under the insertion/deletion of edges and supports the following operations, where all update times and recourse mentioned are worst-case:*

1. *Find the path/cycle P_u that u belongs to in $O(|P_u|)$ time.*
2. `FindHeads`(P)*: For a path P, find its both ends in $O(|P_u|)$ time.*
3. `Insert/Delete`(uv)*: Insert/delete an edge uv in $O(|P_u| + |P_v| + 1)$ time.*
4. `FindMin`(P, h, ℓ, r)*: For a path P, find the edge with the minimum weight between the ℓ-th and the r-th edges counting from h, one of the end of P, in $O(|P|)$ time.*
5. *For a path/cycle P, explicitly maintain its MWM in $O(|P|)$ time and recourse.*

Proof. We can check all elements in a path in linear time in its size. Thus the first 4 operations are straightforward to achieve. Now we prove that it can output the MWM. Consider the following dynamic programming for computing MWM on paths. For a path P, number its edges from P_1 to $P_{|P|}$. Denote $f_{i,0/1}$ as the MWM on the path $P_1 \dots P_i$ when P_i is in the matching or not. For any $i \le |P|$, $f_{i,0/1}$ can be computed by

$$f_{i,x} = w_i \cdot x + \max_{0 \le y \le 1-x} f_{i-1,y}.$$

Therefore, the value can be computed in $O(|P|)$ time and the edge list corresponding to the MWM can be inferred by taking notes of how each state is updated. □

LEMMA 5.3. *During the execution of Algorithm 2, $\mathcal{D}$ maintains a set of paths/cycles with length at most $3\lceil \varepsilon^{-1} \rceil$.*

Proof. The length of a path only increases after an edge insertion in $\mathcal{D}$, and Algorithm 2 calls `Maintain` every time which splits the path into two whenever its length is at least $3\lceil \varepsilon^{-1} \rceil$. Thus the paths have lengths at most $3\lceil \varepsilon^{-1} \rceil - 1$. The only case $\mathcal{D}$ keeps a cycle is that before the formation of that cycle, the path has a length at most $3\lceil \varepsilon^{-1} \rceil - 1$. Therefore, the cycle has length at most $3\lceil \varepsilon^{-1} \rceil$. □

LEMMA 5.4. *Algorithm 2 initializes in $O(m\varepsilon^{-1})$ time and solves Problem 5.2 by explicitly maintaining an matching with $O(\varepsilon^{-1})$ worst-case update time and $O(\varepsilon^{-1})$ worst-case recourse.*

Proof. Lemma 5.3 show that $\mathcal{D}$ maintains a set of paths/cycles with length at most $3\lceil \varepsilon^{-1} \rceil$. By Lemma 5.2, we know that each operation of $\mathcal{D}$ takes time $O(1/\varepsilon)$. Now consider the recurrence in `Maintain`. Any path that appears in `Maintain` has length at most $O(1/\varepsilon)$ and will be at least $\lceil \varepsilon^{-1} \rceil$ shorter in line 24. Thus there are only $O(1)$ recurrences in `Maintain`. Therefore, the worst-case update time of the algorithm is $O(1/\varepsilon)$ and the worst-case recourse is $O(1/\varepsilon)$.

Now we show it maintains a $(1 - O(\varepsilon))$-approximated MWM. In a degree-two graph, every connected component is either a path or a cycle, and $\mathcal{D}$ maintains an exact MWM on each component of $G \setminus R$ according to the last operation in Lemma 5.2. We know that $\mu_w(G) \ge \frac{1}{2} \sum_{e \in G} w_e$, since for each component in G, the "odd" edges and "even" edges both form a matching. On the other hand, an edge is added into R only if it is the minimum among a set of $\lceil \varepsilon^{-1} \rceil$ edges, and those sets are disjoint for different edges in R since we only add edges to R when the path is at least $3\lceil \varepsilon^{-1} \rceil$ long. Thus $\sum_{e \in R} w_e \le \varepsilon \cdot \sum_{e \in G} w_e \le 2\varepsilon \cdot \mu_w(G)$. Denote M as the matching output by $\mathcal{D}$, we have

$$w(M) = \mu_w(G \setminus R) \ge \mu_w(G) - \sum_{e \in R} w_e \ge (1 - 2\varepsilon)\mu_w(G).$$

□

Algorithm 2: Fully-Dynamic $(1-\varepsilon)$-Approximate MWM on Degree-Two Graphs

```
function Initialize()
    D ← an instance of the dynamic path/cycle maintainer described in Lemma 5.2.
    R ← ∅.
    for uv ∈ E do Insert(uv).
function Insert(u, v)
    D.Insert(uv).
    if P_u is a path then Maintain(P_u).
function Delete(uv)
    if uv ∈ R then  R ← R \ uv.
    D.Delete(uv).
    h_u, u ← D.FindHeads(P_u).
    if there is an edge h_u h'_u ∈ R then
        R ← R \ h_u h'_u.
        D.Insert(h_u h'_u).
        Maintain(P_u).
    h_v, v ← D.FindHeads(P_v).
    if there is an edge h_v h'_v ∈ R then
        R ← R \ h_v h'_v.
        D.Insert(h_v h'_v).
        Maintain(P_v).
function Maintain(P)
    h, t ← D.FindHeads(P).
    if |P| ≥ 3⌈ε^{-1}⌉ then
        uv ← D.FindMin(P, h, ⌊(|P| − ⌈ε^{-1}⌉)/2⌋, ⌊(|P| − ⌈ε^{-1}⌉)/2⌋ + ⌈ε^{-1}⌉ − 1).
        D.Delete(uv).
        R ← R ∪ {uv}.
        Maintain(P_u).
        Maintain(P_v).
```

5.3 Weight Reduction Framework for General Graphs We are now ready to show our main result, a deterministic framework with $\mathrm{poly}(1/\varepsilon)$ multiplicative overhead and recourse, which reduces the aspect ratio from W to $\mathrm{poly}(1/\varepsilon)$ for any $(1-\varepsilon)$-approximate dynamic MWM algorithm.

Theorem 5.2. *Given a dynamic $(1-\varepsilon)$-approximate MWM algorithm $\mathcal{A}$ that, on input n-vertex m-edge graph with aspect ratio W, has initialization time $\mathcal{I}(n,m,W,\varepsilon)$, amortized/worst-case update time $\mathcal{U}(n,m,W,\varepsilon)$, amortized/worst-case recourse $\sigma(n,m,W,\varepsilon)$, there is a transformation which produces a dynamic $(1-O(\varepsilon))$-approximate MWM algorithm with initialization time*

$$O(\mathcal{I}(n,m,\Theta(\varepsilon^{-5}),\Theta(\varepsilon)) + m\varepsilon^{-1})$$

time, amortized/worst-case update time

$$O(\mathcal{U}(n,m,\Theta(\varepsilon^{-5}),\Theta(\varepsilon)) + \sigma(n,m,\Theta(\varepsilon^{-5}),\Theta(\varepsilon))\varepsilon^{-1}),$$

and amortized/worst-case recourse

$$O(\sigma(n,m,\Theta(\varepsilon^{-5}),\Theta(\varepsilon))\varepsilon^{-1}).$$

The transformation is partially dynamic preserving.

There are three steps in Algorithm 3. In the first step, for all $1 \le i \le \lceil (L+1)/3 \rceil$, M_i is maintained by $\mathcal{A}_i$ and is a $(1-\varepsilon)$-approximation of $\mu_w(\widetilde{E}_i)$. In the second step, we use the locally greedy census matching Algorithm 1

Algorithm 3: Reduction Framework

Input: A dynamic algorithm for $(1-\varepsilon)$-approximate maximum weight matching $\mathcal{A}$

function `Initialize()`
- $L \leftarrow \lfloor \log_{1/\varepsilon} W \rfloor = \widetilde{O}(1)$.
- $E_{-1} = E_{L+1} = \emptyset$.
- **for** $i = 0, \ldots, L$ **do**
  - $E_i \leftarrow \{e \in E : \lfloor \log_{1/\varepsilon} w(e) \rfloor = i\}$.
- **for** $i = 1, \ldots, \lceil (L+1)/3 \rceil$ **do**
  - $\ell_i \leftarrow 3i - 3, r_i \leftarrow \min(L, 3i-1)$.
  - $\widetilde{E}_i \leftarrow \bigcup_{j=\ell_i - 1}^{r_i + 1} E_j$.
  - $\mathcal{A}_i \leftarrow$ an independent copy of $\mathcal{A}$.
  - Initialize $\mathcal{A}_i$ with $\widetilde{E}_i$.
  - Denote M_i as the matching maintained by $\mathcal{A}_i$.
- $\mathcal{C}_1, \mathcal{C}_2 \leftarrow$ two independent copies of Algorithm 1.
- Initialize $\mathcal{C}_1$ with $\{M_i \mid i \equiv 1 \pmod 2 \wedge 1 \le i \le \lceil (L+1)/3 \rceil\}$.
- Initialize $\mathcal{C}_2$ with $\{M_i \mid i \equiv 0 \pmod 2 \wedge 1 \le i \le \lceil (L+1)/3 \rceil\}$.
- Denote M_{odd} as the matching maintained by $\mathcal{C}_1$ and M_{even} as the one maintained by $\mathcal{C}_2$.
- $\mathcal{M} \leftarrow$ Algorithm 2.
- Initialize $\mathcal{M}$ with $M_{\text{odd}} \cup M_{\text{even}}$.
- **output** the matching maintained by $\mathcal{M}$.

function `Update(`e`)`
- $j \leftarrow \lfloor \log_{1/\varepsilon} w(e) \rfloor$.
- Update E_j accordingly.
- **for** $i : 1 \le i \le \lceil (L+1)/3 \rceil \wedge \ell_i - 1 \le j \le r_i + 1$ **do**
  - Update $\widetilde{E}_i$ based on the update in E_j.
  - Use $\mathcal{A}_i$ to maintain M_i based on the update in $\widetilde{E}_i$.
  - **if** *i is odd* **then** Use $\mathcal{C}_1$ to maintain M_{odd} based on the update in M_i.
  - **else** Use $\mathcal{C}_2$ to maintain M_{even} based on the update in M_i.
- Feed the updates in $\mathcal{C}_1$ and $\mathcal{C}_2$ into $\mathcal{M}$.
- **output** the matching maintained by $\mathcal{M}$.

to aggregate M_i for odd i and even i respectively, into M_{odd} and M_{even}, with the guarantee from Lemma 5.1 that M_{odd} and M_{even} both keep at least a $(1-4\varepsilon)$ fraction of the total weight of the corresponding matchings. Then we use Algorithm 2 for degree-two graphs to aggregate M_{odd} and M_{even}, and Lemma 5.4 shows that the final matching output by Algorithm 3 is a $(1-2\varepsilon)$-approximated MWM on $M_{\text{odd}} \cup M_{\text{even}}$. We will prove that since at each step we lose a $O(\varepsilon)$ fraction, the final matching we output keeps a $(1-O(\varepsilon))$-approximate MWM.

Lemma 5.5. *For $\varepsilon \le 1/2$, Algorithm 3 maintains a matching M with $\mu_w(M) \ge (1-O(\varepsilon))\mu_w(G)$.*

Proof. Lemma 3.1 shows that

$$\mu_w(M_1 \cup M_2 \cup \cdots \cup M_{\lceil (L+1)/3 \rceil}) \ge (1-O(\varepsilon))\mu_w(G).$$

Consider the locally greedy census matching M_{odd} and M_{even}. Denote $I_{\text{odd}} = \{1 \le i \le \lceil (L+1)/3 \rceil : i \text{ is odd}\}$, and $I_{\text{even}} = \{1 \le i \le \lceil (L+1)/3 \rceil : i \text{ is even}\}$. Lemma 5.1 shows that

$$w(M_{\text{odd}}) \ge (1-O(\varepsilon)) \sum_{i \in I_{\text{odd}}} w(M_i) \quad \text{and} \quad w(M_{\text{even}}) \ge (1-O(\varepsilon)) \sum_{i \in I_{\text{even}}} w(M_i).$$

Also, we know

$$M_{\text{odd}} \subseteq \bigcup_{i \in I_{\text{odd}}} M_i \quad \text{and} \quad M_{\text{even}} \subseteq \bigcup_{i \in I_{\text{even}}} M_i,$$

thus

$$\begin{aligned}\mu_w(M_{\text{odd}} \cup M_{\text{even}}) &\geq \mu_w\left(\bigcup_{i=1}^{\lceil (L+1)/3 \rceil} M_i\right) - \left(\sum_{i \in I_{\text{odd}}} w(M_i) - w(M_{\text{odd}})\right) - \left(\sum_{i \in I_{\text{even}}} w(M_i) - w(M_{\text{even}})\right) \\ &\geq (1 - O(\varepsilon))\mu_w(G) - O(\varepsilon) \cdot \sum_{i \in I_{\text{odd}}} w(M_i) - O(\varepsilon) \cdot \sum_{i \in I_{\text{even}}} w(M_i).\end{aligned}$$

Lemma 4.1 shows that

$$\sum_{i \in I_{\text{odd}}} w(M_i) \leq (1 + 4\varepsilon)\mu_w(G) \quad \text{and} \quad \sum_{i \in I_{\text{even}}} w(M_i) \leq (1 + 4\varepsilon)\mu_w(G),$$

and thus

$$\mu_w(M_{\text{odd}} \cup M_{\text{even}}) \geq (1 - O(\varepsilon))\mu_w(G).$$

The final matching M we output is a $(1 - 2\varepsilon)$-approximate MWM on $M_{\text{odd}} \cup M_{\text{even}}$. Therefore,

$$\mu(M) \geq (1 - O(\varepsilon))\mu_w(G).$$

∎

LEMMA 5.6. *Algorithm 3 initializes in $O(\mathcal{I}(n, m, \Theta(\varepsilon^{-5}), \Theta(\varepsilon)) + m\varepsilon^{-1})$ time, and has update time $O(\mathcal{U}(n, m, \Theta(\varepsilon^{-5}), \Theta(\varepsilon)) + \sigma(n, m, \Theta(\varepsilon^{-5}), \Theta(\varepsilon))\varepsilon^{-1})$ and recourse $O(\sigma(n, m, \Theta(\varepsilon^{-5}), \Theta(\varepsilon))\varepsilon^{-1})$.*

Proof. Each weight class $[\ell, r)$ has $\Theta(\varepsilon^{-5})$ aspect ratio. Thus each edge update $e \in E_j$ causes $\sigma(n, m, \Theta(\varepsilon^{-5}), \Theta(\varepsilon))$ changes in corresponding M_is which take $O(\mathcal{U}(n, m, \Theta(\varepsilon^{-5}), \Theta(\varepsilon)))$ update time. By Theorem 5.1, $\mathcal{C}_1, \mathcal{C}_2$ both handle each of these changes in $O(\log_{1/\varepsilon} W / \log n) = O(1)$ time and recourse, thus the update time of $\mathcal{M}$ would be $O(\sigma(n, m, \Theta(\varepsilon^{-5}), \Theta(\varepsilon))\varepsilon^{-1})$ and the recourse is at most $O(\sigma(n, m, \Theta(\varepsilon^{-5}), \Theta(\varepsilon))\varepsilon^{-1})$. The initialization time follows from that of each subroutine. ∎

By repeatedly applying Lemma 3.1 and Algorithm 2, we can further reduce the aspect ratio.

