# A Framework for Dynamic Matching in Weighted Graphs

Aaron Bernstein*
bernstei@gmail.com
Rutgers University
New Brunswick, NJ, USA

Aditi Dudeja
aditi.dudeja@rutgers.edu
Rutgers University
New Brunswick, NJ, USA

Zachary Langley
zach.langley@rutgers.edu
Rutgers University
New Brunswick, NJ, USA

## ABSTRACT

We introduce a new framework for computing approximate maximum weight matchings. Our primary focus is on the fully dynamic setting, where there is a large gap between the guarantees of the best known algorithms for computing weighted and unweighted matchings. Indeed, almost all current weighted matching algorithms that reduce to the unweighted problem lose a factor of two in the approximation ratio. In contrast, in other sublinear models such as the distributed and streaming models, recent work has largely closed this weighted/unweighted gap.

For bipartite graphs, we almost completely settle the gap with a general reduction that converts *any* algorithm for $\alpha$-approximate unweighted matching to an algorithm for $(1-\varepsilon)\alpha$-approximate weighted matching, while only increasing the update time by an $O(\log n)$ factor for constant $\varepsilon$. We also show that our framework leads to significant improvements for non-bipartite graphs, though not in the form of a universal reduction. In particular, we give two algorithms for weighted non-bipartite matching:

(i) A randomized (Las Vegas) fully dynamic algorithm that maintains a $(1/2-\varepsilon)$-approximate maximum weight matching in worst-case update time $O_\varepsilon(\text{polylog } n)$ with high probability against an adaptive adversary. Our bounds are essentially the same as those of the unweighted algorithm of Wajc [STOC 2020].

(ii) A deterministic fully dynamic algorithm that maintains a $(2/3-\varepsilon)$-approximate maximum weight matching in amortized update time $\tilde{O}_\varepsilon(m^{1/4})$. Our bounds are essentially the same as those of the unweighted algorithm of Bernstein and Stein [SODA 2016].

A key feature of our framework is that it uses existing algorithms for unweighted matching as black-boxes. As a result, our framework is simple and versatile. Moreover, our framework easily translates to other models, and we use it to derive new results for the weighted matching problem in streaming and communication complexity models.

*This work was done while funded by NSF CAREER Grant 1942010.


*STOC '21, June 21–25, 2021, Virtual, Italy*

ACM ISBN 978-1-4503-8053-9/21/06…$15.00
https://doi.org/10.1145/3406325.3451113

## CCS CONCEPTS

• **Theory of computation → Dynamic graph algorithms**; *Sparsification and spanners.*

## KEYWORDS

Dynamic Algorithms, Dynamic Matching, Matching Sparsifiers, Weighted Matching, Adaptive Adversary

**ACM Reference Format:**

Aaron Bernstein, Aditi Dudeja, and Zachary Langley. 2021. A Framework for Dynamic Matching in Weighted Graphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC '21), June 21–25, 2021, Virtual, Italy*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3406325.3451113

## 1 INTRODUCTION

Computing a maximum matching is a fundamental problem in graph algorithms that has many applications. In the unweighted version, known as the maximum *cardinality* matching problem, the goal is to find a matching with the largest number of edges. In the weighted version, each edge $e$ has a positive real weight $w(e)$, and the goal is to find a matching maximizing the sum of its edge weights. This paper primarily focuses on the *fully dynamic model*, where the algorithm must maintain a matching in a graph that undergoes edge insertions and deletions. The goal is to minimize the update time, i.e., the time to process a single edge-change to the graph. For exact matching, there is a conditional lower bound of $\Omega(m^{1/2})$ update time [1, 24, 30], and thus, most research on dynamic matching focuses on maintaining an *approximate* maximum matching.

There is a vast literature on approximate fully dynamic matching in unweighted graphs. The state-of-the-art covers a large range of different trade-offs between update times and approximation ratios, as well as secondary features such as whether the algorithm is amortized/worst-case or deterministic/randomized. Progress on weighted matching lags far behind.

A general reduction of Stubbs and Williams [32] shows how to convert any $\alpha$-approximate algorithm for unweighted matching into a $(1/2-\varepsilon)\alpha$-approximate algorithm for weighted matching with nearly the same update time. Almost every state-of-the-art algorithm for weighted matching comes from applying this reduction to existing unweighted algorithms. As far as we know, the only exception is an algorithm of Gupta and Peng [23] with an $O(\sqrt{m})$ update time and a $(1-\varepsilon)$-approximation ratio, which can be extended to weighted graphs with essentially the same bounds. Except for Gupta and Peng's result, all algorithms for weighted

graphs have twice as large an approximation error as those for unweighted graphs. This gap is especially significant in the dynamic matching setting, where approximation errors for unweighted algorithms are typically small constants. Moreover, a 1/2-approximation is considered a fundamental barrier for matching, and one cannot use the Stubbs-Williams framework [32] to surmount this barrier in weighted graphs.

By contrast, in related models such as streaming or distributed algorithms, there are several recent algorithms for weighted matching that match the best known results for unweighted matching. Closing the weighted/unweighted gap in dynamic matching remains a fundamental open problem, posed explicitly by Stubbs and Williams [32] and Wajc [33].

## 1.1 Our Contribution

For all the results below, let $R$ be the ratio between the heaviest edge weight ever present in the graph and the lightest such edge weight.

*Bipartite graphs.* In bipartite graphs, we effectively close the gap between weighted and unweighted graphs by showing a black-box reduction from dynamic weighted matching to dynamic unweighted matching.

**Result 1** (See Theorem 3.1)**.** *Let $\varepsilon > 0$ be some fixed constant. If there is a fully dynamic algorithm $\mathcal{A}_u$ that maintains an $\alpha$-approximate maximum* cardinality *matching with update time $T_u$, then there is a fully dynamic algorithm $\mathcal{A}_w$ that maintains a $(1-\varepsilon)\alpha$-approximate maximum* weight *matching with update time $O_\varepsilon(T_u \cdot \log R)$. If $\mathcal{A}_u$ is deterministic, worst-case, and/or works against an adaptive adversary, then the same is true of $\mathcal{A}_w$.*

Our reduction immediately implies many new algorithms for weighted matching, obtaining essentially the same bounds as those for unweighted matching. The black-box nature of our reduction also makes it highly relevant for future work. We highlight three results in particular:

(1) One of the biggest successes of dynamic unweighted matching is $(1/2-\varepsilon)$-approximate algorithms with polylog$(n)$ update time. For example, Bhattacharya, Henzinger, Nanongkai [15] gave an algorithm that is amortized and deterministic, and Wajc [33] gave an algorithm that is worst-case and randomized (and works against adaptive adversaries). Our reduction extends both of these results to weighted graphs. Previously, the analogous results for weighted graphs had an approximation ratio of $(1/4-\varepsilon)$. (There are also various unweighted results that achieve a strict 1/2-approximation [6, 31] or *constant* update time [12, 31]; our reduction cannot extend these to weighted graphs due to the additional $(1-\varepsilon)$ factor in the approximation error and the $\log(R)$ factor in the update time.)
(2) Bernstein and Stein [10] gave a deterministic algorithm to maintain a $(2/3-\varepsilon)$-approximate maximum weight matching in an unweighted bipartite graph with worst-case update time $O(m^{1/4})$. Our reduction implies essentially the bounds for weighted graphs. Previously, the analogous result for weighted graphs had an approximation ratio of $(1/3-\varepsilon)$.
(3) Behnezhad, Łącki, and Mirrokni [7] and Wajc [33] both showed algorithms that achieve a better-than-1/2 approximation with arbitrarily small polynomial update time. Formally, for any fixed $\delta > 0$, there is an algorithm that maintains a $(1/2+\Omega(1))$-approximate maximum cardinality matching with update time $O(n^\delta)$. Our reduction leads to the first such results for *weighted* matching, though unlike the unweighted algorithm of [7], our extension is limited to bipartite graphs. (The algorithm of [7] actually has update time $O(\Delta^\delta)$, where $\Delta$ is the maximum degree; it is easy to check that our reduction preserves $\Delta$ up to a constant and therefore has the same guarantee.)

*Non-bipartite graphs.* Our techniques also lead to a general framework for dynamic weighted matching in non-bipartite graphs, though it is not a universal reduction like Result 1. In particular, the first two highlighted items above apply to non-bipartite graphs as well.

**Result 2** (See Theorem 5.11)**.** *For any fixed $\varepsilon > 0$, there exists a fully dynamic algorithm that maintains a $(1/2-\varepsilon)$-approximate maximum weight matching in update time $O_\varepsilon(\log^5(n)\log(R))$. The algorithm is worst-case, randomized (Las Vegas), and works against an adaptive adversary.*

**Result 3** (See Theorem 5.12)**.** *For any fixed $\varepsilon > 0$, there exists a fully dynamic algorithm that maintains a $(2/3-\varepsilon)$-approximate maximum weight matching in update time $O_\varepsilon(m^{1/4}\log(R))$. The algorithm is deterministic and amortized.*

Our two results each use an algorithm for unweighted matching as a black box: Result 2 uses the algorithm of Wajc [33], while Result 3 uses the algorithm of Bernstein and Stein [11]. Both results essentially match the guarantees of these unweighted algorithms. Previous weighted algorithms had twice as high an approximation error.

*Additional factors in the update time.* All of our guarantees match those of the corresponding unweighted algorithms almost exactly. There are two minor caveats. Firstly, the update time is multiplied by $O(\log(R))$, which is $O(\log(n))$ as long as weights are polynomial in $n$. We can use the black box of Stubbs and Williams [32] to reduce the dependence on $R$ to $\log\log(R)$; see Appendix A for details. Secondly, our algorithms have a significantly worse dependence on $\varepsilon$: the update time is multiplied by $(1/\varepsilon)^{O(1/\varepsilon)}$. This term is still $O(1)$ for any fixed $\varepsilon$.

## 1.2 Techniques

*Bipartite graphs.* Our bipartite result, Result 1, is technically simple; our main contribution here is showing how to combine ideas from disparate parts of the matching literature.

We first observe that a bucketing scheme of Gupta and Peng [23], while not presented as such, implies a black-box reduction from general weights to integer weights in the range $[1,(1/\varepsilon)^{O(1/\varepsilon)}]$. We then show that in bipartite graphs, a technique called *graph unfolding* by Kao, Lam, Sung and Ting [25] allows us to efficiently transform a graph with small weights into an unweighted graph $\phi(G)$ such that size of the maximum *cardinality* matching of $\phi(G)$ is equal to the weight of the maximum *weight* matching in $G$. We

can then use any unweighted matching algorithm $\mathcal{A}_u$ as a black-box to compute a matching $M_u$ in $\phi(G)$. Finally, we show that the matching $M_u$ of $\phi(G)$ can be *refolded* into a low-degree subgraph $\mathcal{R}(M_u)$ of $G$ that contains a high-weight matching; we maintain the maximum weight matching of $\mathcal{R}(M_u)$ using a simple algorithm for low-degree graphs.

*Non-bipartite graphs.* Our main technical contribution is for non-bipartite graphs. We rely on the same tools above, but the challenge is that in non-bipartite graphs, refolding a matching in $\phi(G)$ might incur an extra $2/3$-factor loss in the approximation ratio; this error is analogous to the integrality gap of non-bipartite graphs. To avoid this loss, we observe that several existing algorithms for unweighted graphs do not just compute a matching, but rather compute a subgraph $H$ with properties that guarantee that $H$ contains a large matching. The algorithm of Wajc [33] maintains a subgraph called a *kernel* (introduced in [14]), while the algorithm of Bernstein and Stein [10, 11] maintains a subgraph called an *edge-degree-constrained subgraph (EDCS)*.

The key to our non-bipartite results are two new structural theorems that show that kernels and EDCS are *refolding-approximate*, which, loosely speaking, means that they continue to contain a large matching even after refolding. This means that they can be used in our new unfolding/refolding framework without incurring additional error. Our proofs of these structural theorems use the probabilistic method to effectively reduce the non-bipartite case to the bipartite one.

One advantage of our framework is that it does not require modifying existing unweighted algorithms; we simply run those algorithms as black-boxes in the unfolded graph $\phi(G)$ and show that the properties of these algorithms are preserved. As a result, our framework is simple and versatile.

