Dynamic Matching with Better-than-2 Approximation in Polylogarithmic Update Time*

SAYAN BHATTACHARYA†, University of Warwick, United Kingdom
PETER KISS‡, University of Warwick, United Kingdom
THATCHAPHOL SARANURAK§, University of Michigan, United States
DAVID WAJC¶, Technion — Israel Institute of Technology, Israel

We present dynamic algorithms with *polylogarithmic* update time for estimating the size of the maximum matching of a graph undergoing edge insertions and deletions with approximation ratio *strictly better than* 2. Specifically, we obtain a $1+\frac{1}{\sqrt{2}}+\epsilon \approx 1.707+\epsilon$ approximation in bipartite graphs and a $1.973+\epsilon$ approximation in general graphs. We thus answer in the affirmative the value version of the major open question repeatedly asked in the dynamic graph algorithms literature. Our randomized algorithms' approximation and worst-case update time bounds both hold w.h.p. against adaptive adversaries.

Our algorithms are based on simulating new two-pass streaming matching algorithms in the dynamic setting. Our key new idea is to invoke the recent sublinear-time matching algorithm of Behnezhad (FOCS'21) in a white-box manner to efficiently simulate the second pass of our streaming algorithms, while bypassing the well-known vertex-update barrier.

CCS Concepts: • **Theory of computation** → Streaming, sublinear and near linear time algorithms; Approximation algorithms analysis; **Dynamic graph algorithms**.

Additional Key Words and Phrases: dynamic algorithms, approximate matching

ACM Reference Format:
Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. 2018. Dynamic Matching with Better-than-2 Approximation in Polylogarithmic Update Time. *J. ACM* 37, 4, Article 111 (August 2018), 32 pages.
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The maximum matching problem is a cornerstone of combinatorial optimization and theoretical computer science more broadly. (We recommend [41] for a brief history of this problem.) The study of this problem and its extensions has contributed foundational advances and concepts

*A preliminary version of this paper appeared in the Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2023 [31]. This full version contains (in Sections 1.3.2 and 6) a discussion of the numerous developments in the area following our conference publication, as well as more detailed follow-up questions, including concerning applications of our adversarially-robust fully-dynamic almost-maximal matching (AMM) algorithms.

†Supported by Engineering and Physical Sciences Research Council, UK (EPSRC) Grant EP/S03353X/1.
‡Work was partially conducted while the author was visiting Max-Planck-Institut für Informatik.
§Supported by NSF grant CCF-2238138.
¶Supported in part by a Taub Family "Leaders in Science & Technology" Fellowship. Work done in part while the author was at Stanford University.

Authors' addresses: Sayan Bhattacharya, S.Bhattacharya@warwick.ac.uk, University of Warwick, United Kingdom; Peter Kiss, peter.kiss@warwick.ac.uk, University of Warwick, United Kingdom; Thatchaphol Saranurak, University of Michigan, United States, thsa@umich.edu; David Wajc, Technion — Israel Institute of Technology, Israel, david.wajc@gmail.com.

0004-5411/2018/8-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

to the theory of computing, from the introduction of the primal-dual method [67], impact on polyhedral combinatorics [44], and the advocacy for polynomial-time computability as the measure of efficiency (in static settings) [45].

The maximum matching problem has also been intensely studied in *dynamic* settings. Here, the graph undergoes edge *updates* (insertions and deletions), and we wish to approximate the maximum matching, while spending little computation time between updates, referred to as *update time*. Polynomial update time is trivial to achieve by running exact static algorithms (e.g., [45]) after each update. However, intuitively, such minor changes to the graph should allow for much faster algorithms, with possibly even exponentially smaller, *polylogarithmic* update times.