THEOREM 5.3. *Given a dynamic $(1 - \varepsilon)$-approximate MWM algorithm $\mathcal{A}$ that, on input n-vertex m-edge graph with aspect ratio W, has initialization time $\mathcal{I}(n, m, W, \varepsilon)$, amortized/worst-case update time $\mathcal{U}(n, m, W, \varepsilon)$, and amortized/worst-case recourse $\sigma(n, m, W, \varepsilon)$, there is a transformation that produces a dynamic $(1 - O(\varepsilon))$-approximate MWM algorithm that has initialization time*

$$O(\mathcal{I}(n, m, \Theta(\varepsilon^{-2-3\cdot 2^{-d}}), \Theta(\varepsilon)) + m\varepsilon^{-1}),$$

amortized/worst-case update time

$$O(\mathcal{U}(n, m, \Theta(\varepsilon^{-2-3\cdot 2^{-d}}), \Theta(\varepsilon)) + \sigma(n, m, \Theta(\varepsilon^{-2-3\cdot 2^{-d}}), \Theta(\varepsilon))\varepsilon^{-(1+d)}),$$

and amortized/worst-case recourse

$$O(\sigma(n, m, \Theta(\varepsilon^{-2-3\cdot 2^{-d}}), \Theta(\varepsilon))\varepsilon^{-(1+d)})$$

for any integer parameter $d \in \mathbb{Z}_{\geq 0}$. The transformation is partially dynamic preserving.

Proof. For each weight class $[\ell, r)$ with aspect ratio $\Theta(\varepsilon^{-2-3\cdot 2^{-x}})$ (we start with $x = 0$, i.e., $\Theta(\varepsilon^{-5})$), denote $m = \sqrt{\ell \cdot r}$. There is a consistent constant c in Lemma 3.1 such that given two $(1 - \varepsilon)$-approximate MWM M_1

and M_2 on the "padded" weight classes $[\ell, m \cdot \varepsilon^{-1})$ and $[m \cdot \varepsilon, r)$ respectively, there is a matching of the weight class $[\ell, r)$ on $M_1 \cup M_2$ with approximation ratio

$$\left(1 - c \cdot \left(\log(1/\varepsilon)/\log(\varepsilon^{-1-3\cdot 2^{-(x+1)}})\right) \cdot \varepsilon\right) = \left(1 - \frac{c}{1 + 3 \cdot 2^{-(x+1)}} \cdot \varepsilon\right) \geq (1 - c \cdot \varepsilon).$$

Algorithm 2 can maintain a $(1-\varepsilon)$-approximate MWM on the union, thus maintain a $(1-(c+1)\cdot\varepsilon)$-approximate matching, and we reduce the aspect ratio from $\Theta(\varepsilon^{-2-3\cdot 2^{-x}})$ to $\Theta(\varepsilon^{-2-3\cdot 2^{-(x+1)}})$. Repeatedly applying Lemma 3.1 and Algorithm 2 for d times, we achieve a d-depth binary tree representation of the weight reduction. Each inner node of the binary tree is a matching maintained on the union of its offspring. Since for each layer, we lose a $c+1$ factor in the approximation error, the matching maintained at the root has an approximation ratio $1-(c+1)^d \cdot \varepsilon$.

Since there are $2^d = O(1)$ nodes in the binary tree, the initialization takes time

$$O(\mathcal{I}(n, m, \Theta(\varepsilon^{-2-3\cdot 2^{-d}}), \Theta(\varepsilon)) + m\varepsilon^{-1}).$$

For the update time and recourse, consider the layers in decreasing depths. In the deepest layer with depth d, there are $2^d = O(1)$ nodes. The edge change could occur in each of them, so there is an update time $O(\mathcal{U}(n, m, \Theta(\varepsilon^{-2-3\cdot 2^{-d}}), \Theta(\varepsilon)))$ and recourse $O(\sigma(n, m, \Theta(\varepsilon^{-2-3\cdot 2^{-d}}), \Theta(\varepsilon)))$. For the layer with depth $d-1$, the number of edge updates in total equals the recourse of the layer with depth d, thus both the update time and recourse would be $O(\sigma(n, m, \Theta(\varepsilon^{-2-3\cdot 2^{-d}}), \Theta(\varepsilon)))\varepsilon^{-1}$. Suppose $\varepsilon \leq 1/2$, an easy induction shows that the total update time would be

$$O(\mathcal{U}(n, m, \Theta(\varepsilon^{-2-3\cdot 2^{-d}}), \Theta(\varepsilon)) + \sigma(n, m, \Theta(\varepsilon^{-2-3\cdot 2^{-d}}), \Theta(\varepsilon))\varepsilon^{-(1+d)}),$$

and the recourse would be

$$O(\sigma(n, m, \Theta(\varepsilon^{-2-3\cdot 2^{-d}}), \Theta(\varepsilon))\varepsilon^{-(1+d)}).$$

Combined with Theorem 5.2 we finish the proof. □

5.4 Low-Recourse Transformation The update time of our reduction comprises two parts: the original update time of the algorithm $\mathcal{A}$ and its recourse. According to Theorem 5.3, the multiplicative overhead on the update time is constant while that on the recourse is $\text{poly}(1/\varepsilon)$. A high recourse of the algorithm could make it inefficient when serving as a subroutine. [37] provides a low-recourse transformation that reduces the recourse to worst-case $O(W/\varepsilon)$ for any α-approximate dynamic MWM algorithm. In this section, we design a tailored low-recourse transformation for $(1-\varepsilon)$-approximate weight matching that reduces the recourse to amortized $O(\text{poly}(\log W)/\varepsilon)$ (see Theorem 5.4). Besides efficiency, the low-recourse transformation can be applied to an algorithm that implicitly maintains a matching as long as it supports the following vertex-match query, relaxing the requirement of explicitly maintaining the matching.

Definition 5.2. (Vertex-Match Query) *A dynamic matching algorithm is said to support the vertex-match query in query time T if given any vertex query v, it answers in $O(T)$ time either v is unmatched in the maintained matching or the matched vertex of v; and it can output all the edges in the maintained matching M in $O(|M| \cdot T)$ time.*

Formally, denote G_0 as the initial graph and G_i as the graph after the i-th update. The recourse of a dynamic matching algorithm $\mathcal{A}$ measures the changes in the support set of the matching maintained by $\mathcal{A}$, which is defined as follows.

Definition 5.3. (Worst-Case Recourse of a Dynamic Matching Algorithm) *For a fixed dynamic matching algorithm $\mathcal{A}$ that (possibly implicitly) maintains a matching M_i on graph G_i, the worst-case recourse of $\mathcal{A}$ on $G_0, G_1, \ldots, G_k$ is defined as $\max_{i\in[k]} |M_i \oplus M_{i-1}|$, i.e., the maximum changes in the matching edge set.*

Definition 5.4. (Amortized Recourse of a Dynamic Matching Algorithm) *For a fixed dynamic matching algorithm $\mathcal{A}$ that (possibly implicitly) maintains a matching M_i on graph G_i, the amortized recourse of $\mathcal{A}$ on $G_0, G_1, \ldots, G_k$ is defined as $\frac{1}{k}\sum_{i\in[k]} |M_i \oplus M_{i-1}|$, i.e., the average changes in the matching edge set.*

We start with designing a transformation between two matchings $\widetilde{M}_i$ on G_i and M_j on G_j where $i < j$ that builds a large matching on G_j based on $\widetilde{M}_i$ and M_j with small $|\widetilde{M}_i \oplus M_j|$. We will use Δ-additive-approximate to represent an additive approximation. Formally, a matching M on G is Δ-*additive-approximate* if $w(M) \geq \mu_w(G) - \Delta$.

Algorithm 4: Direct Transformation between Two Time Points

Input: Two matchings $\widetilde{M}_i$ on G_i and M_j on G_j.

1 Consider $P = \widetilde{M}_i \oplus M_j$.

2 Denote U as the set of updated edges between time $i+1$ and time j.

3 $D \leftarrow \emptyset$.

4 **for** *any edge e in $U \cap P$ and each of its direction* **do**

5   **if** *there exists at least $2/\varepsilon$ edges in that direction in P* **then**

6   Denote E' as the closest $2/\varepsilon$ edges.

7   **if** *E' doesn't contain any edge in U* **then**

8   Add the edge with minimum weight in $E' \cap M_j$ into D.

9 For those paths and cycles in $P \setminus D$ that contains edges in U, $\widetilde{M}_j$ picks edges in M_j.

10 For the remaining ones, $\widetilde{M}_j$ pick edges in $\widetilde{M}_i$.

11 `return` $\widetilde{M}_j$.

Claim 5.1. *Suppose $\widetilde{M}_i$ is a Δ_i-additive-approximate MWM on G_i and M_j is a Δ_j-additive-approximate MWM on G_j. Then, Algorithm 4 outputs a $(\Delta_i + \Delta_j + \varepsilon \cdot \mu_w(G_j))$-additive-approximate MWM $\widetilde{M}_j$ on G_j such that $|\widetilde{M}_i \oplus \widetilde{M}_j| \leq O((j-i) \cdot \varepsilon^{-1})$.*

Proof. Starting with M_j, the additive approximation is Δ_j. It increases by $\varepsilon \cdot \mu_w(G_j)$ during the deletion in line 8 and Δ_i during the substitution in line 10. Since any edge in $\widetilde{M}_i \oplus \widetilde{M}_j$ belongs to a path or cycle in $P \setminus D$ that contains an edge in U, by construction, its size is bounded by $O(1/\varepsilon) \cdot |U| = O((j-i) \cdot \varepsilon^{-1})$. ∎

Now we are ready to introduce the full transformation. By running an independent copy of $\mathcal{A}$ on the unweighted version of G we assume that we have access to a $(1-\varepsilon)$-approximation ν_i to the size of the MCM in G_i. The full transformation works in multiple phases, where each phase spans a contiguous segment of time points. Suppose that a phase starts at time t. The transformation reads the entire edge set of the weighted matching M_t maintained by $\mathcal{A}$ on G_t and sets the length of the phase to be $\varepsilon \cdot \nu_t$. Then, we define several checkpoints t_i within this phase, where t_0 is set to t and the remaining checkpoints are defined iteratively as $t_{i+1} \stackrel{\text{def}}{=} t_i + \frac{\varepsilon \cdot w(M_{t_i})}{W}$. The transformation will compute a matching $\widetilde{M}_{t_i}$ for each checkpoint, and this matching will be used as the output from this time point until the next checkpoint. That is, for each time point $t_i < j < t_{i+1}$, the matching output by the transformation on G_j will simply be $\widetilde{M}_{t_i} \cap G_j$, which is $\widetilde{M}_{t_i}$ with edges deleted in G_j dropped.

Lemma 5.7. *Suppose on G_{t_i}, $\widetilde{M}_{t_i}$ is a $(1 - x \cdot \varepsilon)$-approximate MWM. Then, for any j such that $t_i < j < t_{i+1}$, $\widetilde{M}_{t_i} \cap G_j$ is a $(1 - (x+2) \cdot \varepsilon)$-approximate MWM on G_j.*

Proof. Denote U as the edge updates between t_i and t_{i+1} then

$$w(U) \stackrel{\text{def}}{=} \sum_{e \in U} w(e) \leq \frac{\varepsilon \cdot w(M_{t_i})}{W} \cdot W \leq \varepsilon \cdot \mu_w(G_{t_i}).$$

The weight of the matching is at least

$$w(\widetilde{M}_{t_i} \cap G_j) \geq w(\widetilde{M}_{t_i}) - w(U) \geq (1 - (x+1) \cdot \varepsilon) \cdot \mu_w(G_{t_i}),$$

while the MWM in the graph has weight at most

$$\mu_w(G_j) \leq \mu_w(G_{t_i}) + w(U) \leq (1+\varepsilon) \cdot \mu_w(G_{t_i}).$$

Thus

$$w(\widetilde{M}_{t_i} \cap G_j) \geq \frac{1-(x+1)\cdot\varepsilon}{1+\varepsilon} \cdot \mu_w(G_j) \geq (1-(x+2)\cdot\varepsilon)\cdot\mu_w(G_j).$$

□

Lemma 5.7 shows that it suffices to maintain a good approximate MWM $\widetilde{M}_{t_i}$ at each checkpoint. Below we show that the number of checkpoints within each phase is bounded by $O(W)$.