### 1.3 Application to Other Models

None of the ingredients in our framework are specific to the dynamic model; graph unfolding was previously used only in the regular static setting [25], and our new theorems about refolding-approximate subgraphs are general structural claims. Our focus in this paper is on dynamic algorithms because, prior to our work, the dynamic setting had the most significant gap between the weighted and unweighted approximation ratios.

Indeed, our framework translates seamlessly to other models, and we use it to achieve new results for weighted matching in the streaming and communication complexity models. All of our results have essentially the same bounds as the state-of-the-art for unweighted matching; see Section 6 for details. We note that Gamlath, Kale, Mitrovic, and Svennson [21] also showed a reduction from weighted to unweighted matching in the streaming models. Their reduction has the advantage of applying more directly to non-bipartite graphs, but it is significantly more complex and thus works in a narrower range of settings. Most relevant to this paper, their reduction cannot be applied in the dynamic setting, and in the streaming setting, their reduction increases the number of passes by a large constant, whereas ours preserves the number of passes exactly.

## 2 PRELIMINARIES

Let $G = (V, E, w)$ be a graph with edge weight function $w : E \to \mathbb{R}$. We use $n = |V|$ to denote the number of vertices of $G$ and $m = |E|$ the number of edges. For any $U \subseteq V$, we denote by $\mathrm{N}_G(U)$ the set of neighbors of vertices in $U$ in $G$, and by $\mathrm{E}_G(U)$ the set of edges incident on $U$. For any $v \in V$, let $\deg_G(v)$ denote the degree of the vertex $v$ in $G$. For any positive integer $x$, we define $[x] = \{1, \ldots, x\}$.

Our results have an exponential dependence on $1/\varepsilon$. Throughout the paper, we will use $\gamma_\varepsilon := (4/\varepsilon)^{\lceil 4/\varepsilon \rceil}$, a large constant that we incur in update time through our reductions; see Appendix C for details. To simplify notation, we will use $O_\varepsilon(\cdot)$ to supress the dependence on $\varepsilon$ (and therefore $\gamma_\varepsilon$).

*Dynamic graphs.* In the *fully dynamic setting*, the input is a weighted graph $G = (V, E, w)$ subject to a sequence of updates, where each update either inserts an edge into or deletes an edge from $G$. The goal is to maintain a large matching in $G$ while minimizing the *update time*, i.e., the time to process an update.

We use $R := \max_{e \in E} w(e) / \min_{e \in E} w(e)$ to be the ratio between the largest edge weight ever contained in $G$ and the smallest edge weight ever contained in $G$. In this paper, all the algorithms we obtain have update times that depend on $\log(R)$. The dependence can be reduced to $\log\log(R)$ via a black-box reduction of Stubbs and Williams [32]; see Appendix A for more details. We use $\Delta$ to denote the maximum degree ever attained by $G$.

Given a dynamic matching algorithm $\mathcal{A}$, we say that $\mathcal{A}$ has worst-case *recourse* $\sigma$ if every adversarial update causes $\mathcal{A}$ to make at most $\sigma$ most changes to the matching. We define amortized recourse analogously.

In addition to the update time and recourse, a dynamic algorithm may have several other features of interest, such as whether it works against an adaptive adversary, whether it is amortized or worst-case, and whether it is deterministic or randomized. We refer to these as the algorithm's *secondary features*.

*Matchings.* A *matching* $M$ is a subset of edges, no two of which share an endpoint. A *maximum cardinality matching* in $G$ is a matching of maximum possible size; we denote this maximum size by $\textsc{mcm}(G)$. For graphs $G$ with weighted edges, a *maximum weight matching* is a matching maximum the sum of its edge weights; we denote this maximum weight by $\textsc{mwm}(G)$. For $\alpha \in (0, 1]$, a matching $M$ is said to be an *$\alpha$-approximate maximum cardinality matching* if $|M| \geq \alpha \cdot \textsc{mcm}(G)$ and an *$\alpha$-approximate maximum weight matching* if $w(M) \geq \alpha \cdot \textsc{mwm}(G)$.

In our results, we use a deterministic algorithm that maintains a $(1 - \varepsilon)$-approximate maximum weight matching in a low-degree graph. The algorithm is a trivial extension of an existing unweighted algorithm of Gupta and Peng [23]; see Appendix B for formal proof.

**Theorem 2.1** ([23]). *Let $G$ be a dynamic weighted graph, with weights in range $[W]$. For $\varepsilon \in (0, 1)$, there exists a fully dynamic deterministic algorithm that maintains a $(1-\varepsilon)$-approximate maximum weight matching in $G$ with worst-case update time $O(\Delta W^2/\varepsilon^2)$.*

*Probabilistic tools.* We will use the following standard variant of the Chernoff bound.

**Proposition 2.2** (Chernoff bound). *Let $Z_1, \ldots, Z_n$ be negatively correlated random variables such that $Z_i \in [0,1]$ for all $i \in [n]$, and let $Z = \sum_{i=1}^{n} Z_i$. If $\mathbb{E}[Z] \leq \lambda$, then for any $\delta > 0$,*

$$\Pr\left(|Z - \mathbb{E}[Z]| \geq \delta \cdot \lambda\right) \leq 2 \cdot \exp\left(-\delta^2 \lambda / 3\right).$$

We also need the following version of the Lovász Local Lemma (LLL).

**Proposition 2.3** (Lovász Local Lemma). *Let $p \in (0,1)$ and let $d \geq 1$. Suppose $\mathcal{E}_1, \ldots, \mathcal{E}_t$ are $t$ events such that $\Pr(\mathcal{E}_i) \leq p$ for all $i \in [t]$ and each $\mathcal{E}_i$ is mutually independent of all $\mathcal{E}_j$ except (at most) $d$ other events $\mathcal{E}_j$. If $p \cdot (d+1) < 1/e$, then $\Pr\left(\bigcap_{i=1}^{n} \overline{\mathcal{E}_i}\right) > 0$.*

# 3 A NEW REDUCTION TO UNWEIGHTED MATCHINGS IN BIPARTITE GRAPHS

The main theorem of this section is the following formalization of Result 1, which leverages a black-box reduction from weighted bipartite matching to unweighted bipartite matching. Recall that $\gamma_\varepsilon = (4/\varepsilon)^{\lceil 4/\varepsilon \rceil}$; for any fixed $\varepsilon$, we have $\gamma_\varepsilon = O(1)$.

**Theorem 3.1.** *Let $G$ be a weighted bipartite graph and let $\varepsilon > 0$. If there is an algorithm $\mathcal{A}_u$ that maintains an $\alpha$-approximate maximum cardinality matching in a dynamic unweighted bipartite graph $G$ with update time $T_u(n, m, \alpha)$, and if $\mathcal{A}_u$ has recourse $\sigma_u(n, m, \alpha) = O(T_u(n, m, \alpha))$, then there exists an algorithm $\mathcal{A}_w$ that maintains a $(1-\varepsilon)\alpha$-approximate maximum weight matching in weighted bipartite graphs with update time $O_\varepsilon(T_u(\gamma_\varepsilon n, \gamma_\varepsilon m, \alpha) \cdot \log(R))$, where $R$ is the ratio between the heaviest edge weight ever present in the graph and the lightest such edge weight. Further, the secondary features of $\mathcal{A}_w$ are same as those of $\mathcal{A}_u$.*

**Remark 3.2.** *Theorem 3.1 requires the unweighted algorithm $\mathcal{A}_u$ that we plug in to satisfy $\sigma_u = O(T_u)$. This assumption automatically holds when one defines the dynamic matching problem in the most natural way: If a dynamic matching algorithm is required to maintain an explicit list of the edges in the maintained matching, then it will need to spend $O(1)$ per change to the matching, and thus will have $\sigma_u = O(T_u)$.*

*An algorithm has $\sigma_u > T_u$ only if the output matching is stored* implicity. *As far as we know, the only dynamic matching algorithm in the literature with $\sigma_u > T_u$ is the maximal matching algorithm of Bernstein, Forster, and Henzinger [9]. Their algorithm maintains several different matchings along with a pointer to a matching that is guaranteed to be correct at the current time. An update may change the pointer in their algorithm from one solution to another. As a result, a single update might require only $O(1)$ update time, and yet cause the algorithm to point to a matching with a completely different set of edges. Our Result 1 can be easily modified to convert unweighted algorithms of this form as well.*

## 3.1 Previous Work

*Graph unfolding.* We start by describing the reduction from weighted to unweighted bipartite matchings through a transformation called *graph unfolding* due to Kao *et al.* [25]. Although our focus in this section is bipartite graphs, we note that graph unfolding can be applied to non-bipartite graphs as well, which we will consider in Section 4.

**Definition 3.3.** *Let $G = (V, E, w)$ be a graph with non-negative integral edge weights. The* unfolded graph $\phi(G)$ *is an* unweighted *graph defined as follows. For each node $u \in V(G)$, there are $\beta_u$ copies of $u$ in $\phi(G)$, denoted by $u^1, \ldots, u^{\beta_u}$, where $\beta_u = \max_{e \ni u} w(e)$. Corresponding to each edge $e = uv$ in $G$, there are $w(e)$ edges $\left\{u^i v^{w(e)-i+1}\right\}_{i \in [w(e)]}$ in $\phi(G)$.*

Observe that unfolding a subgraph $K \subseteq G$ produces a subgraph $\phi(K) \subseteq \phi(G)$; in particular, unfolding a (weighted) matching $M$ of $G$ produces an (unweighted) matching $\phi(M)$ of $\phi(G)$. The key property of graph unfolding in bipartite graphs is the following:

**Theorem 3.4** ([25]). *If $G$ is a weighted bipartite graph, then*

$$\textsc{mwm}(G) = \textsc{mcm}(\phi(G)).$$

The assumption that $G$ is bipartite in Theorem 3.4 is necessary; if the graph is not bipartite, $\textsc{mcm}(\phi(G))$ could be up to 1.5 times larger than $\textsc{mwm}(G)$, exhibited by a triangle with all edges of weight 2. Nonetheless, we will later see that one can sensibly apply graph unfolding to nonbipartite graphs.

We will often want to "reverse" the operation of unfolding to obtain a subgraph back in $G$, an operation we call *graph refolding*.

**Definition 3.5.** *Let $G$ be a weighted graph and let $H \subseteq \phi(G)$. The* refolded graph $\mathcal{R}(H)$ *of $H$ has vertex set $V$ and edges $E(\mathcal{R}(H)) = \left\{uv \in G : u^i v^{w(uv)-i+1} \in H \text{ for some } i \in [w(uv)]\right\}$.*

**Observation 3.6.** *Let $G$ be a weighted bipartite graph. If $M$ is an $\alpha$-approximate maximum cardinality matching of $\phi(G)$, then $\textsc{mwm}(\mathcal{R}(M)) \geq \alpha \cdot \textsc{mwm}(G)$.*

Proof. Since $\mathcal{R}(M)$ is bipartite, by Theorem 3.4, $\textsc{mwm}(\mathcal{R}(M)) = \textsc{mcm}(\phi(\mathcal{R}(M)))$. Observing that that $M \subseteq \phi(\mathcal{R}(M))$, we further have $\textsc{mcm}(\phi(\mathcal{R}(M))) \geq |M|$. Finally, since $M$ is $\alpha$-approximate in $\phi(G)$, it follows that $|M| \geq \alpha \cdot \textsc{mcm}(\phi(G)) = \alpha \cdot \textsc{mwm}(G)$, where we have used Theorem 3.4 again to justify the last equality. ∎

*The reduction of Gupta and Peng.* We now state a reduction of Gupta and Peng [23] that allows us to effectively assume the maximum weight of the input graph is a large constant. In their paper, Gupta and Peng do not emphasize that their reduction can be used as a black box; indeed, they only apply their reduction to a specific weighted matching algorithm. But without changing their argument, their techniques apply to any weighted matching algorithm, giving the following theorem. See the appendix for more details.