The first sublinear (i.e., $o(m) = o(n^2)$) update time dynamic matching algorithm was given 15 years ago by Sankowski [79], who used fast dynamic matrix inversion to maintain the maximum matching size in update time $O(n^{1.495})$, recently improved to $O(n^{1.407})$ [34]. Unfortunately, a number of fine-grained complexity results rule out fast, and even sublinear-in-n update time [1, 2, 40, 58, 66] for (exact) maximum matching size estimation. This motivates the wealth of work on computing *approximate* matchings dynamically.

The first polylogarithmic update time dynamic matching algorithm is due to an influential work of Onak and Rubinfeld [74], who gave a (large) constant approximation in polylog update time. This was later improved by Baswana et al. [11] to a 2-approximation in logarithmic update time, later improved to constant time by Solomon [80]. Numerous other algorithms achieving a 2- or $(2+\epsilon)$-approximation in polylog update time were subsequently developed, with expected amortized update time improved to worst-case w.h.p.,[1] and oblivious randomized algorithms improved to advsersarially-robust ones, and then to deterministic ones [4, 14, 21, 26, 28, 33, 35, 36, 62, 82].[2]

A complementary line of work studied better-than-two-approximate dynamic matching, providing a number of small polynomial (even sublinear in n) update times for approximation ratios below the natural bound of 2 achieved by inclusion-wise maximal matchings. This includes $(1+\epsilon)$-approximate algorithms with $O_\epsilon(\sqrt{m}) = O_\epsilon(n)$ update time [57, 77], $\left(\frac{3}{2}+\epsilon\right)$-approximate algorithms with $O_\epsilon(\sqrt[4]{m}) = O_\epsilon(\sqrt{n})$ update time [22, 23, 56, 62] and a number of tradeoffs between approximation in the range $(3/2, 2)$ and sublinear-in-n polynomial update times [16, 17, 28, 78, 82].[3]

This state of affairs leaves open a key question, repeatedly raised in the literature [16, 17, 23, 26, 35, 68, 82], first posed by Onak and Rubinfeld in their aforementioned groundbreaking work [74]:

> *How small can [approximation factors] be made with polylogarithmic update time? [...] Can the approximation constant be made smaller than 2 for maximum matching?*

1.1 Our Results

We resolve the question of polylogarithmic update time better-than-two-approximate dynamic matching algorithms in the affirmative, *for the value version of the problem*. That is, letting $\mu(G)$ denote the maximum matching size in G, we maintain an estimate ν that is $\alpha < 2$ approximate, i.e., it satisfies $\nu \leq \mu(G) \leq \alpha \cdot \nu$ at every point in time. Our main result is the following.

[1] An algorithm has *amortized* update time $f(n)$ if every sequence of t updates starting from an empty graph takes at most $t \cdot f(n)$ update time. If each operation takes at most $f(n)$ time, it has *worst-case* update time $f(n)$.

[2] An algorithm *works against an adaptive adversary* if its guarantees hold even when future updates depend on the algorithm's previous output. We also say that such an algorithm is *adversarially robust*, or *robust* for short. The importance of robustness for *static* applications has motivated a recent concentrated effort to design robust dynamic algorithms for myriad problems (see, e.g., discussions in [19, 28, 38, 49, 71, 82]).

[3] Throughout the paper, we use $O_\epsilon(\cdot)$ to suppress $\text{poly}(1/\epsilon)$ factors and $\tilde{O}(\cdot)$ to suppress $\text{poly}(\log n)$ factors.

Theorem 1.1. *For every $\epsilon \in (0,1)$, there exists a randomized $(1.973+\epsilon)$-approximate dynamic matching size estimation algorithm with $\text{poly}(\log n, 1/\epsilon)$ worst-case update time. Both the algorithm's approximation ratio and update time hold w.h.p against an adaptive adversary.*

For bipartite graphs, we obtain a stronger approximation guarantee of $1+\frac{1}{\sqrt{2}}+\epsilon \approx 1.707+\epsilon$.

Secondary results. Our approach is versatile, and yields the following generic reduction.