Lemma 5.8. *Each phase has at most $\frac{W}{(1-\varepsilon)^2}$ checkpoints.*

Proof. For a phase starting at time t_0, its length is $\varepsilon \cdot \nu_{t_0}$. The gap between any checkpoints t_i and t_{i+1} is at least

$$\frac{\varepsilon \cdot w(M_{t_i})}{W} \geq \frac{\varepsilon(1-\varepsilon)\cdot\mu_w(G_{t_i})}{W} \geq \frac{\varepsilon(1-\varepsilon)\cdot\mu(G_{t_i})}{W} \geq \frac{\varepsilon(1-\varepsilon)\cdot(\mu(G_{t_0})-\varepsilon\nu_{t_0})}{W} \geq \frac{\varepsilon(1-\varepsilon)^2\nu_{t_0}}{W},$$

thus the number of checkpoints is at most $\frac{W}{(1-\varepsilon)^2}$. □

Algorithm 4 provides a direct transformation between any two checkpoints. The full transformation will use Algorithm 4 as a subroutine. The initial idea is to link the checkpoints in a path-like way, i.e., the matching maintained by the full transformation $\widetilde{M}_{t_i}$ on G_{t_i} is the output of Algorithm 4 on $\widetilde{M}_{t_{i-1}}$ and M_{t_i}, where M_{t_i} is the matching maintained by $\mathcal{A}$ on G_{t_i}. Two issues arise. The first issue is that the guarantee of Claim 5.1 is an additive approximation, thus $\mu_w(G_{t_{i-1}})$ should not be too much larger than $\mu_w(G_{t_i})$. The second issue is that the path length is $O(W)$. Since after one direct transformation, the approximation error accumulates, suppose we choose a $(1-\delta)$-approximate MWM algorithm, the final approximation error could reach $O(W\cdot\delta)$.

We solve the above issues in the following way. The first issue can be fixed by only allowing $\widetilde{M}_{t_i}$ to be transformed by some checkpoint t_j with $\mu_w(G_{t_j}) \leq 2\cdot\mu_w(G_{t_i})$. The second issue is fixed by linking the checkpoints in a tree-like way instead of a path-like. Formally, we define the transformation tree as follows.

Definition 5.5. (Transformation Tree) *The transformation tree is a rooted tree where the nodes represent distinct checkpoints and could have* ***ordered*** *children. The degree of a transformation tree is the maximum number of children of any node. The depth of a node in the transformation tree is the number of edges in the path between the root and that node. The depth of the transformation tree is the largest depth of its node. The mapping between the checkpoints and the nodes will ensure that the preorder traversal of the transformation tree corresponds to a contiguous subarray of the checkpoints, i.e., $t_i, t_{i+1}, \cdots, t_j$. Further, it ensures that for any pair of nodes t_i, t_j such that t_j is an ancestor of t_i in the transformation tree, $\mu_w(G_{t_j}) \leq O(1)\cdot\mu_w(G_{t_i})$.*

Lemma 5.9. *Given a transformation tree with depth d and degree c that corresponds to the checkpoints $t_i, t_{t+1}, \ldots, t_j$ and a dynamic $(1-\varepsilon)$-approximate MWM algorithm $\mathcal{A}$ with initialization time $\mathcal{I}(n,m,W,\varepsilon)$, update time $\mathcal{U}(n,m,W,\varepsilon)$ and query time $T(n,m,W,\varepsilon)$, there is an algorithm that dynamically and explicitly maintains a $(1-O(d\cdot\varepsilon))$-approximate MWM on $G_{t_i}, G_{t_{i+1}}, \ldots, G_{t_j}$ with initialization time*

$$\mathcal{I}(n,m,W,\varepsilon) + O(\nu_{t_i})\cdot T(n,m,W,\varepsilon),$$

amortized update time

$$\mathcal{U}(n,m,W,\varepsilon) + O(c\cdot d\cdot\varepsilon^{-1})\cdot T(n,m,W,\varepsilon),$$

and ensures that

$$\frac{1}{t_j - t_i}\sum_{k=i+1}^{j} |\widetilde{M}_{t_k} \oplus \widetilde{M}_{t_{k-1}}| = O(c\cdot d\cdot\varepsilon^{-1}),$$

where $\widetilde{M}_{t_k}$ is the matching output by the framework on G_{t_k}. The transformation is partially dynamic preserving.

Proof. We first describe the transformation. For the root t_i, we set $\widetilde{M}_{t_i} = M_{t_i}$. For any checkpoint $t_k > t_i$, it has a parent node $t_p < t_k$ in the transformation tree. We run Algorithm 4 on $\widetilde{M}_{t_p}$ and M_{t_k} to get $\widetilde{M}_{t_k}$. We now establish the guarantee of the transformation.

Approximation Error Claim 5.1 shows that for any checkpoint $t_k > t_i$ and its parent node t_p, the additional approximation error of $\widetilde{M}_{t_k}$ increases by $O(\varepsilon) \cdot \mu_w(G_{t_k})$ compared to that of $\widetilde{M}_{t_p}$, since M_{t_k} is a $(1-\varepsilon)$-approximate MWM on G_{t_k}. Denote A_k as the set of ancestors of t_k, then the additive approximation error of $\widetilde{M}_{t_k}$ is at most $O(\varepsilon) \cdot \sum_{t_l \in A_k \cup \{t_k\}} \mu_w(G_{t_l}) = O(d \cdot \varepsilon) \cdot \mu_w(G_{t_k})$, since the definition of a transformation tree ensures that for any $t_l \in A_k$, $\mu_w(G_{t_l}) \leq O(1) \cdot \mu_w(G_{t_k})$.

Runtime and Recourse The additional initialization time is the cost of reading the edge set of M_{t_i}. For the update time and recourse, consider a fixed checkpoint t_k and its parent t_p. Using the vertex-match query of $\mathcal{A}$, we can find all edges in $\widetilde{M}_{t_k} \oplus \widetilde{M}_{t_p}$ in time $O((t_k - t_p) \cdot \varepsilon^{-1}) \cdot T(n, m, W, \varepsilon)$ by Claim 5.1. In other words, the cost of a direct transformation from t_p to t_k can be amortized by all updates between t_p and t_k and the amortized additional update time is $O(\varepsilon^{-1}) \cdot T(n, m, W, \varepsilon)$ while the amortized recourse is $O(\varepsilon^{-1})$. It suffices to show that for any fixed update t, there will be at most $O(c \cdot d)$ direct transformation that covers it, i.e., the number of pairs t_k and its parent node t_p such that $t_p \leq t \leq t_k$ is at most $O(c \cdot d)$. The definition of the transformation tree ensures that its subtree also corresponds to a contiguous subarray of checkpoints. Consider a fixed depth of nodes in the transformation tree. The subtrees with those nodes as root correspond to disjoint contiguous subarray. Thus t could be included in at most one of them, i.e., the number of distinct t_p is at most $O(d)$. Since the degree of each node is c, we conclude the proof. ∎

Below we show an online construction of $O(\log W)$ transformation trees corresponding to disjoint contiguous subarrays whose union covers the entire phase.

Algorithm 5: Online Construction of Transformation Trees within a Phase

---

Input: A set of checkpoints $\{t_0, t_1, \ldots, t_l\}$ where $l = O(W)$.

Denote M_{t_i} as the matching maintained by $\mathcal{A}$ on G_{t_i}.
$\theta \leftarrow \lceil \log \frac{W}{(1-\varepsilon)^2} \rceil = O(\log W)$.
`root` $\leftarrow t_0$.
`root.complete` $\leftarrow 0$.
`root.depth` $\leftarrow 0$.
`cur` $\leftarrow t_0$.
for $i = 1, \ldots, l$ **do**
  while $\lfloor \log w(M_{\texttt{cur}}) \rfloor > \lfloor \log w(M_{t_i}) \rfloor$ *and* `cur` $\neq$ `root` **do**
    `cur` $\leftarrow$ `cur.father`.
  if $\lfloor \log w(M_{\texttt{cur}}) \rfloor > \lfloor \log w(M_{t_i}) \rfloor$ **then**
    `root` $\leftarrow t_i$.
    t_i`.depth` $\leftarrow 0$.
  else
    t_i`.father` $\leftarrow$ `cur`.
    t_i`.depth` $\leftarrow$ `cur.depth` $+ 1$.
  if t_i`.depth` $= \theta$ **then**
    while `cur.complete` $= 1$ **do**
      `cur` $\leftarrow$ `cur.father`.
    `cur.complete` $= 1$.
  else
    `cur` $\leftarrow t_i$.
    `cur.complete` $= 0$.

Lemma 5.10. *Algorithm 5 constructs $O(\log W)$ transformation trees with depth $O(\log W)$ and degree $O(\log W)$ in amortized $O(1)$ time that corresponds to contiguous subarrays that are disjoint and their union covers the entire phase.*

Proof. It is clear that Algorithm 5 constructed a set of transformation trees in amortized $O(1)$ time with depth

$O(\log W)$ that are disjoint. Since there are $O(W)$ checkpoints within a phase according to Lemma 5.8 and the choice of θ, those transformation tree satisfy the covering property. We will prove that there are $O(\log W)$ of them, each with degree $O(\log W)$.

Algorithm 5 creates a new transformation tree whenever the root changes, i.e., at line 11. Thus the new root satisfies that $\lfloor \log w(M_{\texttt{new\_root}}) \rfloor < \lfloor \log w(M_{\texttt{old\_root}}) \rfloor$. The length of a phase starting with t_0 is set to be $\varepsilon \cdot \nu_{t_0}$. Thus for any checkpoint t_i within the phase,

$$(1-\varepsilon) \cdot \nu_{t_0} \leq w(M_{t_i}) \leq \frac{1+\varepsilon}{1-\varepsilon} \cdot W \cdot \nu_{t_0},$$

meaning that the number of different $\lfloor \log w(M_{t.}) \rfloor$ is at most $O(\log W)$.

For each node v in the constructed tree, $v.\texttt{complete}$ represents whether there is a subtree with one of its children as root that is a complete binary tree with depth $\theta - v.\texttt{depth}$. $v.\texttt{complete}$ can only be 0 or 1 during the execution. And whenever it changes to 1, v would continue building its subtree with a new child node. Therefore, with the same reason as the number of roots, v would have $O(\log W)$ children when $v.\texttt{complete} = 0$ and $O(\log W)$ children when $v.\texttt{complete} = 1$, proving that the degree is $O(\log W)$. ∎

Theorem 5.4. *Given a dynamic $(1-\varepsilon)$-approximate MWM algorithm $\mathcal{A}$ with initialization time $\mathcal{I}(n,m,W,\varepsilon)$, update time $\mathcal{U}(n,m,W,\varepsilon)$ and query time $T(n,m,W,\varepsilon)$, there is a transformation that produces a dynamic algorithm that explicitly maintains a $(1-O(\varepsilon \cdot \log W))$-approximate MWM with initialization time*

$$\mathcal{I}(n,m,W,\varepsilon),$$

amortized update time

$$\mathcal{U}(n,m,W,\varepsilon) + O(\log^2 W \cdot \varepsilon^{-1}) \cdot T(n,m,W,\varepsilon),$$

and amortized recourse

$$O(\log^2 W \cdot \varepsilon^{-1}).$$

The transformation is partially dynamic preserving.

Proof. The only initialization time is for $\mathcal{A}$. We would prove the amortized update time and recourse within each phase. According to Lemma 5.10, Algorithm 5 builds $O(\log W)$ transformation trees with $O(\log W)$ depth and degree that correspond to disjoint contiguous subarrays of checkpoints and their union covers the entire phase. Lemma 5.9 shows that for each transformation tree, we spend $O(\nu_{t_0}) \cdot T(n,m,W,\varepsilon)$ time and $O(\nu_{t_0})$ recourse for initialization of the transformation. The initialization for $O(\log W)$ transformation trees can be amortized over the entire phase to be $O(\log W \cdot \varepsilon^{-1}) \cdot T(n,m,W,\varepsilon)$ amortized update time and $O(\log W \cdot \varepsilon^{-1})$ recourse. Thus the bottleneck is the $O(\log^2 W \cdot \varepsilon^{-1}) \cdot T(n,m,W,\varepsilon)$ amortized update time and $O(\log^2 W \cdot \varepsilon^{-1})$ recourse that each transformation tree induces by the checkpoints other than root. ∎

Consequently, our framework has a $\mathrm{poly}(1/\varepsilon)$ additive overhead independent of the underlying algorithm. For simplicity, in the further use of this result in this work, we only consider algorithms that explicitly maintain the matching with $T(n,m,W,\varepsilon) = O(1)$.

Corollary 5.1. *Given a dynamic $(1-\varepsilon)$-approximate MWM algorithm $\mathcal{A}$ that, on input n-vertex m-edge graph with aspect ratio W, has initialization time $\mathcal{I}(n,m,W,\varepsilon)$, update time $\mathcal{U}(n,m,W,\varepsilon)$ and query time $T(n,m,W,\varepsilon)$, there is a transformation that produces a dynamic $(1-O(\varepsilon \log \varepsilon^{-1}))$-approximate MWM algorithm that has initialization time*

$$O(\mathcal{I}(n,m,\Theta(\varepsilon^{-2-3\cdot 2^{-d}}),\Theta(\varepsilon)) + m\varepsilon^{-1}),$$

amortized update time

$$O(\mathcal{U}(n,m,\Theta(\varepsilon^{-2-3\cdot 2^{-d}}),\Theta(\varepsilon)) + \varepsilon^{-2-d}\log^2 \varepsilon^{-1} \cdot T(n,m,\Theta(\varepsilon^{-2-3\cdot 2^{-d}}),\varepsilon)),$$

and amortized recourse

$$O(\varepsilon^{-2-d}\log^2 \varepsilon^{-1})$$

for any integer parameter $d \in \mathbb{Z}_{\geq 0}$. The transformation is partially dynamic preserving.