**Theorem 3.7** ([23]). *Let $G$ be a weighted graph and let $\varepsilon \in (0, 1/2)$. If there is an algorithm $\mathcal{A}$ that maintains an $\alpha$-approximate maximum weight matching in a graph whose weights belong to $[W]$ with update time $T(n, m, \alpha, W)$, then there is an algorithm $\mathcal{A}'$ that maintains a $(1-\varepsilon)\alpha$-approximate maximum weight matching in $G$ in update time $O_\varepsilon(T(n, m, \alpha, \gamma_\varepsilon) \cdot \log(R))$. Moreover, $\mathcal{A}$ has the same secondary features as $\mathcal{A}'$.*

### 3.2 A New Reduction to Unweighted Matching via Graph Unfolding

We now state the core of our reduction. Let $\mathcal{A}_u$ be a dynamic algorithm for maintaining an $\alpha$-approximate maximum cardinality matching in update time $T_u(n, m, \alpha)$, and let $\mathcal{A}_b$ be the dynamic algorithm for maintaining a maximum weight matching in bounded-degree weighted graphs implied by Theorem 2.1. We now give our algorithm $\mathcal{A}'_w$ that, using $\mathcal{A}_u$ and $\mathcal{A}_b$ as black boxes, maintains a $(1-\varepsilon)\alpha$-approximate maximum weight matching in a dynamic weighted graph $G$ whose weights are all in the set $[W]$. Our reduction will incur a large dependence on $W$ which we reduce later using Theorem 3.7.

**ALGORITHM 1:** BipartiteReduction

**Result:** Maintain a $(1-\varepsilon)\alpha$-approximate maximum weight matching in a bipartite graph with weights $[W]$.
**Input:** An insertion or deletion of edge $uv$.
1 Insert (delete) edges $u^i v^{w(uv)-i}$ into (from) $\phi(G)$ for $i \in [w(uv)]$.
2 Use $\mathcal{A}_u$ to maintain a matching $M_u$ of $\phi(G)$.
3 Update $\mathcal{R}(M_u)$ by tracking updates to $M_u$.
4 Use $\mathcal{A}_b$ to maintain a weighted matching $M$ of $\mathcal{R}(M_u)$.

Lemma 3.8. *The matching $M$ maintained by $\mathcal{A}'_w$ is a $(1-\varepsilon)\alpha$-approximate maximum weight matching.*

Proof. By the guarantees of $\mathcal{A}_u$, the matching $M_u$ of $\phi(G)$ is $\alpha$-approximate. Observation 3.6 states that $\text{MWM}(\mathcal{R}(M_u)) \geq \alpha \cdot \text{MWM}(G)$. Finally, $\mathcal{A}_b$ maintains a weighted matching $M$ such that

$$w(M) \geq (1-\varepsilon) \cdot \text{MWM}(\mathcal{R}(M_u)) \geq (1-\varepsilon)\alpha \cdot \text{MWM}(G). \qquad \blacksquare$$

Lemma 3.9. *The update time of Algorithm 1 is*

$$O(W^4 \cdot T_u(nW, mW, \alpha)/\varepsilon^2).$$

Proof. First observe that for every edge $uv$ that is updated in $G$, at most $W$ edges are updated in $\phi(G)$. By the update time of $\mathcal{A}_u$, it takes $W \cdot T_u(nW, mW, \alpha)$ total time to maintain $M_u$ after these $W$ updates to $\phi(G)$. Clearly the number of changes to $M_u$ cannot be more than the total update time, and thus there are also at most $W \cdot T_u(nW, mW, \alpha)$ changes to $M_u$.

Each update to $M_u$ in turn triggers at most one update to $\mathcal{R}(M_u)$. Since we use $\mathcal{A}_b$ to process updates to $\mathcal{R}(M_u)$ and the maximum degree of $\mathcal{R}(M_u)$ is $W$, the update time of $\mathcal{A}_b$ per udpate to $\mathcal{R}(M_u)$ is $O(W^3/\varepsilon^2)$. Thus, by Theorem 2.1, the update time per update to $G$ is $O(W^4 \cdot T_u(nW, mW, \alpha)/\varepsilon^2)$. $\blacksquare$

To reduce the dependence of $\mathcal{A}'_w$ on $W$, we apply the bucketing scheme of Gupta and Peng (Theorem 3.7), as illustrated in the following proof of Theorem 3.1.

Proof of Theorem 3.1. Suppose $T_u$ is the runtime of $\mathcal{A}_u$. Then, the previous two lemmas, Lemma 3.8 and Lemma 3.9, state that $\mathcal{A}'_w$ maintains a $(1-\varepsilon)\alpha$-approximate maximum weight matching in a weighted graph $G$ in $O(W^4 \cdot T_u(nW, mW, \alpha)/\varepsilon^2)$ update time. Applying Theorem 3.7 to $\mathcal{A}'_w$ with $T(n, m, \alpha, W) = O(W^4 \cdot T_u(nW, mW, \alpha)/\varepsilon^2)$, we obtain an algorithm $\mathcal{A}_w$ that maintains a $(1-2\varepsilon)\alpha$-approximate maximum weight matching in update time $O_\varepsilon(T_u(\gamma_\varepsilon n, \gamma_\varepsilon m, \alpha) \cdot \log(R))$.

All the subroutines we use—folding, refolding, $\mathcal{A}_b$, and the black box of Theorem 3.7— are worst-case and deterministic. Thus, the *secondary features* of $\mathcal{A}_u$ carry over to $\mathcal{A}_w$. $\blacksquare$

## 4 NON-BIPARTITE GRAPHS AND REFOLDING-APPROXIMATE SUBGRAPHS

As noted in the previous section, unfolding a non-bipartite graph $G$ poses additional issues. Since a matching in $\phi(G)$ is akin to a fractional matching in $G$, the unfolded graph $\phi(G)$ may have a matching up to 1.5 times larger than the maximum weight matching in $G$. At the same time, a maximum cardinality matching in $\phi(G)$ may refold into a graph that does not contain a maximum weight matching of $G$. In other words, both Theorem 3.4 and Observation 3.6 are false when $G$ is not bipartite.

### 4.1 A New Reduction for Weighted Matchings in General Graphs

In this section, we show that if a subgraph $H$ of $\phi(G)$ has sufficient structure, then $\mathcal{R}(H)$ may still contain a good matching. This motivates the following definition.

**Definition 4.1.** *Let $G$ be a weighted graph. A subgraph $H$ of $\phi(G)$ is* $\alpha$-refolding-approximate *if*

$$\text{MWM}(\mathcal{R}(H)) \geq \alpha \cdot \text{MWM}(G).$$

In Section 5, we will show that certain natural subgraphs $H$ have the property that, even in non-bipartite graphs, they incur only a $(1-\varepsilon)$-factor-loss in approximation after refolding, rather than the generic $2/3$-factor. For example, we will show that if $H$ is a *kernel* of $\phi(G)$ (defined in Section 5), then $H$ contains a $(1/2-\varepsilon)$-approximate maximum cardinality matching in $\phi(G)$ and is also $(1/2-\varepsilon)$-refolding-approximate. The existence of such graphs is itself not *a priori* obvious and is one of the key contributions of this paper.

We leave these structural theorems for the next subsection. Here, we formalize why maintaining a matching in a weighted graph can be reduced to maintaining a refolding-approximate subgraph of an unweighted graph. We maintain such subgraphs by running existing unweighted algorithms as *black-boxes* on the unfolded graph $\phi(G)$. Note that a single update to a graph $G$ with weights in $[W]$ can cause up to $W$ updates to $\phi(G)$; each of these updates might then cause the unweighted algorithm on $\phi(G)$ to make multiple updates to $H$, the $\alpha$-refolding-approximate subgraph. To bound this cascade of updates, we introduce the following definitions.

**Definition 4.2.** *Let $\mathcal{A}$ be an algorithm that maintains a subgraph $H$ of an unweighted graph $G$. We say that $\mathcal{A}$ has* update ratio $r$ *if any update to $G$ results in at most $r$ changes to $H$. We say that is has* amortized update ratio $r$ *if for any large enough sequence $S$ of edge changes (insertions or deletions) to $G$, the algorithm makes at most $r|S|$ edge changes to $H$.*

**Definition 4.3.** *Let $G$ be any weighted graph. Let $\mathcal{A}$ be an algorithm that maintains a subgraph $H$ of $\phi(G)$. We say that $\mathcal{A}$ has* unfolded update ratio *$r$ if any update to $G$ results in at most $r$ changes to $H$. We define* unfolded amortized update ratio *analogously.*

We now state our main theorem, which converts any algorithm that maintains an $\alpha$-refolding-approximate subgraph of $\phi(G)$ into one that maintains $\alpha(1-\varepsilon)$-approximate matching of $G$.

**Theorem 4.4.** *Let $G$ be a graph on $n$ vertices and $m$ edges with weights in $[W]$, and let $\varepsilon > 0$. If there is an algorithm $\mathcal{B}_u$ that maintains an $\alpha$-refolding-approximate subgraph $H$ of $\phi(G)$ over updates to $G$ with update time $T_u(n,m,\alpha,W)$ (per update to $G$), recourse $\sigma_u(n,m,\alpha,W) = O(T_u(n,m,\alpha,W))$, and unfolded update ratio $C_u(n,m,\alpha,W)$, then there is an algorithm $\mathcal{B}_w$ that maintains a $(1-\varepsilon)\alpha$-approximate maximum weight matching in $G$ with update time*

$$O_\varepsilon((T_u + \Delta_H C_u)(\gamma_\varepsilon n, \gamma_\varepsilon m, \alpha, \gamma_\varepsilon) \cdot \log(R)).$$

*Further, the* secondary features *of $\mathcal{B}_u$ extend over to $\mathcal{B}_w$ as well. (For $\mathcal{B}_w$ to be worst-case, both update time and unfolded update ratio of $\mathcal{B}_u$ must be worst-case.)*

Towards proving Theorem 4.4, we describe an algorithm $\mathcal{B}'_w$ that, given as input a dynamic graph $G$ with weights in $[W]$, uses $\mathcal{B}_u$ and $\mathcal{A}_b$ as a subroutine and maintains a $(1-\varepsilon)\alpha$-approximate maximum weight matching at all times. The update time has a polynomial dependence on $W$, which we later convert into a large constant using the reduction of Gupta and Peng (Theorem 3.7).

**ALGORITHM 2:** NonBipartiteReduction

**Result:** Maintain a $(1-\varepsilon)\alpha$-approximate maximum weight matching in a (possibly non-bipartite) graph with weights $[W]$.

**Input:** An insertion or deletion of edge $uv$.

1 Insert (delete) edges $u^i v^{w(uv)-i+1}$ into (from) $\phi(G)$ for $i \in [w(uv)]$.
2 Use $\mathcal{B}_u$ to maintain an $\alpha$-refolding-approximate subgraph $H$ of $\phi(G)$.
3 Update $\mathcal{R}(H)$ by tracking updates to $H$.
4 Use $\mathcal{A}_b$ to maintain a weighted matching $M$ of $\mathcal{R}(H)$.

Lemma 4.5. *The matching $M$ maintained by $\mathcal{B}'_w$ is a $(1-\varepsilon)\alpha$-approximate maximum weight matching of $G$.*

Proof. By Definition 4.1, the subgraph $\mathcal{R}(H)$ maintained with the help of $\mathcal{B}_u$ contains an $\alpha$-approximate maximum weight matching. Thus, running $\mathcal{A}_b$ on $\mathcal{R}(H)$ finds a matching $M$ such that $w(M) \geq (1-\varepsilon) \cdot \text{MWM}(\mathcal{R}(H)) \geq (1-\varepsilon)\alpha \cdot \text{MWM}(G)$. ∎

Lemma 4.6. *The update time of Algorithm 2 is $O(T_u(n,m,\alpha,W) + W^3\Delta_H \cdot C_u(n,m,\alpha,W)/\varepsilon^2)$.*

Proof. The proof is similar to the proof of Lemma 3.9, except now we use the update ratio to bound the number of changes to $\mathcal{R}(H)$.

After an update $uv$ in $G$, it takes $O(T_u(n,m,\alpha,W))$ time to maintain $H$. Since $C_u(n,m,\alpha,W)$ is the unfolded update ratio, the number of changes triggered in $H$ due to an update in $G$ is at most $C_u(n,m,\alpha,W)$. Note that each update in $H$ can cause at most one update in $\mathcal{R}(H)$. Additionally, the maximum degree of $\mathcal{R}(H)$ is at most $W \cdot \Delta_H$. So the total time taken by Algorithm 2 in response to a single update to $G$ is $O(T_u(n,m,\alpha,W) + W^3\Delta_H \cdot C_u(n,m,\alpha,W)/\varepsilon^2)$. ∎

Proof of Theorem 4.4. The two previous lemmas, Lemma 4.5 and Lemma 4.6, state that $\mathcal{B}'_w$ maintains a $(1-\varepsilon)\alpha$-approximate maximum weight matching in

$$O(T_u(n,m,\alpha,W) + W^3\Delta_H \cdot C_u(n,m,\alpha,W)/\varepsilon^2)$$

update time. Applying Theorem 3.7 to $\mathcal{B}'_w$, we obtain an algorithm $\mathcal{B}_w$ that maintains a $(1-2\varepsilon)\alpha$-approximate maximum weight matching in update time $O_\varepsilon((T_u + \Delta_H \cdot C_u)(n,m,\alpha,\gamma_\varepsilon) \cdot \log(R))$.