Theorem 1.2. *For any $\alpha > 1.5$, a dynamic α-approximate matching algorithm with update time t_u implies a dynamic $\left(\alpha - \Omega\left(\left(1-6\left(\frac{1}{\alpha}-\frac{1}{2}\right)\right)^2\right)\right)$-approximate matching* size estimator *with update time $\tilde{O}(t_u)$.*

Very recently, Behnezhad and Khanna [16] presented new dynamic matching algorithms trading off approximations $\alpha \in (1.5, 2]$ and small polynomial update times. Applying Theorem 1.2 to their algorithms, we obtain improved approximation for dynamic matching size estimation, within the same update time up to polylog factors.

To obtain our main results, we design several two-pass *semi-streaming* algorithms (see Section 1.3), including a deterministic $(1+1/\sqrt{2}+\epsilon)$-approximate algorithm on bipartite graphs. This matches the prior state-of-the-art [63, 65] up to an ϵ term, while removing the need for randomization.

1.2 Our Techniques

We take the following high-level approach to prove Theorem 1.1: (1) compute a maximal (and hence 2-approximate) matching M_1, and (2) augment M_1 if it is no better than 2-approximate, using the myriad short augmenting paths M_1 must have in this case. This approach is common in many computational models, including the two-pass semi-streaming model (see Section 1.3). Implementing this approach in a dynamic setting, however, faces several challenges. The first challenge if we want robust algorithms with low worst-case update times is that no robust (near-)maximal matching algorithms with worst-case $\tilde{O}_\epsilon(1)$ update time are known. Of possible independent interest, we resolve this first challenge in Section 5, by leveraging the robust fast matching sparsifiers of [82].

The more central challenge when trying to implement the above approach is that the search for augmenting paths requires us to find many (disjoint) edges between matched and unmatched nodes in M_1. In a (multi-pass) streaming setting, this can be done by computing a large (b-)matching in the bipartite graph induced by edges in $V(M_1) \times \overline{V(M_1)}$. In a dynamic setting, however, this requires us to deal with *vertex updates*, which are notoriously challenging in the context of dynamic matching, and all algorithms to date require reading all $\Omega(n)$ edges of each updated vertex [68].

To overcome the above key challenge, we first note that we do not need to handle vertex updates individually, but may instead process these in *batches* of $\Theta(\epsilon n)$ vertex updates, building on the periodic recomputation and sparsification techniques common in the literature (see Theorem 2.1). Our main observation is that these batches of vertex updates, which need to be handled if we wish to maintain the b-matchings from the second pass of our semi-streaming algorithms, can be implemented in $\tilde{O}_\epsilon(n)$ time using the sublinear-time algorithm of Behnezhad [12]. This leads to an amortized $\tilde{O}_\epsilon(n)/(\epsilon n) = \tilde{O}_\epsilon(1)$ additive overhead in the update time (easily deamortized), implying our main result. This approach is versatile, and similarly underlies our secondary results.

1.3 Further Related Work

Having discussed the rich literature on the dynamic matching problem above, we do not elaborate on it further here. We do, however, highlight some connections to the literature on matching in other computational models that is closely related to our work.

Streaming Matching. In the (semi-)streaming model, an n-node graph is revealed in a stream, edge by edge, and we wish to compute a large matching using only (optimal) $\tilde{O}(n)$ space. A line of work studying the problem of computing an approximately-maximum weighted matching [39, 47, 50, 53, 70, 76] culminated in a $(2+\epsilon)$-approximation [53, 76]. For unweighted graphs, lower bounds are known [54, 60, 61], but it remains a major open question whether one can break the barrier of 2-approximation achievable by a trivial maximal matching algorithm. Striving for better approximation (and insights to break this barrier), several works designed algorithms using *multiple passes* over the stream [3, 3, 6, 46, 46, 48, 51, 52, 59, 70]. For 2 passes, the state-of-the-art approximation ratios are 1.857 [51], and $1+\frac{1}{\sqrt{2}} \approx 1.707$ for bipartite graphs using the randomized algorithms of [63, 65], with the best prior deterministic bound being $\frac{12}{7} \approx 1.714$ [48].