Proof. It follows from combining Theorem 5.3 and Theorem 5.4. ∎

5.5 Putting Everything Together With the help of our low-recourse transformation, we obtain an efficient fully dynamic algorithm on low-degree graphs, which then leads to a $\widetilde{O}(\text{poly}(1/\varepsilon))$ update time rounding algorithm for weighted fractional matchings. The low-degree algorithm can also serve as an efficient aggregation even if we reduce the aspect ratio to $O(\varepsilon^{-2})$ in our framework, which is the best we can hope for based on Lemma 3.1. Finally, combined with [11] we reduce weighted matching algorithms to unweighted ones in bipartite graphs.

5.5.1 Fully Dynamic Algorithm on Low-Degree Graphs

THEOREM 5.5. *Given an n-vertex m-edge graph G with edge weights bounded by W that undergoes edge insertions and deletions such that the maximum degree of G is bounded by Δ, there is an algorithm with $O(\Delta\varepsilon^{-5}\log^2\varepsilon^{-1})$ amortized update time and $O(\varepsilon^{-5}\log^2\varepsilon^{-1})$ amortized recourse that explicitly maintains a $(1-O(\varepsilon\log\varepsilon^{-1}))$-approximate MWM in G.*

Proof. For a subgraph of G, we can use the following standard algorithm [29] to maintain a $(1-\varepsilon)$-approximate MWM on it.

FACT 5.1. *There is a fully dynamic $(1-\varepsilon)$-approximate MWM algorithm that has amortized update time $O(\Delta W\varepsilon^{-2}\log(\varepsilon^{-1}))$ and recourse $O(\Delta W\varepsilon^{-1})$ on a graph with maximum degree Δ and aspect ratio W.*

The theorem follows by applying Corollary 5.1 with $d = 3$. ◻

5.5.2 Rounding Weighted Fractional Matching We obtain a weighted rounding algorithm with polynomial dependence on ε^{-1}, showing that the dynamic fractional matching problem is as hard as the integral one up to $\text{poly}(1/\varepsilon)$ factors. Formally, a dynamic weighted rounding algorithm is defined as follows.

DEFINITION 5.6. (SEE E.G., [22, DEFINITION 3.11]) *A* dynamic rounding algorithm*, for a given n-vertex graph $G=(V,E)$, edge weights $\boldsymbol{w}\in\mathbb{N}^E$ bounded by $W=\text{poly}(n)$, and accuracy parameter $\varepsilon>0$, initializes with an $\boldsymbol{x}\in\mathcal{M}_G$ and must maintain an integral matching $M\subseteq\text{supp}(\boldsymbol{x})$ with $\boldsymbol{w}(M)\geq(1-\varepsilon)\boldsymbol{w}^\top\boldsymbol{x}$ under entry updates to $\boldsymbol{x}$ that guarantee $\boldsymbol{x}\in\mathcal{M}_G$ after each operation.*

We are going to use the given fractional matching to find a sparse subgraph on which there is a large weight matching.

LEMMA 5.11. ([22]) *Given an $\varepsilon>0$, there is a deterministic algorithm that, on an m-edge graph G with edge weights bounded by W and a fractional matching $\boldsymbol{x}\in\mathcal{M}_G$, initializes in $\widetilde{O}(m)$ time, supports*

- *inserting/deleting and edge or changing the value of $\boldsymbol{x}_e$ in amortized $\widetilde{O}(W\cdot\varepsilon^{-1})$ time,*

and maintains

- *a subgraph $H\subseteq G$ with maximum degree $\widetilde{O}(\varepsilon^{-2})$ on which a fractional matching $\boldsymbol{x}^{(H)}$ of weight $\sum_{e\in E}w(e)\boldsymbol{x}_e^{(H)}\geq(1-\varepsilon)\sum_{e\in E}w(e)\boldsymbol{x}_e$ that satisfies $\boldsymbol{x}^{(H)}(v)\leq\boldsymbol{x}(v)+O(\varepsilon)$ for all $v\in V$ and $\left|\boldsymbol{x}_e^{(H)}-\boldsymbol{x}_e\right|\leq O(\varepsilon^2)$ for all $e\in E$.*

LEMMA 5.12. *Given a fractional matching $\boldsymbol{x}\in\mathcal{M}_G$, for any $\varepsilon>0$ we can initialize in $\widetilde{O}(m)$ a subgraph H of G and maintain it with $\widetilde{O}(\varepsilon^{-1})$ time per entry update to $\boldsymbol{x}$ such that H has maximum degree $\widetilde{O}(\varepsilon^{-2})$ and $\mu_w(H)\geq(1-O(\varepsilon))\boldsymbol{w}^\top\boldsymbol{x}$ holds.*

Proof. We split the edges into $K=O(\log W)$ classes $E_0\cup\cdots\cup E_K$ such that E_i contain precisely the edges with weights between 2^i and $2^{i+1}-1$ (inclusively). Let G_i be the induced subgraph of E_i. Let $\varepsilon'=\Theta(\varepsilon/K)$. We run Lemma 5.11 on each of the G_i with accuracy ε' and obtain H_i. Let $H=H_0\cup\cdots\cup H_{O(\log W)}$. By Lemma 5.11, there is a fractional matching $\boldsymbol{x}^{(H_i)}$ of weight

$$\sum_{e\in E_i}\boldsymbol{w}_e\boldsymbol{x}_e^{(H_i)}\geq(1-\varepsilon')\sum_{e\in E_i}\boldsymbol{w}_e\boldsymbol{x}_e$$

and therefore letting $\boldsymbol{x}^{(H)} \stackrel{\text{def}}{=} \boldsymbol{x}^{(H_1)} + \cdots + \boldsymbol{x}^{(H_K)}$ we have $\boldsymbol{w}^\top \boldsymbol{x}^{(H)} \geq (1-\varepsilon')\boldsymbol{w}^\top \boldsymbol{x}$. Notice that $\boldsymbol{x}^{(H)}(v) \leq \boldsymbol{x}(v) + O(\varepsilon' K) \leq 1 + O(\varepsilon)$ for each $v \in V$ and $\boldsymbol{x}^{(H)}[B] \leq \boldsymbol{x}[B] + O((\varepsilon')^2 \cdot K) \cdot |B|^2 \leq \boldsymbol{x}[B] + O(\varepsilon)|B|$ for all odd sets of size at most $O(1/\varepsilon)$. Therefore, we see that if we scale $\boldsymbol{x}^{(H)}$ down by a multiplicative $1 + O(\varepsilon)$ factor then it will be a feasible fractional matching supported on H. This proves that $\mu_w(H) \geq (1-\varepsilon)\boldsymbol{w}^\top \boldsymbol{x}$. The update time of the algorithm follows from that of Lemma 5.11 which is $\widetilde{O}({\varepsilon'}^{-1}) = \widetilde{O}(\varepsilon^{-1})$ since the edge weights in a single E_i are within a factor of two from each other. ∎

Theorem 5.6. *Given an m-edge graph, there is a dynamic rounding algorithm that initializes in $\widetilde{O}(m)$ time and handles each entry update to $\boldsymbol{x}$ in $\widetilde{O}(\varepsilon^{-8})$ time per update.*

Proof. Given the fractional matching $\boldsymbol{x}$, we run Lemma 5.12 to maintain a $\widetilde{O}(\varepsilon^{-2})$-degree subgraph $H \subseteq G$ with $\mu_w(G) \geq (1 - O(\varepsilon))\boldsymbol{w}^\top \boldsymbol{x}$. We then apply Theorem 5.5 to maintain a $(1 - O(\varepsilon \log \varepsilon^{-1}))$-approximate matching over H. By the guarantee of H, such a matching will have weight at least $(1 - O(\varepsilon \log \varepsilon^{-1}))\boldsymbol{w}^\top \boldsymbol{x}$. Since H has maximum degree $\widetilde{O}(\varepsilon^{-2})$, Theorem 5.5 handles each update to H in $\widetilde{O}(\varepsilon^{-7})$. The theorem follows as there are $\widetilde{O}(\varepsilon^{-1})$ modifications to H per update to $\boldsymbol{x}$ by Lemma 5.12. ∎

5.5.3 Improved Weight Reduction Framework for General Graphs

The fully dynamic low-degree algorithm in Theorem 5.5 allows us to reduce the aspect ratio to $O(\varepsilon^{-2})$, which is the best we can get using Lemma 3.1.

Theorem 5.7. *Given a dynamic $(1-\varepsilon)$-approximate MWM algorithm $\mathcal{A}$ that, on input n-vertex m-edge graph with aspect ratio W, has initialization time $\mathcal{I}(n, m, W, \varepsilon)$, and update time $\mathcal{U}(n, m, W, \varepsilon)$, there is a transformation which produces a dynamic $(1 - \varepsilon \log(\varepsilon^{-1}))$-approximate MWM dynamic algorithm that has initialization time*

$$\log(\varepsilon^{-1}) \cdot O(\mathcal{I}(n, m, \Theta(\varepsilon^{-2}), \Theta(\varepsilon)) + m\varepsilon^{-1}),$$

amortized update time

$$\mathrm{poly}(\log(\varepsilon^{-1})) \cdot O(\mathcal{U}(n, m, \Theta(\varepsilon^{-2}), \Theta(\varepsilon)) + \varepsilon^{-5}),$$

and amortized recourse

$$\mathrm{poly}(\log(\varepsilon^{-1})) \cdot O(\varepsilon^{-5}).$$

The transformation is partially dynamic preserving.

Proof. According to Lemma 3.1, we consider any 2-wide weight partition of G. Denote $g = \lceil \log(\varepsilon^{-3}) \rceil + 1$, and for $0 < j \leq g-1$, denote $I_j = \{i : i \equiv j \pmod{g}\}$. The weight gap between neighboring "padded" weight classes in I_j is $\Omega(\varepsilon^{-1})$, and we use Algorithm 1 to aggregate matchings in I_j and use the low-degree algorithm in Theorem 5.5 to maintain a $(1 - \varepsilon \log \varepsilon^{-1})$-approximate MWM on the union. Following a similar proof of Lemma 3.1 itself, the union of the greedy matchings keeps a $1 - O(\varepsilon \log \varepsilon^{-1})$ fraction thus so does the output matching.

Any edge would be contained in $O(\log(\varepsilon^{-1}))$ "padded" weight classes. For initialization, we use Theorem 2.1 to compute a $(1-\varepsilon)$-approximate MWM on the union, and it takes $m \log(\varepsilon^{-1})\varepsilon^{-1}$ time. The update time and recourse come from Theorem 5.1 and Theorem 5.5. ∎

5.5.4 From Weighted Matching to Unweighted Matching in Bipartite Graphs

[11] provides a framework to reduce dynamic weighted matching algorithms to unweighted ones in bipartite graphs. We slightly optimize their algorithm to have better runtime. The description of the algorithm, Algorithm 6, and its proof are deferred to Section A.

Lemma 5.13. ([11, 33]) *For $\varepsilon \leq 1/6$, given a dynamic algorithm $\mathcal{A}$ that, on input n-vertex m-edge **bipartite** graph, initializes in $\mathcal{I}(n, m, \varepsilon)$ time and explicitly maintains an $(1-\varepsilon)$-approximate MCM in $\mathcal{U}(n, m, \varepsilon)$ update time, there is a dynamic algorithm that initializes in*

$$O(\mathcal{I}(nW, mW, \varepsilon) + m \log(\varepsilon^{-1})\varepsilon^{-1})$$

time and explicitly maintains an $(1-\varepsilon)$-approximate MWM on a bipartite graph with integer edge weights bounded by W in

$$O(W \cdot \mathcal{U}(nW, mW, \Theta(\varepsilon))) + W \log(\varepsilon^{-1})\varepsilon^{-2})$$

amortized update time and has amortized recourse

$$O(W \log(\varepsilon^{-1})\varepsilon^{-2}).$$

The transformation is partially dynamic preserving. On non-bipartite graphs, the approximation ratio is $\frac{2}{3} - \varepsilon$.

Lemma 5.13 requires integer weights. By a standard scaling and rounding argument, one can reduce a problem with real weights and W aspect ratio to the same problem with integer weights and $W\varepsilon^{-1}$ aspect ratio. Combined Theorem 3.2, Lemma 5.13 and the above fact, we have the following reduction from weighted matching to unweighted matching.

THEOREM 5.8. *Given a dynamic algorithm $\mathcal{A}$ that, on input n-vertex m-edge unweighted* ***bipartite*** *graph, initializes in $\mathcal{I}(n, m, \varepsilon)$ time and explicitly maintains an $(1 - \varepsilon)$-approximate MCM in $\mathcal{U}(n, m, \varepsilon)$ update time, there is a dynamic algorithm that initializes in*

$$\log(\varepsilon^{-1}) \cdot O(\mathcal{I}(\Theta(\varepsilon^{-3})n, \Theta(\varepsilon^{-3})m, \Theta(\varepsilon)) + m\varepsilon^{-1})$$

time and explicitly maintains an $(1 - O(\varepsilon \log(\varepsilon^{-1})))$-approximate MWM in

$$\text{poly}(\log(\varepsilon^{-1})) \cdot O(\mathcal{U}(\Theta(\varepsilon^{-3})n, \Theta(\varepsilon^{-3})m, \Theta(\varepsilon)) \cdot \varepsilon^{-3} + \varepsilon^{-5})$$

amortized update time, and has amortized recourse

$$\text{poly}(\log(\varepsilon^{-1})) \cdot O(\varepsilon^{-5}).$$

The transformation is partially dynamic preserving. In non-bipartite graphs, the approximation ratio changes to $\frac{2}{3} - O(\varepsilon \log(\varepsilon^{-1}))$ *while the runtime and recourse remain the same.*

The reduction improves on the work of [11], whose reduction, when combined with [29], has an update time of $\mathcal{U}(\varepsilon^{-O(1/\varepsilon)}n, \varepsilon^{-O(1/\varepsilon)}m, \Theta(\varepsilon)) \cdot \varepsilon^{-O(1/\varepsilon)} \cdot \log W$.

6 Applications

In this section, we discuss the implications of our reduction frameworks for obtaining $(1-\varepsilon)$-approximate maximum weight matching algorithms in various models.