All the subroutines we use—folding, refolding, $\mathcal{A}_b$, and the black box of Theorem 3.7—are worst-case and deterministic. Thus, the secondary features of $\mathcal{A}_u$ carry over to $\mathcal{A}_w$. ∎

# 5 KERNELS AND EDCS ARE REFOLDING APPROXIMATE SUBGRAPHS

In this section, we prove that two existing matching sparsifiers for unweighted graphs—kernels and edge-degree-constrained subgraphs (EDCS)—are refolding approximate.

## 5.1 Existing Work in Unweighted Graphs

We first review existing work on the two sparsifiers and their properties in unweighted graphs.

*Kernels.* The first matching sparsifier we consider is the *kernel*, introduced in [14].

**Definition 5.1.** *A subgraph $H$ of an unweighted graph $G$ is a $(d,\varepsilon)$-*kernel *if the following two properties hold:*

*(1) for every vertex $v \in G$ we have $\deg(v) \leq d$, and*
*(2) for every edge $uv \in G \setminus H$ we have $\max\{\deg(u), \deg(v)\} \geq (1-\varepsilon)d$.*

Arar, Chechik, Cohen, Stein and Wajc [3] proved that every $(d,\varepsilon)$-kernel contains a $\left(\frac{1-\varepsilon}{2(1+1/d)}\right)$-approximate maximum cardinality matching. In bipartite graphs, following their proof exactly and using that bipartite graphs are $\Delta$-edge-colorable—where $\Delta$ is the maximum degree of the graph—gives the following simpler (independent of $d$) approximation ratio.

**Theorem 5.2** ([3])**.** *Let $G$ be an unweighted bipartite graph, let $d \in \mathbb{N}$, and let $\varepsilon > 0$. If $H$ is a $(d,\varepsilon)$-kernel of $G$, then $\text{MCM}(H) \geq \frac{1-\varepsilon}{2} \cdot \text{MCM}(G)$.*

Recently, Wajc [33] showed that kernels can be maintained in polylog$(n)$ update time against adaptive adversaries. More precisely, he proved the following theorem.

**Theorem 5.3** ([33])**.** *Let $G$ be an unweighted graph and let $\varepsilon \in (0, 1/2)$. For $d = \Theta(\log n \cdot \text{poly}(1/\varepsilon))$, there is a randomized (Las Vegas) algorithm that maintains a $(d,\varepsilon)$-kernel of $G$ in worst-case update time $O((\log^4 n + d \log n) \cdot \text{poly}(1/\varepsilon))$ with high probability against an adaptive adversary.*

*Edge-degree-constrained subgraphs (EDCS).* Another matching sparsifier used in dynamic graph algorithms is the *edge-degree-constrained subgraph* (EDCS), introduced in [10].

**Definition 5.4.** *A subgraph $H$ of an unweighted graph $G$ is a $(d, \varepsilon)$-EDCS if the following two properties hold:*

*(1) for every edge $uv \in E(H)$ we have $\deg(u) + \deg(v) \le d$, and*

*(2) for every edge $uv \in E(G) \setminus E(H)$ we have $\deg(u) + \deg(v) \ge (1-\varepsilon)d$.*

The EDCS strikes a different trade-off from the kernel: it obtains a $(2/3 - \varepsilon)$-approximate maximum cardinality matching and can be maintained in $O(m^{1/4}/\varepsilon^{5/2})$ update time. The approximation ratio of the EDCS in bipartite graphs was established in [10]. Later, it was shown by the same authors [11] that the EDCS obtains the same $(2/3 - \varepsilon)$ approximation ratio in general graphs. A simpler proof with improved parameters was presented in [5].

**Theorem 5.5** ([5, 10])**.** *Let $G$ be an unweighted bipartite graph, let $\varepsilon < 1/2$, and let $d \ge 16/\varepsilon$. If $H$ is a $(d, \varepsilon)$-EDCS of $G$, then $\textsc{mcm}(H) \ge (2/3 - 4\varepsilon)\textsc{mcm}(G)$.*

The following theorem, proved in [11], says that, with the appropriate choice of $d$, an EDCS can be maintained in the fully dynamic setting in update time sublinear in the number of vertices.

**Theorem 5.6** ([11])**.** *Let $G$ be an unweighted graph. For $d \ge 36/\varepsilon$, there is a deterministic algorithm that maintains a $(d, \varepsilon)$-EDCS of $G$ in $O(\sqrt{m}/(\varepsilon^2 d))$ amortized update time with $O(1/\varepsilon)$ amortized update ratio. (Note that this Theorem can be applied with $d$ a function of $m$ such as $d = m^{1/4}$; this was done in the algorithm of [11] and we apply their algorithm as a black box.)*

## 5.2 Kernels Are Refolding-Approximate

We now show that kernels are nearly $1/2$-refolding approximate—see Theorem 5.9 for the precise statement. Our proof relies an the existence of a partition of vertices of $G$ that (1) preserves the maximum weight matching and (2) roughly halves the degrees of all vertices in the unfolded graph $\phi(G)$. The existence of such a partition was established for unweighted graphs in [5]; here we generalize their proof.

Lemma 5.7. *Let $G$ be a weighted graph with maximum weight $W$, let $\delta \in (0, 1/2)$, and let $d \ge 36\delta^{-2}\log(W/\delta)$. For any subgraph $H$ of $\phi(G)$ with maximum degree at most $d$, there exists a bipartite subgraph $\tilde{G}$ of $G$ such that, setting $\tilde{H} := H \cap \phi(\tilde{G})$,*

*(1) $\textsc{mcm}(\phi(\tilde{G})) = \textsc{mwm}(G)$, and*

*(2) $\left|\frac{\deg_{\tilde{H}}(v) - \deg_H(v)}{2}\right| \le \delta d$ for all vertices $v \in V(H)$.*

Proof. The proof is based on the probabilistic method. Fix a maximum weight matching $M^*$ in $G$. We produce a random bipartite subgraph $\tilde{G} = (L \cup R, \tilde{E})$ of $G$ as follows. For each edge matched by $M^*$, we randomly and choose one endpoint to be in $L$ and then put the other endpoint in $R$, independent of all other choices. For every vertex unmatched by $M^*$, we place it in $L$ with probability $1/2$ and in $R$ with probability $1/2$, again independent of all other choices. The edges $\tilde{E}$ of $\tilde{G}$ are precisely those edges with one endpoint in $L$ and one endpoint in $R$.

The bipartite subgraph $\tilde{G}$ induces a corresponding bipartite subgraph $\phi(\tilde{G})$ in the unfolded space. Since $M^* \subseteq \tilde{G}$ by construction and since $\tilde{G}$ is bipartite, $\textsc{mwm}(G) = w(M^*) = \textsc{mwm}(\tilde{G}) = \textsc{mcm}(\phi(\tilde{G}))$, as desired. (The last inequality follows from Theorem 3.4, since $\tilde{G}$ is bipartite.)

Now let $H$ be any subgraph of $\phi(G)$ whose maximum degree is at most $d$. It suffices to show that with nonzero probability, $\tilde{H} := H \cap \phi(\tilde{G})$ satisfies the second condition. For each vertex $v \in \phi(G)$, let $\mathcal{E}_v$ be the "bad" event that $\left|\frac{\deg_{\tilde{H}}(v) - \deg_H(v)}{2}\right| > \delta d$, and let $X_v := \deg_{\tilde{H}}(v)$. Notice that we can write $X_v$ as the sum of independent indicators; each edge $uw \in M^* \cap H[v \cup N_H(v)]$ contributes 1 to $X_v$ and every neighbor $u$ not matched to a vertex in $v \cup N_H(v)$ contributes 1 if it is not assigned to the same side of the bipartition as $v$. Thus, $X_v$ is the sum of independent Bernoullis, and further, $\mathbb{E}[X_v] = \frac{\deg_H(v)+1}{2}$ if $v$ is unmatched by $M^*$ and otherwise $\mathbb{E}[X_v] = \frac{\deg_H(v)}{2}$. Thus the bad event $\mathcal{E}_v$ simply occurs when $X_v$ is not sufficiently concentrated around its mean.

As $X_v$ is the sum of independent Bernoullis, we may bound its deviation with a standard Chernoff bound, and thus,

$$\begin{aligned}
\Pr(\mathcal{E}_v) &= \Pr\left(\frac{\left|\deg_{\tilde{H}}(v) - \deg_H(v)\right|}{2} \ge \delta d\right) \\
&\le \Pr(|X_v - \mathbb{E}[X_v]| \ge \delta d/2) \\
&\le 2\exp(-\delta^2 d/12) \\
&\le 2\exp(-3\log(W/\delta)) \\
&\le 2\exp(-3\log(dW)) \\
&= 2(dW)^{-3}.
\end{aligned}$$

We now want to apply the Lovász Local Lemma, and so we must argue that $\mathcal{E}_v$ is independent of all but relatively few other events. Let $u^i, w^j \in H$. If $u$ and $w$ (vertices in $G$) are not within two hops of each other in $G$, then $\mathcal{E}_{u^i}$ and $\mathcal{E}_{w^j}$ are independent by construction of $\phi(\tilde{G})$. Thus, each $\mathcal{E}_v$ is independent of all but at most $(dW)^2$ other bad events, and since $2(dW)^{-3} \cdot ((dW)^2 + 1) \le 1/e$, the Lovász Local Lemma states that $\bigcap_{v \in \phi(G)} \overline{\mathcal{E}_v}$ occurs with nonzero probability. ∎

**Observation 5.8.** *Let $G$ be a weighted bipartite graph. If $H$ is a subgraph of $\phi(G)$, then $H \subseteq \phi(\mathcal{R}(H))$. It follows that $\textsc{mcm}(H) \le \textsc{mcm}(\phi(\mathcal{R}(H))) = \textsc{mwm}(\mathcal{R}(H))$, where Theorem 3.4 justifies the last equality.*

**Theorem 5.9.** *Let $G$ be a (possibly non-bipartite) graph with edge weights in $[W]$ and let $d \ge 4 \cdot 36\varepsilon^{-2}\log(2W/\varepsilon)$. If $H$ is a $(d, \varepsilon)$-kernel of $\phi(G)$, then $H$ is $(1/2 - 3\varepsilon/2)$-refolding-approximate.*

Proof. Let $\tilde{G}$ be the bipartite subgraph of $G$ from Lemma 5.7, and let $\tilde{H} = H \cap \phi(\tilde{G})$. Note that $\tilde{H}$ is bipartite since $\phi(\tilde{G})$ is bipartite. Observe that $\textsc{mwm}(\mathcal{R}(\tilde{H})) \le \textsc{mwm}(\mathcal{R}(H))$ because $\mathcal{R}(\tilde{H}) \subseteq \mathcal{R}(H)$. We now show that $\tilde{H}$ has a large matching by showing that it is a kernel of $\phi(\tilde{G})$.

Suppose $v \in V(H)$. By taking $\delta = \varepsilon/2$ and applying Lemma 5.7, we have

$$\deg_{\tilde{H}}(v) \leq \deg_H(v)/2 + \delta d \leq (1+\varepsilon)d/2.$$

Now suppose $uv \in \phi(\tilde{G}) \setminus \tilde{H}$. Since $uv \notin \tilde{H}$ but $u$ and $v$ are on opposite sides of the bipartition, the edge $uv \in G \setminus H$. Because $H$ is a $(d,\varepsilon)$-kernel, one of its endpoints has degree at least $(1-\varepsilon)d$ in $H$; we may assume w.l.o.g that $v$ is that endpoint. Again applying Lemma 5.7 we have that in $\tilde{H}$:

$$\deg_{\tilde{H}}(v) \geq \deg_H(v)/2 - \delta d \geq (1-2\varepsilon)d/2 \geq (1-3\varepsilon)(1+\varepsilon)d/2.$$

Thus, $\tilde{H}$ is a $((1+\varepsilon)d/2, 3\varepsilon)$-kernel of $\phi(\tilde{G})$.