Sublinear-Time Matching. Computation of large matchings in sublinear *time* has also been the subject of great interest. In regular bipartite graphs, a maximum matching can be computed in $\tilde{O}(n)$ time [55]. In general graphs with bounded degrees, it was known how to achieve a $(2+\epsilon)$-approximation in sublinear time [72, 73, 75, 83]. This was recently improved to an $\tilde{O}(n)$ time algorithm for *any* general graph [12]. As discussed in Section 1.2, we use this latter algorithm in a white-box manner to obtain our main result.

1.3.1 Concurrent work. Independently and concurrently, Behnezhad [13] (in a work in the same conference) obtained the same main qualitative result as ours: a better-than-two-approximate polylogarithmic-xwtime dynamic matching size estimation algorithm. The basic approach to achieve this qualitative result is the same in both papers: Simulate the second pass of a two-pass streaming algorithm using the sublinear-time algorithm of [12], together with batched computation. The quantitative differences in the papers' approximation ratios are due to the two-pass streaming algorithms used—our new maximal-b-matching-based algorithms here, and an algorithm inspired by [64] in [13]. We note that [13] also achieves $(3/2-\Omega(1))$-approximate size estimation algorithm in time $O(\sqrt{n})$ (the best update times for $(3/2+\epsilon)$-approximate explicit matching [22, 23, 56, 62]). This result also uses the high-level approach of batched computation using sublinear-time algorithms, building on a new characterization of tight examples for the 3/2-approximate matching sparsifiers (EDCS) of [22].

1.3.2 Subsequent work. The conference versions of this work and the concurrent work of [13] have sparked an interest in the question of dynamic matching size maintenance. [10] show that our $(1+\frac{1}{\sqrt{2}}+\epsilon)$-approximate two-pass streaming algorithm also works in general graphs, and show how to implement this in dynamic settings, using distributed algorithms. [30] show that $(1+\epsilon)$-approximate size estimation can be maintained in truly sublinear-in-n update time for any constant $\epsilon > 0$. This should be contrasted with a conditional lower bound of [69], who show that such an update time is unlikely if one wishes to *maintain* such an approximately maximum matching. Complementing this conditional lower bound, [7, 15] show conditional upper bounds. Specifically, they provide $(1+\epsilon)$-approximate matching algorithms with sublinear-in-n update time, provided some generalization of Rusza-Szemerédi graphs cannot be too dense. Proving or ruling out algorithms for maintenance of large matchings mirroring the recent results for dynamic size estimation is an exciting research direction. One direction to obtain such algorithmic results would be to obtain algorithms for dynamic *fractional* matchings, since the latter are known to be efficiently roundable dynamically, both in bipartite graphs [4, 28, 32, 82] and in general graphs [37, 43].

2 PRELIMINARIES

Our input is a graph G on n nodes V, with an initially empty edge set E, undergoing edge updates (insertions and deletions). Our objective is to approximate the maximum matching size $\mu(G)$ well, while spending little update time (computation between updates). In addition, we want our algorithms to work in the strictest settings: they should have small *worst-case* update time guarantees, and they should be (adversarially) *robust*, i.e., they should work against an adaptive adversary, and thus their guarantees hold for any update sequence.