6.1 The Dynamic Model In the dynamic setting, barring a few exceptions (for example, [28, 17]), much of the focus has been designing algorithms for $(1 - \varepsilon)$-approximate MCM. Thus, a lot of the weighted matching results for *bipartite graphs* follow from the reduction of [11], and consequently, incur a multiplicative overhead of $\varepsilon^{-O(1/\varepsilon)}$. In this section, we remedy this. A summary of our results is given in Table 1. We start with bipartite graphs.

Table 1: Summary of Prior and Our Results on Dynamic Weighted Matching

Setting	Prior Result	Our Result	Reduction
Fully Dynamic Bipartite	$\varepsilon^{-O(1/\varepsilon)} \cdot \frac{n}{2^{\Omega(\sqrt{\log n})}}$ [36]+[11]	$O(\text{poly}(\varepsilon^{-1}) \cdot \frac{n}{2^{\Omega(\sqrt{\log n})}})$ Lemma 6.1	Thm 5.8
Incremental Bipartite	$O(m \log n \log^2(nW/\varepsilon)\varepsilon^{-2})$ [17] (fractional) $m \cdot \varepsilon^{-O(1/\varepsilon)} \cdot \log W$ [20]+[11]	$O((n\varepsilon^{-9} + m\varepsilon^{-8}) \cdot \text{poly}(\log(1/\varepsilon))$ Lemma 6.2	Thm 5.8
Fully Dynamic General	$\sqrt{m} \cdot \varepsilon^{-O(1/\varepsilon)} \cdot \log W$ [29]	$O((\sqrt{m} \cdot \varepsilon^{-4} + \varepsilon^{-5}) \cdot \text{poly}(\log(1/\varepsilon))$ Lemma 6.7	Thm 3.2
Decremental General	$m \cdot \text{poly}(\log(nW)) \cdot \varepsilon^{-O(1/\varepsilon)}$ [22]	$O(m \cdot \text{poly}(\log n, \varepsilon^{-1}))$ Lemma 6.8	Thm 3.2
Fully Dynamic Offline Bipartite	$O(n^{0.58} \cdot \varepsilon^{-O(1/\varepsilon)} \cdot \log W)$ [36]+[11]	$O(n^{0.58} \cdot \text{poly}(\varepsilon^{-1}))$ Lemma 6.3	Thm 5.8

Bipartite Graphs For bipartite graphs, we show the following three results in the fully dynamic and incremental setting respectively.

Lemma 6.1. *There is a fully dynamic randomized algorithm that maintains a $(1-\varepsilon)$-approximate MWM in a bipartite graph in $O(\text{poly}(\varepsilon^{-1}) \cdot \frac{n}{2^{\Omega(\sqrt{\log n})}})$ update time.*

Lemma 6.2. *There is a deterministic incremental algorithm that maintains a $(1-\varepsilon)$-approximate MWM in an incremental bipartite graph in $O(n\varepsilon^{-9} \text{poly}(\log 1/\varepsilon) + m\varepsilon^{-8} \text{poly}(\log 1/\varepsilon))$ total update time.*

Lemma 6.3. *There is a randomized algorithm that given an offline sequence of edge insertions and deletions to an n-vertex bipartite weighted graph, maintains the edges of a $(1-\varepsilon)$-approximate maximum weight matching in amortized $O(n^{0.58} \text{poly}(\varepsilon^{-1}))$ time with high probability.*

We now prove Lemma 6.1. In order to achieve this, we use the following recent result by [36].

Lemma 6.4. ([36]) *There is a fully dynamic randomized algorithm that maintains a $(1-\varepsilon)$-approximate MCM in a bipartite graph in $O(\text{poly}(\varepsilon^{-1}) \cdot \frac{n}{2^{\Omega(\sqrt{\log n})}})$ update time.*

Proof of Lemma 6.1. The result follows from Lemma 6.4 and Theorem 5.8. ∎

Prior to Lemma 6.1, the best-known result had an update time of $\varepsilon^{-O(1/\varepsilon)} \cdot \frac{n}{2^{-\Omega(\sqrt{\log n})}}$, which was obtained by combining the result of [36] and [11]. We now show Lemma 6.2, and for that we need the following recent result by Blikstad and Kiss.

Lemma 6.5. ([20]) *There exists a deterministic incremental algorithm that maintains a $(1-\varepsilon)$-approximate MCM in an incremental bipartite graph in $O(n\varepsilon^{-6} + m\varepsilon^{-5})$ total update time.*

Proof of Lemma 6.2. The result follows from the application of Theorem 5.8 to the amortized runtime given by Lemma 6.5. ∎

Prior to Lemma 6.2, the best known incremental algorithm has an update time of $O(m \cdot \log n \cdot \log^2(nW\varepsilon^{-1}) \cdot \varepsilon^{-2})$ (see [17]) or $O(\varepsilon^{-O(1/\varepsilon)} \cdot m \cdot \log W)$ (by combining [20] and [29]). We now prove Lemma 6.3. In order to achieve this, we use the following recent result by [36].

Lemma 6.6. ([36]) *There is a randomized algorithm that given an offline sequence of edge insertions and deletions to an n-vertex bipartite graph, maintains the edges of $(1-\varepsilon)$-approximate matching in amortized $O(n^{0.58} \text{poly}(\varepsilon^{-1}))$ time with high probability.*

Proof. The result follows from Theorem 5.8 and Lemma 6.3. ∎

We now show our results for general graphs.

General Graphs For general graphs, we show two results in the fully dynamic and decremental settings, respectively.

Lemma 6.7. *There is a deterministic fully dynamic algorithm that maintains a $(1-\varepsilon)$-approximate MWM in $O((\sqrt{m} \cdot \varepsilon^{-4} + \varepsilon^{-5}) \cdot \text{poly}(\log(1/\varepsilon))$ update time.*

Note that prior to this work, the best-known update time for an algorithm that maintains a $(1-\varepsilon)$-approximate maximum weight matching was $O(\sqrt{m} \cdot \varepsilon^{-O(1/\varepsilon)} \cdot \log W)$. This was obtained by combining the results of [29] with their bucketing scheme.

Lemma 6.8. *There is a randomized decremental algorithm that maintains a $(1-\varepsilon)$-approximate MWM in a decremental general graph in $O(m \cdot \text{poly}(\log n, \varepsilon^{-1}))$ total update time.*

Prior to this work, the best known decremental algorithm maintaining a $(1-\varepsilon)$-approximate maximum weight matching in a general graph had $O(\varepsilon^{-O(1/\varepsilon)} \cdot \text{poly}(\log n))$-update time, and was obtained by combining the results of [22] with the bucketing scheme of [29]. We now proceed with showing the proof of Lemma 6.7. For this, we use the following result.

LEMMA 6.9. ([29]) *There is a deterministic fully dynamic algorithm that maintains a $(1-\varepsilon)$-approximate MWM in $O(\sqrt{m}W\varepsilon^{-2})$ update time.*

Proof of Lemma 6.7. The result follows from Lemma 6.9 and Theorem 3.2. ∎

We now show the proof of Lemma 6.8. Our proof uses the following recent result by [22].

LEMMA 6.10. ([22]) *There is a randomized decremental algorithm that maintains a $(1-\varepsilon)$-approximate MWM in a decremental general graph in $O(m \cdot W \cdot \text{poly}(\log n, \varepsilon^{-1}))$ total update time.*

Proof of Lemma 6.8. The result follows from Lemma 6.10 and Theorem 3.2. ∎

6.2 The Streaming Model

Model Definition In the streaming model, the edges of the input n-vertex graph $G = (V, E)$ are presented to the algorithm in a stream (in an arbitrary order). A *semi-streaming algorithm* is allowed to make one or a few passes over the stream, use a limited amount of memory $O(n \,\text{poly}(\log n))$, and at the end output a solution to the problem at hand, say, find an approximate maximum weight matching of G.

Our Results As in the dynamic case, we obtain two types of reductions. First is an aspect ratio reduction, and the second, is a weighted to unweighted reduction for bipartite graphs.

THEOREM 6.1. *Suppose there is a semi-streaming algorithm for $(1-\varepsilon)$-approximate maximum weight matching in an n-node m-edge general graph with aspect ratio W that uses $p(n, m, W, \varepsilon)$ passes and space $s(n, m, W, \varepsilon)$, then for any constant $c > 0$ there exists a semi-streaming algorithm for $(1 - c^{-1}\varepsilon)$-approximate maximum weight matching that uses $p(n, m, \Theta(\varepsilon^{-(2+c)}), \varepsilon)$ passes and space complexity $O(s(n, m, \Theta(\varepsilon^{-(2+c)}), \varepsilon) \cdot \log_{\varepsilon^{-1}} W)$.*

THEOREM 6.2. *Suppose there is a semi-streaming algorithm for $(1-\varepsilon)$-approximate MCM in an n-node m-edge bipartite graph that uses $p(n, m, \varepsilon)$ passes and space $s(n, m, \varepsilon)$, then there exists a semi-streaming algorithm for $(1 - O(\varepsilon))$-approximate maximum weight matching in a n-node m-edge bipartite graph with aspect ratio W that uses $p(\Theta(n \cdot \varepsilon^{-(3+c)}), \Theta(m \cdot \varepsilon^{-(3+c)}), \Theta(\varepsilon))$ passes and space $O(s(\Theta(n \cdot \varepsilon^{-(3+c)}), \Theta(m \cdot \varepsilon^{-(3+c)}), \Theta(\varepsilon)) \cdot \log_{\varepsilon^{-1}} W)$.*

The above reduction has the property that it is a weighted to unweighted reduction that preserves the number of passes, while increasing the space complexity by a factor of $\log W$.

As a consequence of our reductions, we new results and trade-offs for streaming $(1-\varepsilon)$-approximate bipartite maximum weight matching, which are summarized in the Table 2. We state them formally thereafter.

Table 2: Summary of Results on $(1-\varepsilon)$-approximate Bipartite MWM in Streaming

Prior Result		Our Result ($\forall c > 0$ constant)	Reduction Used
$O(\varepsilon^{-2} \cdot \log(\varepsilon^{-1}))$ passes $O(n\varepsilon^{-2} \log W)$ space	[2]	$O(\varepsilon^{-2})$ passes $O(n \cdot \varepsilon^{-(3+c)} \log W)$ space Lemma 6.11	Theorem 6.2
$O(\varepsilon^{-7} \cdot \log^3(1/\varepsilon))$ passes $O(n \cdot \log(1/\varepsilon) \cdot \log W)$ space	[35]	$O(\varepsilon^{-4} \cdot \log^3(1/\varepsilon))$ passes $O(n \cdot \log W)$ space Lemma 6.15	Theorem 6.1

Our first result is the following lemma.

LEMMA 6.11. *For any constant $c > 0$, there is a semi-streaming algorithm for $(1-\varepsilon)$-approximate bipartite maximum weight matching that uses $O(n \cdot \varepsilon^{-(3+c)} \cdot \log W)$ space and has a pass complexity of $O(\varepsilon^{-2})$.*

Prior to this, the semi-streaming algorithm of [2] had the best known pass complexity of $O(\varepsilon^{-2} \cdot \log(\varepsilon^{-1}))$. The second result is the following.

LEMMA 6.12. *For any constant $c > 0$ there is a semi-streaming algorithm for $(1-\varepsilon)$-approximate bipartite maximum weight matching that uses $O(n \cdot \log \varepsilon^{-1} \cdot \log W)$ space and has a pass complexity of $O(\varepsilon^{-4} \cdot \log^3(1/\varepsilon))$.*

This improves on the result of [35] which had the same space complexity, but a pass complexity of $O(\varepsilon^{-8})$ passes.

Proofs of Our Streaming Results We first begin by stating the proofs of our reductions. We start by proving Theorem 6.2. This will be done using Theorem 6.1 and the following result by [11, 33]. While their result is stated as being applicable to integral weight graphs, as mentioned before, by standard scaling and rounding techniques, one can reduce the arbitrary weight case to the integral case. We state a modified version of their result incorporating this.

LEMMA 6.13. ([11, 33]) *Suppose $\mathcal{A}_u$ is a streaming algorithm that computes a $(1-\varepsilon)$-approximation to the MCM in $p(n, m, \varepsilon)$ passes and $s(n, m, \varepsilon)$ space. Then, there is a streaming algorithm $\mathcal{A}_w$ that computes a $(1-\varepsilon)$-approximation to the maximum weight matching in $p(nW\varepsilon^{-1}, mW\varepsilon^{-1}, \varepsilon)$ passes and $s(nW\varepsilon^{-1}, mW\varepsilon^{-1}, \varepsilon)$ space, where W is the aspect ratio of the weighted graph.*

Proof of Theorem 6.2. Suppose $\mathcal{A}_u$ is the bipartite unweighted matching algorithm in the premise of the lemma. Then, we can use Lemma 6.13 to get an algorithm $\mathcal{A}_w$ with space complexity $O(s(nW\varepsilon^{-1}, mW\varepsilon^{-1}, \varepsilon))$ space, and $p(nW, mW, \varepsilon)$ passes. Applying Theorem 6.1 to $\mathcal{A}_w$, get an algorithm $\mathcal{A}'_w$ with pass complexity $p(\Theta(n\cdot\varepsilon^{-(3+c)}), \Theta(m\cdot\varepsilon^{-(3+c)}), \Theta(\varepsilon))$ and space complexity $O(s(\Theta(n\cdot\varepsilon^{-(3+c)}), \Theta(m\cdot\varepsilon^{-(3+c)}), \Theta(\varepsilon))\cdot\log_{\varepsilon^{-1}} W)$. □

We now show how to implement our reduction in streaming.