Putting everything together,

$$\begin{aligned}
\textsc{mwm}(\mathcal{R}(H)) &\geq \textsc{mwm}(\mathcal{R}(\tilde{H})) && \text{(since } \mathcal{R}(\tilde{H}) \subseteq \mathcal{R}(H)\text{)} \\
&\geq \textsc{mcm}(\tilde{H}) && \text{(by Observation 5.8)} \\
&\geq \frac{1-3\varepsilon}{2} \cdot \textsc{mcm}(\phi(\tilde{G})) && \text{(by Theorem 5.2)} \\
&= \frac{1-3\varepsilon}{2} \cdot \textsc{mwm}(G). && \blacksquare
\end{aligned}$$

## 5.3 EDCS Are Refolding-Approximate

**Theorem 5.10.** *Let $G$ be a (possibly non-bipartite) graph with weights in $[W]$ and let $d \geq 16 \cdot 36\varepsilon^{-2}\log(4W/\varepsilon)$. If $H$ is a $(d,\varepsilon)$-EDCS of $\phi(G)$, then $H$ is $(2/3 - 12\varepsilon)$-refolding-approximate.*

Proof. Let $\tilde{G}$ be the bipartite subgraph of $G$ from Lemma 5.7, and let $\tilde{H} = H \cap \phi(\tilde{G})$. Observe that $\textsc{mwm}(\mathcal{R}(\tilde{H})) \leq \textsc{mwm}(\mathcal{R}(H))$ because $\mathcal{R}(\tilde{H}) \subseteq \mathcal{R}(H)$. We now show that $\tilde{H}$ has a large matching by showing that it is an EDCS of $\phi(\tilde{G})$.

Suppose $uv \in \tilde{H}$. By Lemma 10, taking $\delta = \varepsilon/4$, we have

$$\deg_{\tilde{H}}(uv) \leq \deg_H(uv)/2 + 2\delta d \leq (1+\varepsilon)d/2.$$

Now suppose $uv \in \phi(\tilde{G}) \setminus \tilde{H}$. Since $uv \in \phi(G) \setminus H$ and $H$ is a $(d,\varepsilon)$-EDCS, we have

$$\deg_{\tilde{H}}(uv) \geq \deg_H(uv)/2 - 2\delta d \geq (1-2\varepsilon)d/2.$$

Finally, since $(1-2\varepsilon) \geq (1-3\varepsilon)(1+\varepsilon)$, it follows that $\tilde{H}$ is a $((1+\varepsilon)d/2, 3\varepsilon)$-EDCS of $\tilde{G}$.

Putting everything together,

$$\begin{aligned}
\textsc{mwm}(\mathcal{R}(H)) &\geq \textsc{mwm}(\mathcal{R}(\tilde{H})) && \text{(since } \mathcal{R}(\tilde{H}) \subseteq \mathcal{R}(H)\text{)} \\
&\geq \textsc{mcm}(\tilde{H}) && \text{(by Observation 5.8)} \\
&\geq (2/3 - 12\varepsilon)\textsc{mcm}(\phi(\tilde{G})) && \text{(by Theorem 5.5)} \\
&= (2/3 - 12\varepsilon)\textsc{mwm}(G). && \blacksquare
\end{aligned}$$

## 5.4 Results on Dynamic Weighted Matching in General Graphs

Since kernels and EDCS are refolding-approximate, we can can apply Theorem 4.4 to obtain algorithms for weighted non-bipartite graphs. More precisely, we have the following two theorems.

**Theorem 5.11.** *Let $G$ be a (possibly non-bipartite) weighted graph and let $\varepsilon > 0$. There is randomized (Las Vegas) algorithm that maintains a $(1/2 - \varepsilon)$-approximate maximum weight matching with high probability and against an adaptive adversary in worst-case update time $O_\varepsilon(\log^5(n)\log(R))$.*

Proof. Choosing $d = \Theta(\log(n) \cdot \text{poly}(1/\varepsilon))$, we can compute a $(d,\varepsilon)$-kernel $H$ of $\phi(G)$ using the algorithm of Theorem 5.3 in $T_u = O(\log^4(W \cdot n) \cdot \text{poly}(1/\varepsilon))$ worst-case update time. By Theorem 5.9, $H$ is $(1/2-\varepsilon)$-refolding-approximate. Since each update in $G$ leads to at most $W$ updates in $\phi(G)$, and since the number of updates to $H$ is bounded by $T_u$, we have $C_u = W \cdot T_u$. Applying Theorem 4.4, there is an algorithm to maintain a $(1/2-\varepsilon)$-approximate maximum weight matching of $G$ in $O_\varepsilon(\log^5(n)\log(R))$ worst-case update time that works against an adaptive adversary with high probability. ∎

**Theorem 5.12.** *Let $G$ be a dynamic weighted (not necessarily bipartite) graph and let $\varepsilon > 0$. There is deterministic algorithm that maintains a $(2/3 - \varepsilon)$-approximate maximum weight matching with amortized update time $O_\varepsilon(m^{1/4}\log(R))$.*

Proof. Choosing $d = 16 \cdot 36\varepsilon^{-2}\log(4W/\varepsilon) \cdot m^{1/4}$, we can compute an EDCS $H$ of $\phi(G)$ with update time $O((m \cdot W)^{1/4})$ and amortized update ratio $O(1/\varepsilon)$ by Theorem 5.6. However, these update times and update ratios are with respect to updates in $\phi(G)$. As discussed before, each update in $G$ triggers at most $W$ updates in $\phi(G)$. So the update time and the amortized refolding update ratio for $H$ are $O((m \cdot W)^{1/4} \cdot W)$ and $O(W/\varepsilon)$, respectively. Applying Theorem 4.4 with $T_u = O((m \cdot W)^{1/4} \cdot W)$ and $C_u = O(W/\varepsilon)$, and from Theorem 5.10, we have an algorithm that maintains a $(2/3 - 2\varepsilon)$-refolding-approximate subgraph with update time $O_\varepsilon(m^{1/4}\log(R))$. ∎

# 6 FURTHER APPLICATIONS

The framework introduced in this paper is applicable beyond the dynamic setting. Indeed, the tools we use are quite general and easily translate to a wide variety of models. In this section, we demonstrate how to use our framework to obtain new results for weighted matching in the semi-streaming model and in the one-way two-player communication complexity model.

## 6.1 Previous Work: Revisiting Gupta and Peng's Reduction

One of the basic tools of our framework was the bucketing scheme of Gupta and Peng [23] (see Theorem 3.7), which allowed us to effectively reduce edge weights to a large constant. Although they described this scheme in the context of a particular algorithm for dynamic graphs, it is in fact extremely general and can be applied to almost any model. In this subsection, we state the consequences of this reduction to different models of interest; see Appendix C for the proofs of these theorems.

**Theorem 6.1.** *If there is a $p$-pass semi-streaming algorithm $\mathcal{A}_w$ to compute an $\alpha$-approximate maximum weight matching in a graph whose edge weights are in $[W]$ and that uses space $S(n,m,W,\alpha)$, then there is an $p$-pass semi-streaming algorithm $\mathcal{A}'_w$ to compute a $(1-\varepsilon)\alpha$-approximate maximum weight matching with weights in $\mathbb{R}^+$ using space $O_\varepsilon(S(n,m,\gamma_\varepsilon,\alpha)\log(n))$. Further, if $\mathcal{A}_w$ works for*

*vertex-order edge arrivals or random-order edge arrivals, then so does* $\mathcal{A}'_w$.

**Theorem 6.2.** *If there exists a one-way communication complexity protocol for the maximum weight matching problem in a graph with edge weights in* $[W]$ *using* $C(n, m, W, \alpha)$ *bits of communication, then there exists a protocol to compute a* $(1-\varepsilon)\alpha$*-approximate maximum weight matching with weights in* $\mathbb{R}^+$ *using* $O_\varepsilon(C(n, m, \gamma_\varepsilon, \alpha)\log(n))$ *bits of communication.*

## 6.2 Semi-Streaming in Bipartite Graphs

*Model definition.* In the semi-streaming model, the goal is to solve a problem over a stream of the input graph's edges while only using $\widetilde{O}(n)$ space. The main focus is typically on *one-pass algorithms*, which are only permitted to read the stream of edges once. For many problems, including computing maximum weight matchings, strong barriers are known for one-pass algorithms [22, 26] and so $p$-pass algorithms for $p > 1$ have also received much attention [2, 18, 19, 28, 29].

The semi-streaming model is doubly worst-case in the sense that algorithms should work both for any input graph and any permutation of the graph's edges. In many instances, requiring that the algorithm works against an adversarially-ordered stream is unnecessarily restrictive. One standard relaxation is to assume that the edges arrive in random order. For bipartite graphs, another relaxation is to assume a vertex arrival order, where each vertex on a fixed side of the bipartition arrives with all its incident edges. In other words, the edges are streamed to the algorithm from the adjacency list of one side of the bipartition.

*Our results.* As in the dynamic graph setting, we can use our framework to show that any semi-streaming algorithm for approximate unweighted matching can be converted to one for approximate weighted matching with exactly the same number of passes and essentially the same space and approximation guarantees. The high-level approach is identical to the dynamic setting: We first use graph unfolding to compute a maximum weight matching in a bipartite graph whose weights are in $[W]$, and then we apply Theorem 6.1 to reduce the dependence on $W$. In particular, we have the following theorem.

**Theorem 6.3.** *Let* $G$ *be a weighted bipartite graph and let* $\varepsilon \in (0, 1/2)$. *If there exists a* $p$*-pass semi-streaming algorithm* $\mathcal{A}_u$ *to compute an* $\alpha$*-approximate maximum cardinality matching using* $S_u(n, m, \alpha)$ *space, then there exists a* $p$*-pass semi-streaming algorithm* $\mathcal{A}_w$ *to compute an* $(1-\varepsilon)\alpha$*-approximate maximum weight matching using* $O_\varepsilon(S_u(\gamma_\varepsilon n, \gamma_\varepsilon m, \alpha)\log(n))$ *space. Further, if* $\mathcal{A}_u$ *works for vertex-order arrivals, then so does* $\mathcal{A}_w$.

**Remark 6.4.** *We note that although the bucketing scheme of Gupta and Peng can also be used for random edge arrivals (Theorem 6.1), our reduction in Theorem 6.3 cannot. The reason is that even if edges arrive in a random order in* $G$*, the order will not be random in the unfolded graph* $\phi(G)$*. See the conclusion section for a more detailed discussion.*

Proof (Theorem 6.3). Let $\mathcal{A}_u$ be the unweighted matching algorithm that uses space $S_u(n, m, \alpha)$. We describe how to compute an $\alpha$-approximate maximum weight matching in a graph whose edge weights are in $[W]$ using $\mathcal{A}_u$. On the arrival of edge $uv$, we stream the corresponding edges $\{u^i v^{w(uv)-i+1}\}_{i\in[w(uv)]}$ of $\phi(G)$ to $\mathcal{A}_u$. At the end of the stream, $\mathcal{A}_u$ reports an $\alpha$-approximate maximum weight matching $M_\phi$ in $\phi(G)$ using $S_u(nW, mW, \alpha)$ space. By Observation 3.6, the graph $\mathcal{R}(M_\phi)$—which we can easily recover from $M_\phi$—contains an $\alpha$-approximate maximum weight matching of $G$. Since we can compute the maximum weight matching of $G$ offline, this completes the reduction.

Notice that to preserve vertex-order, we cannot just stream the unfolded edges as is suggested above. However, by storing all the edges incident to a single vertex in $G$ in memory for the duration of that vertex's arrival, we can produce the corresponding vertex-order stream in $\phi(G)$.