Matching theory basics. A *matching* is a vertex-disjoint subset of edges. A *maximal matching* is an inclusionwise-maximal matching. A maximum matching is a matching of largest cardinality. In a weighted graph with edge weights $w_e \in \mathbb{R}$, a maximum weight matching is a matching M of largest total weight, $w(M) := \sum_{e \in M} w_e$. An *augmenting path* P with respect to a matching M is a simple path starting and ending with distinct nodes unmatched in M, with the edges alternatingly outside and inside M. Setting $M \leftarrow M \bigoplus P$, where $\bigoplus$ denotes the symmetric difference, referred to as *augmenting* M along P, increases the cardinality of M by one. A *b-matching* with capacities $\{b_v\}_{v \in V}$ is a collection of *multi-edges* F of E (that is, edges of E may appear multiple times in F) with no vertex v having more than b_v multi-edges in F. A *fractional matching* $x : E \to \mathbb{R}_{\geq 0}$ assigns non-negative values to edges so that each vertex v has *fractional degree* $\sum_{e \ni v} x_e$ at most one. In bipartite graphs, the existence of a fractional matching of size k implies the existence of an integral matching of cardinality $\lceil k \rceil$. In general graphs, this fractional relaxation has a maximum integrality gap of $3/2$, attained by a triangle graph with values $x_e = 1/2$ for each edge e.

Notation: Let $V(M)$ denote the set of all endpoints of edges in a matching M, and let $\overline{V(M)} := V \setminus V(M)$. For any disjoint vertex sets $A, B \subseteq V$, we let $G[A, B]$ denote the bipartite subgraph induced by the edges in G with one endpoint in A and another in B. Finally, for any subset of edges $E' \subseteq E$, we let $G[E']$ denote the subgraph of G induced by E'.

2.1 Previous building blocks

A ubiquitous paradigm in the approximate dynamic matching literature is *periodic recomputation*, introduced by Gupta and Peng [57]. This approach is particularly useful in conjunction with sparsification techniques. We will use the vertex sparsification technique introduced by Assadi et al. [8] in the context of stochastic optimization, and adapted to dynamic settings by Kiss [62]. Combined, these approaches yield the following "reduction" from dynamic matching algorithms with *immediate* queries to ones with slower query time.

Proposition 2.1. *Let $\epsilon \in (0, 1)$ and $\alpha \geq 1$. Suppose there exists an algorithm $\mathcal{A}$ on a dynamic n-node graph G with update time t_u, that, provided $\mu(G) \geq \epsilon \cdot n$, supports t_q-time α-approximate size estimate queries w.h.p. Then, there is another algorithm $\mathcal{A}'$ on G that always maintains an $(\alpha + O(\epsilon))$-approximate estimate ν' in $\tilde{O}_\epsilon(t_u + t_q/n)$ update time. Moreover if the update time of $\mathcal{A}$ is worst-case, so is that of $\mathcal{A}'$, and if $\mathcal{A}$ works against an adaptive adversary, then so does $\mathcal{A}'$.*

The above proposition, implicit in prior work, serves as a useful abstraction, and so we provide a proof of this proposition for completeness in Appendix A. As discussed in Section 1.2, this reduction is one of the crucial ingredients that allows us to bypass the vertex-update barrier.

Another key ingredient we use is the sublinear-time (approximate) maximal matching algorithm of Benhezhad [12], whose guarantees are captured by the following proposition. (See Lemma 4.6 for a proof of a detailed description of that algorithm, adapted for a variant of this proposition.)

Proposition 2.2. *Let $\epsilon \in (0, 1/2)$. Using $\tilde{O}_\epsilon(n)$ time and $\tilde{O}_\epsilon(n)$ adjacency matrix queries w.h.p. in an n-node graph G, one can compute a value ν which approximates $\tilde{\mu}$, the size of some maximal matching in G, within additive error ϵn. Namely, $\tilde{\mu} \geq \nu \geq \tilde{\mu} - \epsilon n$.*

A simple combination of propositions 2.1 and 2.2 (with $t_q = \tilde{O}_\epsilon(n)$) immediately yields (yet) another $\tilde{O}_\epsilon(1)$-time $(2+\epsilon)$-approximation algorithm. As we will show, these propositions are also useful ingredients for breaking the barrier of 2-approximation within the same update time.