Proof of Theorem 6.1. Let $\mathcal{A}$ be the algorithm given in the premise of the lemma. As in Theorem 3.2, we consider any ε^{-c}-wide weight partition of G, and let I_j's be the set of "padded" weight classes. Then, by Lemma 3.1, the union of matchings M_j on I_j contains a $(1-c^{-1}\varepsilon)$-approximate maximum weight matching of G. We run a copy of $\mathcal{A}$ on each of these weight classes I_j and then combine them. Since we run $\log_{\varepsilon^{-1}} W$ copies of $\mathcal{A}$ and the aspect ratio of the weight classes is $\varepsilon^{-(2+c)}$, we have the desired space and pass bound. □

We now show the proof of Lemma 6.11. Our proof uses the following result by [8].

LEMMA 6.14. ([8]) *There is a semi-streaming algorithm for $(1-\varepsilon)$-approximate bipartite MCM that uses $O(n)$ space, and has a pass complexity of $O(\varepsilon^{-2})$.*

Proof of Lemma 6.11. Instantiating the reduction of Theorem 6.2 with the algorithm of Lemma 6.14, we obtain the desired $O(n\cdot\varepsilon^{-(3+c)}\cdot\log W)$ space complexity and has a pass complexity of $O(\varepsilon^{-2})$ □

For Lemma 6.12, we need the following result.

LEMMA 6.15. ([35]) *There is a semi-streaming algorithm for $(1-\varepsilon)$-approximate bipartite maximum weight matching that has uses $O(n\log(1/\varepsilon))$ space and has a pass complexity of $O(\log^3 W\cdot\varepsilon^{-4})$. By applying the reduction of [29], one can obtain an algorithm that uses $O(n\cdot\log(\varepsilon^{-1})\cdot\log W)$ space and has a pass complexity of $O(\varepsilon^{-7}\log^3(1/\varepsilon))$.*

We now show Lemma 6.12, which improves the pass Lemma 6.15, while still achieving a space complexity which has logarithmic dependence on $\frac{1}{\varepsilon}$.

Proof of Lemma 6.12. Let $\mathcal{A}$ be the algorithm in Lemma 6.15 which has a space complexity of $O(n\cdot\log(1/\varepsilon))$ and has a pass complexity of $O(\log^3 W\cdot\varepsilon^{-4})$. We instantiate the reduction in Theorem 6.1 with $\mathcal{A}$. This yields a semi-streaming algorithm that satisfies the premise of the corollary. □

6.3 The MPC Model

Model Definition In the MPC Model, there are p machines, each with a memory of size s, such that $p\cdot s = O(m)$. The computation proceeds in synchronous rounds: in each round, each machine performs some local computation and at the end of the round they exchange messages. All messages sent and received by each machine in each round have to fit into the local memory of the machine, and hence their length is bounded by s in each round. At the end, the machines collectively output the solution. In this paper, we work in the *linear memory* model in which, the memory per machine is $s = \tilde{O}(n)$. We first state our results in this model.

Table 3: Summary of Results on $(1-\varepsilon)$-approximate Bipartite Matching in MPC Model

Rounds	Space	Weighted/Unweighted	Reference
$O(\varepsilon^{-2}\log\log n)$	$O(n)$	Unweighted	[8]
$O(\varepsilon^{-8}\log\log n)$	$O(n\log_{\varepsilon^{-1}} W)$	Weighted	[35]
$O(\log\log(n/\varepsilon)\cdot\varepsilon^{-2})$	$O(n\cdot\varepsilon^{-(3+c)}\cdot\log_{\varepsilon^{-1}} W)$	Weighted	Lemma 6.16
$O(\log^3(1/\varepsilon)\cdot\log\log(n/\varepsilon)\cdot\varepsilon^{-4})$	$O(n\log_{\varepsilon^{-1}} W)$	Weighted	Lemma 6.17

Our Results As in the dynamic and streaming case, we give the following reductions, the first one being an aspect ratio reduction, and the second, a reduction from weighted to unweighted matching in *bipartite graphs*.

THEOREM 6.3. *Suppose there is an MPC algorithm for $(1-\varepsilon)$-approximate maximum weight matching in an n-node m-edge general graph with aspect ratio W that uses $r(n,m,W,\varepsilon)$ rounds and space $s(n,m,W,\varepsilon)$ per machine, then for any constant $c>0$ there exists an MPC algorithm for $(1-O(\varepsilon))$-maximum weight matching that uses $r(n,m,\Theta(\varepsilon^{-(2+c)}),\varepsilon)$ rounds and $O(s(n,m,\Theta(\varepsilon^{-(2+c)}),\varepsilon)\cdot\log W+n\log W)$ space per machine.*

THEOREM 6.4. *Suppose there is an MPC algorithm for $(1-\varepsilon)$-approximate MCM in an n-node m-edge bipartite graph that uses $r(n,m,\varepsilon)$ passes and space $s(n,m,\varepsilon)$, then there exists an MPC algorithm for $(1-O(\varepsilon))$-approximate maximum weight matching in a n-node m-edge bipartite graph with aspect ratio W that uses $r(\Theta(n\cdot\varepsilon^{-(3+c)}),\Theta(m\cdot\varepsilon^{-(3+c)}),\Theta(\varepsilon))$ rounds and space $O(s(\Theta(n\cdot\varepsilon^{-(3+c)}),\Theta(m\cdot\varepsilon^{-(3+c)}),\Theta(\varepsilon))\cdot\log W+n\log W)$ per machine.*

As a consequence of these two reductions, we get the following two results about about $(1-\varepsilon)$-approximate bipartite maximum weight matching, which matches the round complexity of the best known MPC algorithm for unweighted matching by [8].

LEMMA 6.16. *There is an MPC algorithm for computing a $(1-\varepsilon)$-approximate bipartite maximum weight matching in $O(\log\log(n/\varepsilon)\cdot\varepsilon^{-2})$ rounds and $O(n\cdot\varepsilon^{-(3+c)}\cdot\log_{\varepsilon^{-1}} W)$ space per machine.*

The second lemma improves on the result of [35].

LEMMA 6.17. *There is an MPC algorithm for $(1-\varepsilon)$-approximate bipartite maximum weight matching using $O(\log^3(1/\varepsilon)\cdot\log\log n\cdot\varepsilon^{-4})$ rounds and $O(n\log_{\varepsilon^{-1}} W)$ space per machine.*

We summarize these results in Table 3.

Proofs in the MPC Model We first show the proof of our main reductions. We start with the proof of Theorem 6.4, and for that, we need the following theorem, which is implicit from the work of [11, 33].

LEMMA 6.18. (IMPLICIT IN [11, 33]) *Suppose $\mathcal{A}_u$ is an MPC algorithm that computes a $(1-\varepsilon)$-approximation to the MCM in $r(n,m,\varepsilon)$ rounds and $s(n,m,\varepsilon)$ space per machine. Then, there is an MPC algorithm $\mathcal{A}_w$ that computes a $(1-\varepsilon)$-approximation to the maximum weight matching in $r(nW\varepsilon^{-1},mW\varepsilon^{-1},\varepsilon)$ rounds and $s(nW\varepsilon^{-1},mW\varepsilon^{-1},\varepsilon)$ space per machine, where W is the aspect ratio of the weighted graph.*

The proof of Theorem 6.4 is implied by the above lemma, and Theorem 6.3.

Proof of Theorem 6.4. Suppose $\mathcal{A}_u$ is the bipartite unweighted matching algorithm in the premise of the lemma. Then, we can use Lemma 6.18 to get an algorithm $\mathcal{A}_w$ with $O(s(nW\varepsilon^{-1},mW\varepsilon^{-1},\varepsilon))$ space per machine, and $r(nW\varepsilon^{-1},mW\varepsilon^{-1},\varepsilon)$ rounds. Applying Theorem 6.3 to $\mathcal{A}_w$, get an algorithm $\mathcal{A}'_w$ with round complexity $r(\Theta(n\cdot\varepsilon^{-(3+c)}),\Theta(m\cdot\varepsilon^{-(3+c)}),\Theta(\varepsilon))$ rounds and space $O(s(\Theta(n\cdot\varepsilon^{-(3+c)}),\Theta(m\cdot\varepsilon^{-(3+c)}),\Theta(\varepsilon))\cdot\log W+n\log W)$ per machine. ∎

We now state the proof our aspect ratio reduction in MPC.

Proof of Theorem 6.3. Let $\mathcal{A}$ be the algorithm given in the premise of the lemma. As in Theorem 3.2, we consider any ε^{-c}-wide weight partition of G, and let I_j's be the set of "padded" weight classes. Then, by Lemma 3.1, then, the union of matchings M_j on I_j contains a $(1-c^{-1}\varepsilon)$-approximate maximum weight matching of G. We run a copy of $\mathcal{A}$ on each of these weight classes I_j and then combine them in a single matching. Since we run $\log W$ copies of $\mathcal{A}$ and the aspect ratio of the weight classes is $\varepsilon^{-(2+c)}$, we have the desired space and pass bound. ∎

We now show the proof of Lemma 6.16. In order to do that, we need the following result.

LEMMA 6.19. ([8]) *There is an MPC algorithm for $(1-\varepsilon)$-approximate bipartite matching using $O(\varepsilon^{-2} \cdot \log\log n)$ rounds and $O(n)$ space per machine.*

Proof of Lemma 6.16. Let $\mathcal{A}$ be the algorithm of Lemma 6.19. We instantiate the reduction in Theorem 6.4 with $\mathcal{A}$ to get an MPC algorithm for $(1-\varepsilon)$-approximate bipartite maximum weight matching that has $O(\log\log(n/\varepsilon)\cdot\varepsilon^{-2})$ round complexity and $O(n \cdot \varepsilon^{-(3+c)} \cdot \log_{\varepsilon^{-1}} W)$ space per machine. □

Next, we show the proof of Lemma 6.17. We need the following result.

LEMMA 6.20. ([35]) *There is an MPC algorithm for $(1-\varepsilon)$-approximate bipartite maximum weight matching using $O(\log^3(W) \log\log n \cdot \varepsilon^{-4})$ rounds and $O(n)$ space per machine. By applying the reduction of [29], we can obtain an MPC algorithm for the same problem that uses $O(\log\log n \cdot \varepsilon^{-7} \cdot \log^3(1/\varepsilon))$ rounds and $O(n \log_{\varepsilon^{-1}} W)$ space per machine.*

Proof of Lemma 6.17. Let $\mathcal{A}$ be the algorithm of Lemma 6.20 with space complexity $O(n)$ and round complexity $O(\log^3(W) \log\log n \cdot \varepsilon^{-4})$. Instantiating Theorem 6.3 with $\mathcal{A}$, we get a $(1-\varepsilon)$-approximate bipartite maximum weight matching with round complexity $O(\log^3(1/\varepsilon) \cdot \log\log(n/\varepsilon) \cdot \varepsilon^{-4})$ and space $O(n \cdot \log_{\varepsilon^{-1}} W)$ per machine. □

6.4 The Parallel Shared-Memory Work-Depth Model

Model Definition The parallel shared-memory work-depth model is a parallel model where different processors can process instructions in parallel and read and write from the same shared-memory. In this model, we care about two properties of the algorithm: *work*, which is the total amount of computation done by the algorithm and the *depth*, which is the longest chain of sequential dependencies in the algorithm. Our goal in this section will be to show that our reduction can be implemented in parallel model very efficiently. In particular, we show the following theorems.

THEOREM 6.5. *Suppose there is a parallel algorithm that computes a $(1-\varepsilon)$-approximate maximum weight matching on an n-node m-edge graph with aspect ratio W with $B(n,m,W,\varepsilon)$ work and $D(n,m,W,\varepsilon)$ depth. Then there exists a parallel algorithm that computes a $(1-\varepsilon)$-approximate maximum weight matching in $O(B(n,m,\varepsilon^{-5},\Theta(\varepsilon)) \cdot \log W + n\log n)$ work and $O(D(n,m,\varepsilon^{-5},\Theta(\varepsilon)) + \log W + \log^2 n)$ depth.*

Since the lack of a parallel implementation of the reduction in [11], we currently cannot reduce the weighted matching problem directly to an unweighted one. Thus we use Theorem 6.5 to improve the following weighted parallel algorithm by reducing the weight ranges.

LEMMA 6.21. ([35]) *There exists a shared-memory parallel algorithm that computes a $(1-\varepsilon)$-approximate maximum weight matching with $O(m\log^3(W)\varepsilon^{-4})$ work and $O(\log^3(W)\log^2(n)\varepsilon^{-4})$ depth. Using [29], this translates into a parallel algorithm which computes a $(1-\varepsilon)$-approximate maximum weight matching with $O(m\log(W)\log^3(1/\varepsilon)\varepsilon^{-7})$ work and $O(\log(W)\log^2(n)\log^3(1/\varepsilon)\varepsilon^{-7})$ depth.*

A consequence of our reduction is the following improvement, in both total work and depth.

COROLLARY 6.1. *There exists a shared-memory parallel algorithm that computes a $(1-\varepsilon)$-approximate maximum weight matching on an n-node m-edge graph with aspect ratio W with $O(m\log(W)\varepsilon^{-4})$ work and $O(\log^2(n)\log(W)\varepsilon^{-4})$ depth.*

We now show how to implement our reduction in the parallel model. The most challenging aspect of this implementation is to compute a maximum weight matching on degree two graphs. Such a graph is a collection of paths and cycles. First an observation is in order. Consider two paths $P_1 = (v_0, \cdots, v_{|P_1|})$ or $P_1 = (e_1, \cdots, e_{|P_1|})$ and $P_2 = (u_0, \cdots, u_{|P_2|})$ or $P_2 = (e'_1, \cdots, e_{|P_2|})$. As in the dynamic program described in Lemma 5.2, we maintain for P_1: $f(e_1, x, e_{|P_1|}, y)$ for $x, y \in \{0,1\}$. Here, $f(e_1, 0, e_{|P_1|}, 0)$is for example the value of the maximum weight matching on P_1 in which e_1 and e_2 are unmatched. Similarly, for P_2, we maintain $f(e'_1, x, e'_{|P_2|}, y)$ for $x, y \in \{0,1\}$.