Thus, using $\mathcal{A}_u$, we have obtained an algorithm to compute a maximum weight matching in a graph with edge weights in $[W]$. To finish the proof, we apply Theorem 6.1. ∎

*Applications of our results.* Theorem 6.3 has several consequences for computing maximum weight matchings in the streaming setting:

(1) It is a major open question as to whether there exists a one-pass semi-streaming algorithm to compute a better-than-1/2-approximation to the maximum cardinality matching problem. Our theorem shows that any such algorithm would also break the 1/2-approximation barrier for weighted *bipartite* graphs.
(2) For vertex arrivals, one can compute significantly better than a 1/2-approximation. A seminal result of Karp, Vazirani, and Vazirani [27] showed how to compute a $(1-1/e)$-approximation in vertex-ordered streams with randomization (indeed, their algorithm was designed for the more restrictive online setting). More recently, Goel, Khanna, and Kapralov [22] gave a deterministic single-pass algorithm for vertex-ordered streams with the same approximation guarantee. Kapralov [26] then showed these algorithms are tight: no one-pass algorithm can obtain a better-than-$(1-1/e)$-approximation using $\tilde{O}(n)$ space. Utilizing our reduction, we effectively settle the problem of *weighted* matchings in vertex-ordered streams: our algorithm computes a a $(1-1/e-\varepsilon)$-approximation in $\tilde{O}(n)$ space. To the best of our knowlege, no previous algorithm obtained a better-than-1/2-approximation in the vertex-arrival setting with edge weights.
(3) There are several semi-streaming algorithms for unweighted graphs that go beyond the 1/2-approximation barrier by using 2 or 3 passes instead of a single pass [19, 28, 29]. Our Theorem 6.3 yields algorithms for weighted *bipartite* graphs with essentially the same bounds. As far as we know, these are the first such results for weighted graphs.

## 6.3 One-Way Communication Complexity in Non-Bipartite Graphs

*Model definition.* In the one-way two-player communication complexity model, Alice and Bob each have some portion of the input, and the goal is to compute some function of the entire input. Alice can talk to Bob, but Bob cannot talk back to Alice; all communication flows in one direction. Understanding problems in the one-way two-player model is often seen as a first step to understanding them

in more difficult models such as the streaming model, and thus this communication complexity model has received a lot of recent attention.

For the maximum weight matching problem in the one-way two-party model, Alice receives a weighted graph $G_A = (V, E_A)$, Bob receives a weighted graph $G_B = (V, E_B)$, and their goal is for Bob to compute an approximate maximum weight matching of the graph $G = (V, E_A \cup E_B)$.

*Previous work.* The one-way communication complexity of maximum matchings was first studied by [22], who gave a one-way protocol for bipartite graphs that achieved a $2/3$-approximation using $O(n \log n)$ bits of communication. Further, they showed that any one-way protocol computing a $(2/3 + \delta)$-approximation for $\delta > 0$ requires $n^{1+\Omega(1/\log\log n)}$ bits of communication, showing that there is a sharp threshold at approximation factor $2/3$. In [5], the upper bound of [22] was extended to non-bipartite graphs, though the approximation factor obatined is $2/3 - \varepsilon$ instead of strictly $2/3$.

Within the proof of their upper bound, [5] establishes the following structural result, which we will need here (see Lemma 12 in that paper).

Lemma 6.5 ([5]). *Let $G$ be an unweighted graph and let $G_A$ and $G_B$ be a partition of $G$ into two subgraphs. Further, let $M$ be a matching of $G_A \cup G_B$, let $\varepsilon > 0$, and let $d \geq 64\varepsilon^{-2}\log(1/\varepsilon)$. If $H$ is a $(d, \varepsilon/2)$-EDCS of $G_A$, then $H \cup G_B$ contains a $(d, \varepsilon)$-EDCS of $M \cup G_B$.*

*Our results.* We show that the one-way communication complexity of the weighted matching problem is essentially the same as the one-way communication complexity of unweighted matching, provided that the ratio of the maximum and minimum edge weight is bounded by a polynomial in $n$. More precisely, we have the following theorem.

**Theorem 6.6.** *Let $G$ be a weighted graph and let $\varepsilon \in (0, 1/2)$. There is a one-way communication protocol to compute a $(2/3-\varepsilon)$-approximate maximum weight matching using $O_\varepsilon(n\log^2(n))$ bits of communication.*

Proof. We describe a simple protocol to compute an approximate maximum weight matching for graphs whose edge weights lie in $[W]$. Let $d = 16 \cdot 36\varepsilon^{-2}\log(4W/\varepsilon)$.

The protocol is as follows. Alice unfolds her input graph, computes a $(d, \varepsilon/2)$-EDCS $H_A$ and sends $H_A$ to Bob. Bob unfolds his input graph and adds $H_A$ to $\phi(G_B)$ to obtain a subgraph $K$ of $\phi(G)$. Bob then computes $\mathcal{R}(K)$ (he knows the original weights from, e.g., the labels of the vertices in $\phi(G)$) and computes a maximum weight matching in $\mathcal{R}(K)$.

Since Alice sends Bob $O(nWd)$ edges of a graph on at most $nW$ vertices, the protocol uses $O(nW\log(nW)\varepsilon^{-2}\log(W/\varepsilon))$ bits of communication. We now prove correctness of the protocol. Let $M^*$ be a maximum weight matching of $G$. By Lemma 6.5, the graph $K$ contains a $(d, \varepsilon)$-EDCS $H$ of $\phi(M^*) \cup \phi(G_B) = \phi(M^* \cup G_B)$. By Theorem 5.10, the refolded graph $\mathcal{R}(H)$ contains a $(2/3-12\varepsilon)$-approximate maximum weight matching of the graph $M^* \cup G_B$. Thus, since $K \supseteq H$, we have

$$\begin{aligned}\textsc{mwm}(\mathcal{R}(K)) &\geq \textsc{mwm}(\mathcal{R}(H))\\ &\geq (2/3 - 12\varepsilon)\textsc{mwm}(M^* \cup G_B)\\ &= (2/3 - 12\varepsilon)\textsc{mwm}(G).\end{aligned}$$

To complete the proof, we re-parameterize $\varepsilon$ and apply Theorem 6.2. ∎

## 7 CONCLUSION AND OPEN PROBLEMS

In this paper, we developed a new framework for weighted matching that allows us to use existing unweighted algorithms without modification. In bipartite graphs, this framework effectively settles the weighted/unweighted gap for approximate matching, because it allows us to convert *any* algorithm for unweighted matching into one for weighted matching, while only occurring an additional $(1 - \varepsilon)$ loss in the approximation ratio. In non-bipartite graphs, our framework does not lead to a universal transformation. Still, we show that combining the framework with our new structural properties of certain subgraphs leads to several new algorithms that essentially match the best-known unweighted algorithms. Our framework can be applied to many models, though we focused on dynamic algorithms where the weighted/unweighted gap was largest.

There are several natural open problems that arise from our results.

(1) **Reducing the dependence on $\varepsilon$.** Because we rely on the bucketing scheme of Gupta and Peng [23], our framework introduces a much larger dependence on $\varepsilon$ than the corresponding unweighted algorithms: the update time is multiplied by $(1/\varepsilon)^{O(1/\varepsilon)}$. Is it possible to reduce this factor to $\text{poly}(1/\varepsilon)$?
(2) **Non-bipartite graphs.** Is there a universal reduction that converts *any* dynamic algorithm for unweighted matching in *non-bipartite* graphs into one for weighted matching in non-bipartite graphs? That is, can our Result 1 be extended to non-bipartite graphs? It is not hard to check that our framework already does so if we allow an additional $2/3$ loss in the approximation ratio. Can this loss be made only $1 - \varepsilon$?
(3) **Other models.** As discussed in Section 6, our framework can be applied to many models, not just dynamic algorithms. A general open problem is thus what other models can benefit from our framework.

    We highlight one model in particular: computing a matching in one pass of semi-streaming when edges arrive in a *random* order. There is extensive literature on this problem in unweighted graphs [4, 8, 20, 28, 29], but the state-of-the-art for weighted graphs lags far behind [21]. We would like to apply our framework to close this weighted/unweighted gap, but even in bipartite graphs, there is a subtle issue: our framework runs existing unweighted algorithms in the unfolded graph $\phi(G)$, and even if edges arrive in a random order in $G$, they do not arrive in a random order in $\phi(G)$. We suspect this issue can be resolved because the correlation between edges in $\phi(G)$ seems relatively small.

    We think it is also likely that the current state-of-the-art algorithm for unweighted graphs by Bernstein [8] can be extended to weighted graphs *even in non-bipartite graphs*. The reason is that Bernstein's algorithm maintains an EDCS, which we showed in this paper is refolding-approximate. There are

two issues to resolve: the first is the general issue above about the randomness of edge arrivals in the unfolded graph, and the second is that Bernstein's algorithm only maintains a relaxed version of an EDCS; one would have to show this relaxed version is still refolding-approximate.

## ACKNOWLEDGEMENTS

We would like to thank Sayan Bhattacharya and David Wajc for very helpful discussions.

## A HANDLING SUPERPOLYNOMIAL WEIGHTS

As long as the weights of the input graph are polynomial in $n$, the additional $\log R$ factor in our update time is $O(\log n)$. But if $R$ is very large, then we can additionally use the following reduction of Stubbs and Williams [32].

**Theorem A.1** ([32]). *Let $\mathcal{A}$ be a dynamic algorithm that maintains an $\alpha$-approximate maximum weight matching with update time $T(n, m, \alpha, W)$ over a graph whose edge weights come from $[W]$. For every constant $\varepsilon > 0$, we can convert $\mathcal{A}$ into an algorithm that maintains an $(1+\varepsilon)\alpha$-approximate maximum weight matching with update time $T(n, m, \alpha, n^2\varepsilon^{-2})\log^2 n + \log n \cdot \log\log R + \log\log W$. Further, the new algorithm has the same* secondary guarantees *as the algorithm $\mathcal{A}$.*

Using this reduction with our framework gives, for example, a way to convert an algorithm for unweighted bipartite graphs with update time $T_u$ into one for weighted bipartite graphs with update time $O(T_u \log^3 n + \log n \cdot \log\log R + \log\log W)$. Analagous statements can be made for our other reduction.

## B THE BOUNDED-DEGREE DYNAMIC MATCHING ALGORITHM $\mathcal{A}_b$

In this section we describe an algorithm that meets the description of Theorem 2.1. We need the following linear-time $(1-\varepsilon)$-approximation algorithm of [17].

Lemma B.1 ([17]). *Let $\varepsilon \in (0, 1)$. A $(1-\varepsilon)$-approximate maximum weight matching in a weighted graph can be computed (statically) in $O(m/\varepsilon)$ time.*

Proof of Theorem 2.1. We give an algorithm with an amortized update time of $O(\Delta \cdot W^2/\varepsilon^2)$. The algorithm works in phases. At the beginning of a phase, the algorithm computes a $(1-\varepsilon/2)$-maximum weight matching $M$ in $O(m/\varepsilon)$ time using the algorithm of Lemma B.1. Each phase continues for $\varepsilon|M|/2W$ updates. This phase length ensures that throughout each phase, the total weight of $M$ can reduce by at most $\varepsilon w(M)/2$, and thus $M$ remains a $(1-\varepsilon)$-approximate maximum weight matching at all times. The amortized update time is $O(mW/\varepsilon^2|M|)$.

We finish the proof by showing that $\frac{m}{|M|} = O(\Delta W)$. Let $M^*$ be the maximum weight matching of $G$ and consider $|M^*|$. Notice that every edge of $G$ is adjacent to some edge in $M^*$, or else $M^*$ would not be maximum. Thus,

$$\begin{aligned} m &\le 2\Delta|M^*| \\ &\le 2\Delta w(M^*) \\ &\le 2\Delta(1+\varepsilon)w(M) \\ &\le 2\Delta(1+\varepsilon)W|M|. \end{aligned}$$

Substituting this upper bound on $m/|M|$ gives an amortized bound of $O(\Delta W^2/\varepsilon^2)$. The update time can be deamortized in a standard fashion by staggering the work. ∎

## C THE REDUCTION OF GUPTA AND PENG

Here we describe the bucketing scheme of Gupta and Peng [23]. Loosely speaking, Gupta and Peng show how to reduce the maximum weight matching problem in arbitrarily weighted graphs to the maximum weight matching problem in graphs where weights belong to $[W]$, where $W$ is an integer depending only on the accuracy parameter $\varepsilon$. Although Gupta and Peng only describe their scheme in the context of a specific dynamic matching result, their idea is in fact extremely versatile: it yields a black-box reduction not only for dynamic matching (our Theorem 3.7), but also for streaming (our Theorem 6.1) and communication complexity (our Theorem 6.2). For more details, see Section 4 of [23].