2.2 New algorithmic primitive: Robust Almost-Maximal Matchings

To make our algorithms robust against adaptive adversaries we need an algorithm for maintaining *almost-maximal matchings* (AMM), which are defined as follows.

Definition 2.3 ([77]). *A matching M is an* ϵ-almost maximal matching (ϵ-AMM) *in graph G if M is maximal in some subgraph obtained by removing at most $\epsilon \cdot \mu(G)$ nodes of G.*

Observation 2.4. *If M is an ϵ-AMM in G, then $|M| \geq \frac{1}{2}(1-\epsilon) \cdot \mu(G) = \left(\frac{1}{2} - \frac{\epsilon}{2}\right) \cdot \mu(G)$.*

Peleg and Solomon [77] showed how to deterministically maintain an ϵ-AMM quickly in bounded-arboricity (i.e., globally sparse) graphs. In Section 5 we show how to robustly maintain such matchings quickly in arbitrary graphs, proving the following.

Lemma 2.5. *For any $\epsilon \in (0, 1/2)$, there exists a robust dynamic algorithm that w.h.p. maintains an ϵ-AMM in worst-case update time $\tilde{O}_\epsilon(1)$.*

A well-known fact is that a maximal matching that is close to 2-approximate must admit many length-three augmenting paths (see e.g., [64, Lemma 1]). Our interest in AMMs is in part motivated by the following slight generalization of this fact, also presented in Appendix A for completeness.

Proposition 2.6. *Let $\epsilon > 0$ and $c \in \mathbb{R}$ and let M be an ϵ-AMM in G such that $|M| \leq \left(\frac{1}{2} + c\right) \cdot \mu(G)$. Then M admits a collection of at least $\left(\frac{1}{2} - 3c - \frac{7\epsilon}{2}\right) \cdot \mu(G)$ node-disjoint 3-augmenting paths.*

3 ALGORITHMS ON BIPARTITE GRAPHS

In this section we illustrate our techniques for the special case of bipartite graphs, for which we obtain an improved approximation ratio of $1 + \frac{1}{\sqrt{2}} + \epsilon \approx 1.707 + \epsilon$.

3.1 Two-Pass Streaming Algorithm

Here we present our deterministic two-pass streaming algorithm. We first compute an approximately-maximal matching M_1 from the first pass.[4] Then, in the second pass, we compute a maximal b-matching M_2 in the graph between matched and unmatched vertices, with capacities k and $\lfloor k \cdot b \rfloor$, respectively, where we set the parameters $k \in \mathbb{N}$, $\delta \in \mathbb{R}$ and $b = 1/\delta \in \mathbb{R}$ later. Finally, we output an estimate $(1-\delta) \cdot |M_1| + (\delta/k) \cdot |M_2|$. Our pseudocode is given in Algorithm 1.

Algorithm 1 Bipartite Two-Pass Streaming Algorithm

Parameters: $\epsilon \in \mathbb{R}_{\geq 0}$, $k \in \mathbb{N}$, $\delta \in \mathbb{R}_{\geq 0}$ and $b = 1/\delta \in \mathbb{R}_{\geq 0}$.

1: $M_1 \leftarrow$ $(\epsilon/8)$-AMM in G computed from first pass

2: assign each vertex v capacity $b_v = \begin{cases} k & v \in V(M_1) \\ \lfloor k \cdot b \rfloor & v \notin V(M_1) \end{cases}$

3: $M_2 \leftarrow$ maximal b-matching in $G[V(M_1), \overline{V(M_1)}]$ computed from second pass

4: **Output** $(1-\delta) \cdot |M_1| + (\delta/k) \cdot |M_2|$.

First, we prove that the above algorithm's output estimate corresponds to a matching in G.

[4] We suggest to the reader to think of M_1 as a maximal matching (i.e., $\epsilon = 0$). We relax M_1 to be an $(\epsilon/8)$-AMM since this will be useful in our dynamic implementation that works against adaptive adversaries.