Suppose, $P = P_1 \bigoplus e \bigoplus P_2$, where $e = (v_{|P_1|}, v_0)$, then, we can get the corresponding information for P as follows for all $x, y \in \{0, 1\}$.

$$f(e_1, x, e'_{|P_2|}, y) = \max_{\substack{z \in \{0,1\} \\ 0 \leqslant s,t \leqslant 1-z}} \left\{ f(e_1, x, e_{|P_1|}, s) + w(e) \cdot z + f(e'_1, t, e'_{|P_2|}, y) \right\}$$

Thus, using P_1 and P_2, we can get the information for P using a constant amount of work. We now describe our algorithm. Similarly, using depth 1, and work $|P|$, we can find the corresponding maximum weight matchings for P, given the maximum weight matchings for P_1 and P_2. Analogously, we can also give such a dynamic program for a cycle obtained by concatenating two paths.

Claim 6.1. *There exists a parallel algorithm for computing a maximum weight matching on a degree two graph in $O(n \cdot \log n)$ work and $O(\log^2 n)$ depth.*

Proof. The algorithm proceeds by randomly concatenating paths. At any stage, the algorithm will maintain a collection of paths $\mathcal{P}$. Initially, $\mathcal{P} = \{P(u, u) \mid u \in V\}$. These are just empty paths corresponding to every vertex $u \in V$, and with the endpoints of the paths being u. As the algorithm proceeds, $\mathcal{P}$ is updated as follows. Let $\mathcal{V}$ be the collection of all endpoints of a path. Initially, $\mathcal{V} = V$. For all $u \in \mathcal{V}$, toss a coin. We can do this in parallel. Consider any edge $e = uv$ such that $u, v \in \mathcal{V}$. If the results of coin tosses of u and v are opposite, then we combine the paths $P_u = ((u_0, u_1) = e_1, \cdots, e_{|P_u|} = (u_{|P_u|-1}, u))$ and $P_v = (e'_1 = (v, v_1), \cdots, e'_{|P_v|} = (v_{|P_v|-1}, v_{|P_v|}))$ as follows:

1. In the collection of paths, remove $P[u_0, u]$ and $P[v, v_{|P_v|}]$ and add $P[u_0, v_{|P_v|}]$, which is the concatenation of $P_u \oplus e \oplus P_v$.
2. We also update f as follows, for all $x, y \in \{0, 1\}$,

$$f(e_1, x, e'_{|P_v|}, y) = \max_{\substack{z \in \{0,1\} \\ 0 \leqslant s,t \leqslant 1-z}} \left\{ f(e_1, x, e_{|P_u|}, s) + w(e) \cdot z + f(e'_1, t, e'_{|P_v|}, y) \right\}$$

3. Additionally, in $O(|P|)$ time, we can also do a search version of the above dynamic program to maintain the four candidate matchings which realize $f(e_1, x, e'_{|P_v|}, y)$ for $x, y \in \{0, 1\}$

Now, want to argue about the depth and work. First, with high probability, we have $\log n$ stages. Additionally, within each stage, with high probability, we will have to contract $O(\log n)$ paths. Thus, total depth is $O(\log^2 n)$. The total work done in each stage is proportional to the total lengths of the paths in $\mathcal{P}$. Thus, the total work is $O(n \cdot \log n)$. ∎

Proof of Theorem 6.5. Let $\tilde{E}_i$ be as defined in Algorithm 3. Let $\mathcal{A}$ be the parallel algorithm specified in the premise of the theorem. We consider $G_i = (V, \tilde{E}_i)$ for $i \in [L]$, and run $\mathcal{A}_i$ on G_i to compute M_i, which is the $(1 - \varepsilon)$-approximate maximum weight matching in G_i. Then, for all odd $i \in [L]$ we find a greedy census matching M_{odd}. We do the same for all even $i \in [L]$, to get M_{even}. Since the aspect ratio in G_i is ε^{-5}, the first step takes work $B(n, m, \varepsilon^{-5}, \varepsilon) \cdot \log W$ work and depth $D(n, m, \varepsilon^{-5}, \varepsilon)$. The second step can be implemented in total work $B(n, m, \varepsilon^{-5}, \varepsilon) \cdot \log W$. The depth for the second step is $\log W$, since we are only greedily combining $\log W$ matchings. Finally, we want to compute a maximum matching on the graph $M_{even} \cup M_{odd}$. Thus, we can find a matching in this graph by applying Claim 6.1. Thus, we are able to compute a compute a $(1 - \varepsilon)$-approximate maximum weight matching in $O(B(n, m, \varepsilon^{-5}, \varepsilon) \cdot \log W + n \log n)$ work and $O(D(n, m, \varepsilon^{-5}, \varepsilon) + \log W + \log^2 n)$ depth. ∎

7 Open Problems

Our reductions go a long way toward showing weighted and unweighted matching have the same complexity in a wide variety of models. By reducing the multiplicative overhead to poly$(1/\varepsilon)$, we are able to achieve this equivalence even for small approximation parameter ε. There are, however, a few limitations that we need to overcome.

1. A limitation of *all* existing reductions from weighted to unweighted matching in dynamic graphs is that they incur a large approximation error in non-bipartite graphs: in particular, both our Theorem 1.1 and the reduction of [11] reduce the approximation guarantee by $2/3-\varepsilon$. Achieving a general reduction for non-bipartite graphs that only loses a $(1-\varepsilon)$-factor is probably the main open problem in the area, and would be very interesting even with an update-time overhead that is exponential in ε. A similar open problem is to achieve such a reduction for other models, including a reduction for streaming and MPC that does not increase the number of passes/rounds. Note that both our Theorem 1.2 and the reduction of [29] already apply to non-bipartite graphs, so we can safely assume that weights are small integers. The only remaining challenge is thus that the framework of [11] uses an earlier tool called *graph unfolding* (first given by [33]) to reduce from small weights to unit weights, but this tool relies on the vertex-cover dual and seems limited to bipartite graphs.

2. A second limitation is specific to our paper: as discussed in the introduction, our reduction only works for $(1-\varepsilon)$-approximate matching and not for general α-approximate matching. Removing this restriction would show that in bipartite graphs at least, unweighted and weighted matching have almost equivalent complexities in a wide variety of computational models.

Acknowledgements

We thank the anonymous reviewers and David Wajc for their helpful feedback and discussions.

References

[1] Kook Jin Ahn and Sudipto Guha. Laminar families and metric embeddings: Non-bipartite maximum matching problem in the semi-streaming model. *CoRR*, abs/1104.4058, 2011. Available at https://arxiv.org/abs/1104.4058.

[2] Kook Jin Ahn and Sudipto Guha. Near linear time approximation schemes for uncapacitated and capacitated b-matching problems in nonbipartite graphs. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 239–258. SIAM, 2014. Available at https://arxiv.org/abs/1307.4355.

[3] Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. *ACM Trans. Parallel Comput.*, 4(4):17:1–17:40, 2018. Available at https://arxiv.org/abs/1307.4359.

[4] Abhash Anand, Surender Baswana, Manoj Gupta, and Sandeep Sen. Maintaining approximate maximum weighted matching in fully dynamic graphs. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012*, volume 18 of *LIPIcs*, pages 257–266. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. Available at https://arxiv.org/abs/1207.3976.

[5] Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In *Proc. 45th Int. Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPIcs*, pages 7:1–7:16, 2018. Available at https://arxiv.org/abs/1711.06625.

[6] Sepehr Assadi. A simple $(1-\varepsilon)$-approximation semi-streaming algorithm for maximum (weighted) matching. In *2024 Symposium on Simplicity in Algorithms, SOSA 2024*, pages 337–354. SIAM, 2024. Available at https://arxiv.org/abs/2307.02968.

[7] Sepehr Assadi, Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Semi-streaming bipartite matching in fewer passes and optimal space. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 627–669. SIAM, 2022. Available at https://arxiv.org/abs/2011.03495.

[8] Sepehr Assadi, S. Cliff Liu, and Robert E. Tarjan. An auction algorithm for bipartite matching in streaming and massively parallel computation models. In *4th Symposium on Simplicity in Algorithms, SOSA 2021*, pages 165–171. SIAM, 2021.

[9] Amir Azarmehr, Soheil Behnezhad, and Mohammad Roghani. Fully dynamic matching: $(2-\sqrt{2})$-approximation in polylog update time. In *Proceedings of the 35th ACM-SIAM Symposium on Discrete Algorithms, SODA 2024*, 2024. Available at https://arxiv.org/abs/2307.08772.

[10] Soheil Behnezhad. Dynamic algorithms for maximum matching size. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 129–162. SIAM, 2023. Available at https://arxiv.org/abs/2207.07607.

[11] Aaron Bernstein, Aditi Dudeja, and Zachary Langley. A framework for dynamic matching in weighted graphs. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing, STOC 2021*, pages 668–681. ACM, 2021.

[12] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *61st IEEE Annual Symposium on*

Foundations of Computer Science, FOCS 2020, pages 1123–1134. IEEE, 2020. Available at https://arxiv.org/abs/2009.02584.

[13] Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015*, volume 9134 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2015. Available at https://arxiv.org/abs/1506.07076.

[14] Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 692–711. SIAM, 2016.

[15] Sayan Bhattacharya and Peter Kiss. Deterministic rounding of dynamic fractional matchings. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 27:1–27:14, 2021. Available at https://arxiv.org/abs/2105.01615.

[16] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic $(1+\varepsilon)$-approximate matching size in truly sublinear update time. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, pages 1563–1588. IEEE, 2023. Available at https://arxiv.org/abs/2302.05030.

[17] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic algorithms for packing-covering lps via multiplicative weight updates. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 1–47. SIAM, 2023. Available at https://arxiv.org/abs/2207.07519.

[18] Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 100–128. SIAM, 2023. Available at https://arxiv.org/abs/2207.07438.

[19] Sayan Bhattacharya, Peter Kiss, Aaron Sidford, and David Wajc. Near-optimal dynamic rounding of fractional matchings in bipartite graphs. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 59–70. ACM, 2024. Available at https://arxiv.org/abs/2306.11828.

[20] Joakim Blikstad and Peter Kiss. Incremental $(1-\varepsilon)$-approximate dynamic matching in $O(\text{poly}(1/\varepsilon))$ update time. In *31st Annual European Symposium on Algorithms, ESA 2023*, volume 274 of *LIPIcs*, pages 22:1–22:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. Available at https://arxiv.org/abs/2302.08432.

[21] Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial worst-case time barrier. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 33:1–33:14, 2018. Available at https://arxiv.org/abs/1711.06883.

[22] Jiale Chen, Aaron Sidford, and Ta-Wei Tu. Entropy regularization and faster decremental matching in general graphs. In *arXiv Preprint*, 2023. Available at https://arxiv.org/abs/2312.09077.

[23] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 612–623, 2022. Available at https://arxiv.org/abs/2203.00671.

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

[25] Aditi Dudeja. A note on rounding matchings in general graphs. In *arXiv Preprint*, 2024. Available at https://arxiv.org/abs/2402.03068.

[26] Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019*, pages 491–500. ACM, 2019. Available at https://arxiv.org/abs/1811.02760.

[27] Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1+\varepsilon)$-approximate incremental matching in constant deterministic amortized time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1886–1898. SIAM, 2019.

[28] Manoj Gupta. Maintaining approximate maximum matching in an incremental bipartite graph in polylogarithmic update time. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014*, volume 29 of *LIPIcs*, pages 227–239, 2014.

[29] Manoj Gupta and Richard Peng. Fully dynamic $(1+\varepsilon)$-approximate matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013*, pages 548–557. IEEE Computer Society, 2013. Available at https://arxiv.org/abs/1304.0378.

[30] Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *Algorithmica*, 76(3):654–683, 2016. Available at https://arxiv.org/abs/1212.6925.

[31] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing, STOC 2015*, pages 21–30, 2015. Available at https://arxiv.org/abs/1511.06773.

[32] Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Regularized box-simplex games and dynamic decremental bipartite matching. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022*, volume 229 of *LIPIcs*, pages 77:1–77:20, 2022. Available at https://arxiv.org/abs/2204.12721.

[33] Ming-Yang Kao, Tak Wah Lam, Wing-Kin Sung, and Hing-Fung Ting. A decomposition theorem for maximum weight bipartite matchings. *SIAM J. Comput.*, 31(1):18–26, 2001.
[34] Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 1874–1893. SIAM, 2021. Available at https://arxiv.org/abs/2103.11669.
[35] Quanquan C. Liu, Yiduo Ke, and Samir Khuller. Scalable auction algorithms for bipartite maximum matching problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2023*, volume 275 of *LIPIcs*, pages 28:1–28:24, 2023. Available at https://arxiv.org/abs/2307.08979.
[36] Yang P. Liu. On approximate fully-dynamic matching and online matrix-vector multiplication. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024*. IEEE, 2024. Available at https://arxiv.org/abs/2403.02582.
[37] Noam Solomon and Shay Solomon. A generalized matching reconfiguration problem. In *12th Innovations in Theoretical Computer Science Conference, ITCS 2021*, volume 185 of *LIPIcs*, pages 57:1–57:20, 2021. Available at https://arxiv.org/abs/1803.05825.
[38] Daniel Stubbs and Virginia Vassilevska Williams. Metatheorems for dynamic weighted matching. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017*, volume 67 of *LIPIcs*, pages 58:1–58:14, 2017.
[39] Jan van den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P. Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, pages 503–514, 2023. Available at https://arxiv.org/abs/2309.16629.
[40] David Wajc. Rounding dynamic matchings against an adaptive adversary. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 194–207. ACM, 2020. Available at https://arxiv.org/abs/1911.05545.

A Analysis of [11]

We first give the definitions and notations used in the statement of the algorithm and proof.