We now turn to the details of the bucketing scheme and its structural guarantees. Let $G = (V, E, w)$ be a weighted graph, let $\varepsilon \in (0, 1/2]$, and let $\beta = \lceil 1/\varepsilon \rceil$. For each $i \in [\beta]$, we define graphs $G^{(i)}$ are defined as follows. First partition the edges of the graph geometrically according to their weights: an edge $e$ with $w(e) \in [\varepsilon^{-b}, \varepsilon^{-(b+1)})$ lies in *bucket* $b$. The graph $G^{(i)}$ is then obtained by deleting all edges in each bucket $b$ such that $b \equiv i \pmod{\beta}$. Notice that this deletion procedure naturally defines *levels* in each $G^{(i)}$, where the $\ell$th level is comprised of the edges that lie in the buckets $\{d \cdot \beta + i + 1, \ldots, (d+1) \cdot \beta + i - 1\}$. We denote the edges at level $\ell$ of $E^{(i)}$ as $E^{(i)}_\ell$. More tersely, we have the following definitions.

**Definition C.1.** *For $i \in [\beta]$ and $\ell \ge 0$, let*

$$E^{(i)}_\ell := \{e \in E : (1/\varepsilon)^{i+\ell\cdot\beta+1} \le w(e) < (1/\varepsilon)^{i+\ell\cdot(\beta+1)}\}.$$

*Let $E^{(i)} := \bigcup_\ell E^{(i)}_\ell$ and let $G^{(i)} := (V, E^{(i)}, w)$.*

Notice that for each level $\ell$ of $G^{(i)}$, there is at least a $(1/\varepsilon)^{1/\varepsilon}$-factor gap between the largest weight in level $\ell$ and the smallest weight in level $\ell+1$. The analysis of Gupta and Peng provides a proof for Lemma C.2 below, which uses Algorithm 3.

**ALGORITHM 3:** GreedyCombine

```
Input: A list of matchings M_1, M_2, ..., M_k.
Output: A matching M.
for i ← 1 to k do
    for e ∈ M_i do
        if M ∪ e is a matching then
            M ← M ∪ e
        end
    end
end
return M
```

Lemma C.2 ([23]). *The following hold:*

(1) *There is some* $i \in [\beta]$ *such that* $\text{MWM}(G^{(i)}) \geq (1-\varepsilon)\text{MWM}(G)$.

(2) *Let* $i \in [\beta]$ *and, for each level* $\ell$*, let* $M_\ell^{(i)}$ *be an* $\alpha$*-approximate maximum weight matching of* $E_\ell^{(i)}$*. Also, let* $L$ *be the largest non-empty level of* $E^{(i)}$*. The matching*

$$M := \textsc{GreedyCombine}\left(M_L^{(i)}, M_{L-1}^{(i)}, \ldots, M_0^{(i)}\right)$$

*satisfies* $w(M) \geq (1-3\varepsilon)\alpha \cdot \text{MWM}(G^{(i)})$.

Proof. The following simple averaging argument establishes that some $G^{(i)}$ contains a good matching. Let $F^{(i)} = E \setminus E^{(i)}$ be the set of edges missing from $G^{(i)}$ and let $M^*$ be a maximum weight matching of $G$. Observe that $\{F^{(i)}\}_{i\in\beta}$ is a partition $E$, there is some $j \in [\beta]$ such that $w(M^* \cap F^{(j)}) \leq w(M^*)/\beta \leq \varepsilon w(M^*)$. Thus, $w(M^* \cap E^{(j)}) \geq (1-\varepsilon)w(M^*)$.

We now show that $\alpha$-approximate maximum weight matchings from each level can be greedily combined via Algorithm 3 to produce a $(1-3\varepsilon)\alpha$-approximate maximum weight matching of $G^{(i)}$. Let $H = \bigcup_\ell M_\ell^{(i)}$. Clearly $w(H) \geq \alpha \cdot \text{MWM}(G^{(i)})$ since, level-by-level, $H$ gets at least an $\alpha$ fraction of the weight as the maximum weight matching of $G^{(i)}$. Now we show that $(1+3\varepsilon)w(M) \geq w(H)$. We charge the weight of each edge $e \in H$ to some adjacent edge in $f \in M$; if $e \in M$, then we charge the weight of $e$ to itself. Let $\Phi(e)$ be the total amount charged to the edge $e$. Since each edge $e \in M$ of level $\ell$ is charged to once by itself and at most twice from every other level less than $\ell$, we have

$$\begin{aligned}
\Phi(e) &\leq w(e) + 2\sum_{j=0}^{\ell-1} (1/\varepsilon)^{i+j\beta} \\
&= w(e) + 2(1/\varepsilon)^i \cdot \frac{(1/\varepsilon)^{\ell\beta} - 1}{(1/\varepsilon)^\beta - 1} \\
&\leq w(e) + 2(1/\varepsilon)^i \cdot \frac{(1/\varepsilon)^{\ell\beta}}{3/4 \cdot (1/\varepsilon)^\beta} \\
&< w(e) + 3(1/\varepsilon)^{i+(\ell-1)\beta} \\
&\leq w(e) + 3\varepsilon \cdot w(e) \\
&= (1+3\varepsilon)w(e).
\end{aligned}$$

Hence, $w(H) = \Phi(M) \leq (1+3\varepsilon)w(M)$. ∎

Since the ratio between weights within a level is at most $(1/\varepsilon)^\beta$, we may assume by a standard rescaling and rounding argument that the edges weights are integers in $\{1, 2, \ldots, \lceil (1/\varepsilon)^{\beta+1} \rceil\}$. Thus, the previous lemma says that, provided that we can compute approximate maximum matchings whose edges belong to $[W]$ and provided that we can combine the matchings efficiently à la GreedyCombine, we can compute a $(1-4\varepsilon)$-approximate maximum weight matching. We now sketch the proof of the reduction in each model we apply it to.

**Theorem 3.7** ([23]). *Let* $G$ *be a weighted graph and let* $\varepsilon \in (0, 1/2)$*. If there is an algorithm* $\mathcal{A}$ *that maintains an* $\alpha$*-approximate maximum weight matching in a graph whose weights belong to* $[W]$ *with update time* $T(n, m, \alpha, W)$*, then there is an algorithm* $\mathcal{A}'$ *that maintains a* $(1-\varepsilon)\alpha$*-approximate maximum weight matching in* $G$ *in update time* $O_\varepsilon(T(n, m, \alpha, \gamma_\varepsilon) \cdot \log(R))$*. Moreover,* $\mathcal{A}$ *has the same secondary features as* $\mathcal{A}'$*.*

Observe that approximate maximum weight matchings can be maintained in each level using algorithm $\mathcal{A}$. The only challenge then is to maintain the greedily combined matchings for each level set. See [23] for the details and proof.

Proof of Theorem 6.1. In the streaming setting, we run the instance of the matching algorithm for each level of $G^{(i)}$ for all $i \in [\beta]$. Let $\gamma_\varepsilon = (1/\varepsilon)^{1/\varepsilon}$. As we observed in the discussion above, we can assume that at each level, the edges have weights that are integers in $\{1, 2, \cdots, \lceil \gamma_\varepsilon \rceil\}$, each instance of the algorithm takes space $S(n, m, \gamma_\varepsilon, \alpha)$. Additionally, we observe that the total number of levels in each $G^{(i)}$ are $\frac{\log_\beta R}{\beta}$ due to the property that there is a $(1/\varepsilon)^{1/\varepsilon}$-factor gap between the heaviest edge level $d$ and the lightest edge in level $d+1$. Finally, we run our algorithm over $\beta$ many $G^{(i)}$'s, and at the end of the stream all these matchings can be greedily combined. This gives us a space bound of $O_\varepsilon(S(n, m, \gamma_\varepsilon, \alpha) \log R)$.

To improve the $\log R$ factor to $\log n$, we drop light edges when a very heavy edge is encountered. More specifically, at all times we maintain $W_{\max}$, the weight of the heaviest edge. When an edge that has weight greater than $W_{\max}$ is encountered, we update $W_{\max}$ and all edges with weight less than $\delta = \frac{\varepsilon \cdot W_{\max}}{n^2}$ are dropped. Since $w(\text{MWM}(G)) \geq W_{\max}$, and the total weight of the dropped edges is at most $\varepsilon \cdot W_{\max}$, the graph still contains a $(1-\varepsilon)$ approximation to $\text{MWM}(G)$. Additionally, when we drop these light edges, we don't modify the partition of $E^{(i)}$ into different levels. Note that while it is no longer true that the edges of $G^{(i)}$ at level $d$ have scaled weights between $(1/\varepsilon)^{i+d\cdot\beta+1}$ and $(1/\varepsilon)^{i+(d+1)\cdot\beta}$, this property is not essential to ensure guarantees of Lemma C.2. Certain crucial properties of $G^{(i)}$ still hold, namely: $\left\{F^{(i)}\right\}_{i\in[\beta]}$ partitions the undeleted edges of $G$ and within $G^{(i)}$, there is a $(1/\varepsilon)^{1/\varepsilon}$-factor gap between the heaviest edge of level $d$, and the lightest edge of $d+1$. These are sufficient to ensure the guarantees of Lemma C.2. Moreover, due to the second property, we are able to ensure that the number of levels in $G^{(i)}$ are at most $\frac{\log_\beta n}{\beta}$ many. This is because the ratio between $W_{\max}$ and the lowest weight edge in the graph is at most $n^2/\varepsilon$. This gives us the desired space bound. ∎

Proof of Theorem 6.2. The proof for this case is identical to Theorem 6.1. Alice performs the reduction stated in Lemma C.2 on her part of the input $(V, E_A)$. She runs a version of the original protocol on each of the levels of $G^{(i)}$. Additionally, she can throw out edges that are lighter than $\frac{\varepsilon \cdot W_{\max}}{n^2}$ since the total weight of these edges is at most $\varepsilon \cdot W_{\max}$. She sends the messages corresponding to each version of the protocol to Bob. This message has size at most $O_\varepsilon(C(n, n, \gamma_\varepsilon, \alpha) \log n)$. Due to the correctness of the protocol, Bob is able to compute an $\alpha(1-\varepsilon)$-approximation to the maximum matching in each level of $G^{(i)}$ for all $i \in [\beta]$. Bob then runs Algorithm 3 to get a $(1-O(\varepsilon))$-$\text{MWM}(G)$. ∎

# D EFFICIENTLY MAINTAINING INTEGRAL MATCHINGS AND KERNELS

A recent result due to Wajc [33] gives a new dynamic matching sparsification algorithm—an algorithm for computing a sparse subgraph that preserves the matching size approximately and can be maintained efficiently. He uses the scheme to design an algorithm that maintains a kernel of $G$. We use his algorithm as a black box. Wajc's description of the algorithm uses slightly different organization/notation than we do, and so for convenience, we outline how it works in this section.

We first restate the formal result, which stems from Lemma 4.6 of [33] and the discussion of the running time in section 4.2.1 of the same paper.

**Theorem 5.3** ([33]). *Let $G$ be an unweighted graph and let $\varepsilon \in (0, 1/2)$. For $d = \Theta(\log n \cdot \text{poly}(1/\varepsilon))$, there is a randomized (Las Vegas) algorithm that maintains a $(d, \varepsilon)$-kernel of $G$ in worst-case update time $O((\log^4 n + d \log n) \cdot \text{poly}(1/\varepsilon))$ with high probability against an adaptive adversary.*

We first describe Wajc's algorithm and then summarize how it gives a proof of Theorem 5.3.

*Wajc's sparsification scheme.* The algorithm $\mathcal{S}$ creates a sparsifier as follows: it takes as input a fractional matching $\vec{x} = (x_1, \ldots, x_m)$, then it creates $O(\log n/\varepsilon^2)$ graphs $G^{(i)}$, where $G^{(i)}$ only consists of edges $e_j$ such that $x_j \in ((1+\varepsilon)^{-i}, (1+\varepsilon)^{-i+1}]$ (not to be confused with the graphs in Appendix C). The algorithm then proceeds to compute a proper edge coloring $\chi_i$ of each graph $G^{(i)}$. Finally, it samples a set of colors $S_i$ from $\chi_i$ and includes in $H$ all the edges of $G^{(i)}$ that are colored using the pallette $S_i$.