Definition A.1. ([33]) *Let G be a graph with integer edge weights in $[W]$. The unfolded graph $\phi(G)$ is an unweighted graph defined as follows: For each vertex $u \in G$, there are W copies of u, $\{u^1, u^2, \ldots, u^W\}$, in $\phi(G)$. Corresponding to each edge uv in G there are w_{uv} edges $\left\{u^i v^{w_{uv}-i+1}\right\}_{i\in[w_{uv}]}$ in $\phi(G)$.*

A simple consequence of the above definition is the following observation.

Observation A.1. *Let G be any weighted bipartite graph, and suppose W is the ratio between the maximum and minimum edge weights, then $|V(\phi(G))| = W \cdot n$ and $|E(\phi(G))| = W \cdot m$.*

Fact A.1. ([33]) *Let G be a weighted bipartite graph, and suppose M is the maximum weight matching of G and let M_ϕ be the MCM of $\phi(G)$. Then, $w(M) = |M_\phi|$.*

Definition A.2. *Let G be a weighted graph, and let $H \subseteq \phi(G)$. The refolded graph $\mathcal{R}(H)$ has vertex set $V(G)$, and edges $E(\mathcal{R}(H)) = \left\{uv \in G \mid u^i v^j \in H \text{ for } i + j + 1 = w(uv)\right\}$.*

Fact A.2. ([11]) *Let G be a weighted graph with weight function w and let M be an α-approximate matching of $\phi(G)$. If G is bipartite, then $\mu_w(\mathcal{R}(M)) \geq \alpha\mu_w(G)$. If G is not bipartite, then $\mu_w(\mathcal{R}(M)) \geq \frac{2}{3}\alpha\mu_w(G)$.*

Now we modify Algorithm 1 of [11] to achieve a better amortized update time.

Lemma 5.13. ([11, 33]) *For $\varepsilon \leq 1/6$, given a dynamic algorithm $\mathcal{A}$ that, on input n-vertex m-edge **bipartite** graph, initializes in $\mathcal{I}(n, m, \varepsilon)$ time and explicitly maintains an $(1-\varepsilon)$-approximate MCM in $\mathcal{U}(n, m, \varepsilon)$ update time, there is a dynamic algorithm that initializes in*

$$O(\mathcal{I}(nW, mW, \varepsilon) + m \log(\varepsilon^{-1})\varepsilon^{-1})$$

time and explicitly maintains an $(1-\varepsilon)$-approximate MWM on a bipartite graph with integer edge weights bounded by W in

$$O(W \cdot \mathcal{U}(nW, mW, \Theta(\varepsilon))) + W \log(\varepsilon^{-1})\varepsilon^{-2})$$

Algorithm 6: Bipartite Reduction

Input: A dynamic algorithm for $(1-\varepsilon)$-approximate MCM in bipartite graphs $\mathcal{A}_u$

function `Initialize()`
- Initialize $\mathcal{A}_u$ with the unfolded graph $\phi(G)$.
- Denote M_u as the matching maintained by $\mathcal{A}_u$.
- $M \leftarrow$ `Rebuild()`.
- **output** M.

function `Update(`(uv)`)`
- Update $(u^i, v^{w_{uv}-i})$ in $\phi(G)$ for $i \in [w_{uv}]$ accordingly.
- Use $\mathcal{A}_u$ to maintain a matching M_u of $\phi(G)$.
- $c \leftarrow c + 1$.
- **if** $c < \varepsilon \cdot W^*/W$ **then**
  - $M \leftarrow M \setminus uv$.
- **else**
  - $M \leftarrow$ `Rebuild()`.
- **output** M.

function `Rebuild()`
- $M \leftarrow$ a $(1-\varepsilon)$-approximate MWM on the refolded graph $\mathcal{R}(M_u)$.
- $c \leftarrow 0$, $W^* \leftarrow w(M)$.
- **output** M.

amortized update time and has amortized recourse

$$O(W \log(\varepsilon^{-1})\varepsilon^{-2}).$$

The transformation is partially dynamic preserving. On non-bipartite graphs, the approximation ratio is $\frac{2}{3} - \varepsilon$.

Proof. We first prove that Algorithm 6 maintains a $(1-O(\varepsilon))$-approximate MWM on a bipartite G. Since M_u is a $(1-\varepsilon)$-approximate matching, thus by Fact A.2, $\mu_w(\mathcal{R}(M_u)) \geq (1-\varepsilon)\mu_w(G)$. After each `Rebuild`, M is a $(1-\varepsilon)$-approximate MWM on $\mathcal{R}(M_u)$, thus $W^* = w(M) \geq (1-\varepsilon)\cdot\mu_w(R(M_u)) \geq (1-2\varepsilon)\cdot\mu_w(G)$. Between `Rebuild`, there are at most $\varepsilon\cdot W^*/W$ edge updates, thus $w(M) \geq (1-\varepsilon)W^*$ and $\mu_w(G) \leq W^*/(1-2\varepsilon)+\varepsilon\cdot W^* \leq (1+4\varepsilon)W^*$, and $w(M) \geq (1-5\varepsilon)\mu_w(G)$. On a general graph, according to Fact A.2, $\mu_w(\mathcal{R}(M_u)) \geq \frac{2}{3}(1-\varepsilon)\mu_w(G)$ and similarly we can prove that $w(M) \geq (\frac{2}{3} - O(\varepsilon))\mu_w(G)$.

Now we analyze the running time of Algorithm 6 for both bipartite and non-bipartite graphs. By Theorem 2.1, the initialization can be implemented in time

$$O(\mathcal{I}(nW, mW, \varepsilon) + m \log(\varepsilon^{-1})\varepsilon^{-1}).$$

The running time of $\mathcal{A}_u$ is $W \cdot \mathcal{U}(nW, mW, \Theta(\varepsilon))$. Since

$$|\mathcal{R}(M_u)| \leq |M_u| \leq \mu(\phi(G)) = \mu_w(G) \leq O(1)\cdot W^*,$$

each `Rebuild` takes total time $O(|\mathcal{R}(M_u)| \log(\varepsilon^{-1})\varepsilon^{-1}) = O(W^* \log(\varepsilon^{-1})\varepsilon^{-1})$ by Theorem 2.1. Thus the amortized cost of `Rebuild` and the amortized recourse of the algorithm is

$$O(W^* \log(\varepsilon^{-1})\varepsilon^{-1})/(\varepsilon \cdot W^*/W) = O(W \log(\varepsilon^{-1})\varepsilon^{-2}).$$

□

B Counterexample to Question 3.1

To answer Question 3.1 in the negative, we prove the following claim:

Claim B.1. *There is a graph G such that for* any *weight partition of G, $[\ell_1, r_1), [\ell_2, r_2), \ldots, [\ell_k, r_k) \subseteq \mathbb{R}$, the following holds: if all possible choices of MWM M_i of $G_{[\ell_i, r_i)}$ satisfy*

$$\mu_w(M_1 \cup M_2 \cup \cdots \cup M_k) \geq (1-\delta) \cdot \mu_w(G),$$

then there is a weight class $[\ell_i, r_i)$ such that $r_i \geq \ell_i \cdot \exp(\Omega(\delta^{-1}))$.

To prove Claim B.1, we will use the following gadget to explicitly build the graph G.

Definition B.1. (Level-i Gadget) *A level-i Gadget is a path with three edges a_i, b_i, c_i such that the edge weights of a_i and b_i are 1.5^i, and the edge weight of c_i is 1.5^{i+1}.*

Proof Intuition: The MWM on a level-i gadget is $1.5^i + 1.5^{i+1} = 1.5^i \cdot 2.5$ by choosing a_i and c_i. But now let us consider what happens if the gadget is "broken", meaning that it is partitioned into two different weight classes. More concretely, say that $a_i, b_i \in [\ell_j, r_j)$, while $c_i \in [\ell_{j+1}, r_{j+1})$. Then, one valid MWM of weight class $[\ell_j, r_j)$ is $M_j = \{b_i\}$, and clearly we have $M_{j+1} = \{c_i\}$. As a result, $\mu_w(M_j \cup M_{j+1}) = w(c_i) = 1.5^{i+1} \leq \frac{3}{5} \cdot \mu_W(\{a_i, b_i, c_i\})$. In other words, the loss incurred by a broken gadget is a constant fraction of the weight of the gadget. Intuitively, we will have gadgets on different levels, and any partition into k weight classes will break $k-1$ of the gadgets. Since we can only afford a total loss of only $\delta \mu_w(G)$, the average weight class $[\ell_i, r_i]$ must contain at least $\Omega(\delta^{-1})$ non-broken gadgets to make up for the loss of the broken ones. Since each gadget is 1.5-wide, this implies that the average weight class must be $1.5^{\Omega(\delta^{-1})}$-wide. We now formally prove Claim B.1.

Proof. We first build the graph G using our gadgets. For $i = 0, 1, \ldots, N$, where $N = \lfloor \log_{1.5} W \rfloor$, G contains 1.5^{N-i} level-i gadgets. Therefore $\mu_w(G) = 1.5^N \cdot 2.5 \cdot N$ since the total MWM of each level is $1.5^{N-i} \cdot 1.5^i \cdot 2.5 = 1.5^N \cdot 2.5$. Now consider any weight partition $[\ell_1, r_1), \ldots [\ell_k, r_k)$. We assume that for any weight class $[\ell_i, r_i)$, there is some integer j such that $\ell_i \leq 1.5^j < r_i$. This is w.l.o.g. since otherwise that weight class does not contain any edge and we can merge it with one of its neighboring weight classes.

$k-1$ levels of broken gadgets can be found in this graph. For $i = 0, 1, 2, \ldots, k-1$, consider the largest j such that $1.5^j < r_i$ and level-j gadgets will be broken. On a broken level-j gadget, the MWM will be 1.5^{j+1} instead of $1.5^j \cdot 2.5$. Since there are 1.5^{N-j} level-j gadgets, the weight loss of all level-j gadgets is 1.5^N. Therefore the total weight loss is $1.5^N \cdot (k-1)$.

Suppose the MWM on the union has weight at least $(1-\delta) \cdot \mu_w(G) = (1-\delta) \cdot 1.5^N \cdot 2.5 \cdot N$. Then we have $1.5^N \cdot (k-1) \leq \delta \cdot 1.5^N \cdot 2.5 \cdot N$ meaning $k \leq 2.5 \cdot \delta \cdot N + 1$. Therefore the would be a weight class $[\ell_i, r_i)$ with $r_i \geq \ell_i \cdot 1.5^{\frac{N}{2.5 \cdot \delta \cdot N + 1}} = \ell_i \cdot \exp(\Omega(\delta^{-1}))$. ◻

C Reduction of α-Approximation Requires Exponential Width

Claim C.1. *For any constant $\frac{1}{2} < \alpha < 1$, there is a graph G such that for* any *set of weight classes (not necessarily a weight partition), $[\ell_1, r_1), [\ell_2, r_2), \ldots, [\ell_k, r_k) \subseteq \mathbb{R}$, the following holds: if all possible choices of α-approximate MWM M_i of $G_{[\ell_i, r_i)}$ satisfy*

$$\mu_w(M_1 \cup M_2 \cup \cdots \cup M_k) \geq (\alpha - \delta) \cdot \mu_w(G),$$

then there is a weight class $[\ell_i, r_i)$ such that $r_i \geq \ell_i \cdot \exp(\Omega(\delta^{-1}))$.

We use a similar gadget for α-approximation as before.

Definition C.1. (Level-i Gadget for α-Approximation) *A level-i Gadget for α-approximation is a path with three edges a_i, b_i, c_i such that the edge weights of a_i and b_i are β^i, and the edge weight of c_i is β^{i+1}, where $\frac{\beta}{\beta+1} = \alpha$.*

Proof. We construct a graph G that contains β^{N-i} number of level-i gadgets for each $0 \leq i \leq N-1$, where $N = \lfloor \frac{\alpha - \alpha^2 - (1-\alpha)^2}{\delta} - 1 \rfloor$. Thus $\mu_w(G) = N \cdot \beta^N \cdot (1+\beta)$. We will construct a sparsifier $S \subseteq G$ such that for any weight class $[\ell_i, r_i)$ that doesn't contain the entire graph, there is an α-approximate matching M_i of $G_{[\ell_i, r_i)}$ in S. That means if no weight classes in $[\ell_1, r_1), [\ell_2, r_2), \cdots, [\ell_k, r_k) \subseteq \mathbb{R}$ contain the entire graph, there is a choice of matchings such that $M_1 \cup M_2 \cup \cdots \cup M_k \subseteq S$. On the other hand, the construction of S will ensure that $\mu_w(S) < (\alpha - \delta) \cdot \mu_w(G)$ suggesting that at least one weight class contains the entire graph and thus has $\exp(\Omega(\delta^{-1}))$ width.

Now we explicitly construct S. For all $0 \leq i \leq N-2$, S contains all b_i and c_i edges in level-i gadgets. Also, S contains all b_{N-1} edges and α fraction of c_{N-1} edges in level-$(N-1)$ gadgets. Consider any weight class $[l, r)$.

1. Suppose $r < \beta^N$. Let $j = \lfloor \log_\beta r \rfloor$. By picking all c_i edges in any corresponding level $i < j$ intersecting with the weight class and all b_j edges in level j, there is an α-approximate MWM.
2. Suppose $r \geq \beta^N$. If $l > \beta^{N-1}$, then S clearly contains an α-approximation. Otherwise, if $l > 1$, by picking all c_i edges in any level $i < N-1$, there is an α-approximate MWM.

So far, we have shown that if none of the weight classes contains G, S can consistently output an α-approximate MWM. However,

$$\mu_w(S) = (N-1) \cdot \beta^{N+1} + \alpha \cdot \beta^{N+1} + (1-\alpha) \cdot \beta^N,$$

and the largest matching in S has approximation ratio

$$\frac{(N-1) \cdot \beta^{N+1} + \alpha \cdot \beta^{N+1} + (1-\alpha) \cdot \beta^N}{N \cdot \beta^N \cdot (1+\beta)} = \alpha - \frac{1}{N}\left(\alpha - \alpha^2 - (1-\alpha)^2\right) < \alpha - \delta.$$

□