Proof sketch of Theorem 5.3. A key observation of [33] is that, if their sparsification algorithm is given as input a specific type of fractional matching, called a $(\delta, d)$-approximately maximal fractional matching $\vec{x}$ for a sufficiently large $d$, and if the edges in the support of $\vec{x}$ are sampled with the right probabilities depending on a parameter $\varepsilon$, and $d$, then the output $H$ is in fact a $(d(1+O(\varepsilon)), O(\delta)+O(\varepsilon))$-kernel with high probability. We define the notion of an approximately maximal fractional matching, first introduced by [3].

**Definition D.1.** *A fractional matching $\vec{x}$ is a $(\delta, d)$-approximately maximal if every edge $e \in E$ has fractional value $x_e > \frac{1}{d}$, or it has one end point $v$ with $\sum_{e' \ni v} x_{e'} \geq 1 - \delta$, with all edges $e'$ incident on this $v$ having value $x_{e'} \leq \frac{1}{d}$.*

**Theorem D.2.** *Let $\delta \geq 0$, $\varepsilon > 0$, and $d \geq \frac{(1+\varepsilon)^2 \log n}{(1-\delta)\varepsilon^2}$. If $\vec{x}$ is a $(\delta, d)$-approximately maximal fractional matching, then the subgraph $H$ output by the sparsification algorithm of [33] when run on $\vec{x}$ with $\varepsilon$ and $d$ is a $(d(1+O(\varepsilon)), O(\delta)+O(\varepsilon))$-kernel with high probability. Further, since $|E(H)| = \tilde{O}(\mu(G))$ and the kernel can be sampled in $\tilde{O}(\mu(G))$ time, it can be maintained against an adaptive adversary in $\tilde{O}(1)$ time with high probability.*

We state briefly why the above theorem is correct. Note that the sparsification algorithm of [33] samples a fixed number of edges from $G$. Essentially, for every $e \in G$,

$$\frac{\min\{1, x_e \cdot d\}}{(1+\varepsilon)^2} \leq \Pr(e \in H) \leq \min\{1, x_e \cdot d\} \cdot (1+\varepsilon) \qquad (1)$$

Further, for $e \neq e'$, the random variables $\mathbb{1}_{\{e \in H\}}$ and $\mathbb{1}_{\{e' \in H\}}$ are negatively associated (intuitively, because the algorithm samples a fixed number of edges from $G$). The bound (1) immediately tells us that $\mathbb{E}\left[\deg_H(v)\right] \leq (1+\varepsilon)d$ for any vertex $v$. Since $\deg_H(v)$ is a sum of negatively associated 0-1 random variables, we may apply a Chernoff bound to obtain $\deg_H(v) \leq d(1+O(\varepsilon))$ with high probability. We are left with proving that $H$ satisfies the second property of kernels with high probability. For any edge $e \notin H$, we know that $\Pr(e \in H) < 1$. For such an edge $e$, we can deduce that $x_e < \frac{1}{d}$. Since $\vec{x}$ is a $(\delta, d)$-approximately maximal fractional matching, there is an endpoint $v$ of $e$ such that $\sum_{e' \ni v} x_{e'} \geq 1-\delta$. So, $\mathbb{E}\left[\deg_H(v)\right] \geq \frac{d(1-\delta)}{(1+\varepsilon)^2}$. A Chernoff bound shows that $\deg_H(v) \geq \frac{d(1-\delta)}{(1+O(\varepsilon))} \geq d(1-\delta)(1-O(\varepsilon))$ with high probability.

To understand why the kernel can be sampled in $\tilde{O}(\mu(G))$ time, note that the sparsification algorithm of [33] edge-colors the graph on the support of $\vec{x}$. It achieves this via a deterministic algorithm with a worst-case update time of $O(\log n)$ (see [13]). Then, it samples $\tilde{O}_\varepsilon(1)$ color classes. Each of these color classes form a matching, so $H$ can be sampled in $\tilde{O}_\varepsilon(\mu(G))$ time. The update time and the adaptive adversary guarantee follow from the stability property of matching. At a high level, we can wait for $\varepsilon \cdot \mu(G)$ updates before recomputing the kernel since after $\varepsilon\mu(G)$ updates, the matching in the support of kernel is still a $(1 - O(\varepsilon))$-matching (even against an adaptive adversary). This already gives an amortized update time of $\tilde{O}_\varepsilon(1)$, and the update-time can be deamortized by spreading the work across the $\varepsilon \cdot \mu(G)$ updates.

The $(1/2 - \varepsilon)$-fractional matching maintained by the deterministic algorithm of [16] in $\text{poly}(\log n, 1/\varepsilon)$ worst case update time was proved by [3] to be a $(O(\varepsilon), d)$-approximately maximal matching for some $d$ large enough to satisfy the conditions of Theorem D.2.

To summarize, after each update, the algorithm after each update, computes a fractional matching $\vec{x}$ using the algorithm of [16]. To then find the kernel, it computes a coloring of the support of $\vec{x}$ using the algorithm of [13]. Then it performs the sampling process described above, and the graph sampled is a kernel with high probability. Since the procedures of [13, 16] and the sampling procedure of [33] take time $\text{poly}(\log n, 1/\varepsilon)$, the overall runtime of the algorithm is $\text{poly}(\log n, 1/\varepsilon)$. ∎

## REFERENCES


[1] Amir Abboud and Virginia Vassilevska Williams. 2014. Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. 55st Annual Symp. on Foundations of Comput. Sci. (FOCS)*. IEEE Computer Society, 434–443.

[2] Kook Jin Ahn and Sudipto Guha. 2011. Linear programming in the semi-streaming model with application to the maximum matching problem. In *Proc. 38th International Colloquium on Automata, Languages, and Programming (ICALP) (Lecture Notes in Comput. Sci.)*, Vol. 6756. 526–538. https://doi.org/10.1007/978-3-642-22012-8_42

[3] Moab Arar, Shiri Chechik, Sarel Cohen, Clifford Stein, and David Wajc. 2018. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In *Proc. 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, Vol. 107. Art. No. 7, 16.

[4] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. 2019. Coresets meet EDCS: algorithms for matching and vertex

cover on massive graphs. In *Proc. 30th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA)*. 1616–1635.

[5] Sepehr Assadi and Aaron Bernstein. 2019. Towards a unified theory of sparsification for matching problems. In *Proc. 2nd Symposium on Simplicity in Algorithms (SOSA)*, Vol. 69. Art. No. 11, 20.

[6] Surender Baswana, Manoj Gupta, and Sandeep Sen. 2018. Fully dynamic maximal matching in $O(\log n)$ update time (corrected version). *SIAM J. Comput.* 47, 3 (2018), 617–650.

[7] Soheil Behnezhad, Jakub Łkacki, and Vahab Mirrokni. 2020. Fully dynamic matching: Beating 2-approximation in $\Delta^{\varepsilon}$ update time. In *Proc. 31th Annual ACM-SIAM Symp. Discrete Algorithms (SODA)*. 2492–2508.

[8] Aaron Bernstein. 2020. Improved bounds for matching in random-order streams. In *Proc. 47th International Colloquium on Automata, Languages, and Programming (ICALP)*, Vol. 168. 12:1–12:13. https://doi.org/10.4230/LIPIcs.ICALP.2020.12

[9] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. 2019. A Deamortization Approach for Dynamic Spanner and Dynamic Maximal Matching. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, Timothy M. Chan (Ed.). SIAM, 1899–1918. https://doi.org/10.1137/1.9781611975482.115

[10] Aaron Bernstein and Cliff Stein. 2015. Fully dynamic matching in bipartite graphs. In *Proc. 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, Vol. 9134. 167–179. https://doi.org/10.1007/978-3-662-47672-7_14

[11] Aaron Bernstein and Cliff Stein. 2016. Faster fully dynamic matchings with small approximation ratios. In *Proc. 27th Annual ACM-SIAM Sym. on Discrete Algorithms (SODA)*. ACM, New York, 692–711. https://doi.org/10.1137/1.9781611974331.ch50

[12] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. 2020. Deterministic dynamic matching in $O(1)$ update time. *Algorithmica* 82, 4 (2020), 1057–1080.

[13] Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. 2018. Dynamic algorithms for graph coloring. In *Proc. 29th Annual ACM-SIAM Symp. Discrete Algorithms (SODA)*. 1–20.

[14] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. 2018. Deterministic fully dynamic data structures for vertex cover and matching. *SIAM J. Comput.* 47, 3 (2018), 859–887. https://doi.org/10.1137/140998925

[15] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 2016. New deterministic approximation algorithms for fully dynamic matching. In *Proc. 48th Annual ACM-SIGACT Symp. on Theory of Comput., (STOC)*. 398–411. https://doi.org/10.1145/2897518.2897568

[16] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 2017. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In *Proc. 28th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*. 470–489. https://doi.org/10.1137/1.9781611974782.30

[17] Ran Duan and Seth Pettie. 2014. Linear-time approximation for maximum weight matching. *J. ACM* 61, 1 (2014), Art. 1, 23. https://doi.org/10.1145/2529989

[18] Sebastian Eggert, Lasse Kliemann, and Anand Srivastav. 2009. Bipartite graph matchings in the semi-streaming model (extended abstract). In *Proc. 17th Annual European Symp. Algorithms (ESA '09)*, Vol. 5757. 492–503. https://doi.org/10.1007/978-3-642-04128-0_44

[19] Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. 2016. Finding large matchings in semi-streaming. In *IEEE International Conference on Data Mining Workshops (ICDM '16)*. IEEE Computer Society, 608–614.

[20] Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. 2020. Approximate maximum matching in random streams. In *Proc. 31th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA)*. 1773–1785.

[21] Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. 2019. Weighted matchings via unweighted augmentations. In *Proc. ACM Symp. on Principles of Distributed Computing, (PODC)*. 491–500.

[22] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. 2012. On the communication and streaming complexity of maximum bipartite matching. In *Proc. 23rd Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*. 468–485.

[23] Manoj Gupta and Richard Peng. 2013. Fully dynamic $(1+\varepsilon)$-approximate matchings. In *Proc. 54th Annual Symp. Foundations Comput. Sci. (FOCS)*. 548–557. https://doi.org/10.1109/FOCS.2013.65

[24] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. 2015. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proc. 47th Annual ACM on Symp. on Theory of Comput., (STOC)*. 21–30.

[25] Ming-Yang Kao, Tak-Wah Lam, Wing-Kin Sung, and Hing-Fung Ting. 2001. A decomposition theorem for maximum weight bipartite matchings. *SIAM J. Comput.* 31, 1 (2001), 18–26. https://doi.org/10.1137/S0097539799361208

[26] Michael Kapralov. 2013. Better bounds for matchings in the streaming model. In *Proc. 24th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*. 1679–1697.

[27] Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. 1990. An optimal algorithm for on-line bipartite matching. In *Proc. 22nd Annual ACM Symp. on Theory of Comput. (STOC)*. 352–358.

[28] Christian Konrad. 2018. A simple augmentation method for matchings with applications to streaming algorithms. In *Proc. 43rd International Symp. on Mathematical Foundations of Comput. Sci., (MFCS)*, Vol. 117. 74:1–74:16. https://doi.org/10.4230/LIPIcs.MFCS.2018.74

[29] Christian Konrad, Frédéric Magniez, and Claire Mathieu. 2012. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization (APPROX/RANDOM)*, Vol. 7408. 231–242. https://doi.org/10.1007/978-3-642-32512-0_20

[30] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. 2016. Higher lower bounds from the 3SUM conjecture. In *Proc. of the 27th Annual ACM-SIAM Symp. on Discrete Algorithms, (SODA)*. 1272–1287.

[31] Shay Solomon. 2016. Fully dynamic maximal matching in constant update time. In *Proc. 57th Annual Symp. on Foundations of Comput. Sci., (FOCS)*. 325–334.

[32] Daniel Stubbs and Virginia Vassilevska Williams. 2017. Metatheorems for dynamic weighted matching. In *Proc. 8th Innovations in Theoretical Computer Science, (ITCS)*, Vol. 67. 58:1–58:14. https://doi.org/10.4230/LIPIcs.ITCS.2017.58

[33] David Wajc. 2020. Rounding dynamic matchings against an adaptive adversary. In *Proc. 52nd Annual ACM-SIGACT Symp. Theory Comput. (STOC)*. 194–207.