Observation 3.1. *We have that* $\mu(G[M_1 \cup M_2]) \geq (1-\delta)\cdot|M_1| + (\delta/k)\cdot|M_2|$.

Proof. Since G is bipartite, by the integrality of the bipartite fractional matching polytope, to prove that $G' := G[M_1 \cup M_2]$ contains a large matching witnessing the desired inequality, it suffices to prove that G' contains a *fractional* matching $\vec{x}$ of value $\sum_e x_e = (1-\delta)\cdot|M_1| + (\delta/k)\cdot|M_2|$. Indeed, such a fractional matching is obtained by assigning edge values

$$x_e = \begin{cases} 1-\delta & e \in M_1 \\ \delta/k & e \in M_2 \setminus M_1. \end{cases}$$

This is indeed a fractional matching, since each vertex v has bounded fractional degree, $\sum_{e\ni v} x_e \leq 1$: every vertex $v \in V(M_1)$ has one incident M_1 edge and at most k many incident M_2 edges, and so $\sum_{e\ni v} x_e \leq (1-\delta) + (\delta/k)\cdot k = 1$, while every vertex $v \notin V(M_1)$ has no incident M_1 edge and has at most $k\cdot b$ incident M_2 edges, and so $\sum_{e\ni v} x_e \leq k\cdot b\cdot(\delta/k) = 1$. □

By Lemma 3.1, Algorithm 1 outputs a valid estimate for the matching size, $\nu \leq \mu(G)$. It remains to prove that ν provides a good approximation of $\mu(G)$. For this, we require the following.

Lemma 3.2. *Let M be a maximal b-matching in a bipartite graph $G = (L \cup R, E)$, with positive integral capacities $b_v = \ell$ for all $v \in L$ and $b_v = r$ for all $v \in R$. Then*

$$|M| \geq \mu(G)\cdot\frac{\ell\cdot r}{\ell + r}.$$

Proof. Fix a maximum matching M^* in G. Next, we define the subset of matched nodes in M^* that are also saturated in M. That is, if $d_M(v)$ is v's degree in M, we let

$$L^*_{sat} := \{u \in L \cap V(M^*) \mid d_M(u) = \ell\}$$
$$R^*_{sat} := \{v \in R \cap V(M^*) \mid d_M(v) = r\}.$$

Let $\alpha := |L^*_{sat}|/|M^*|$ and $\beta := |R^*_{sat}|/|M^*|$ denote the fraction of M^* edges with a saturated L and R node, respectively. Since M is a maximal b-matching in G, each edge has at least one saturated endpoint, and so $\alpha + \beta \geq 1$. By double counting the edges of M, relying on $\alpha+\beta \geq 1$, and noting that $\alpha\cdot r + (1-\alpha)\cdot\ell$ attains its minimum of $2\frac{\ell\cdot r}{\ell+r}$ at $\alpha = \frac{r}{\ell+r}$, we obtain the claimed inequality.

$$\begin{aligned}
|M| &= \frac{1}{2}\left(\sum_{v\in L} d_M(v) + \sum_{v\in R} d_M(v)\right) \\
&\geq \frac{1}{2}\left(\sum_{v\in L^*_{sat}} d_M(v) + \sum_{v\in R^*_{sat}} d_M(v)\right) \\
&= \frac{1}{2}\cdot(|M^*|\cdot\alpha\cdot\ell + |M^*|\cdot\beta\cdot r) \\
&\geq \frac{1}{2}\cdot\mu(G)\cdot(\alpha\cdot\ell + (1-\alpha)\cdot r) \\
&\geq \mu(G)\cdot\frac{\ell\cdot r}{\ell+r}.
\end{aligned}$$

□

We are now ready to bound the approximation ratio of Algorithm 1.

Lemma 3.3. *For any $\epsilon \in (0,1)$, Algorithm 1 with $b = 1+\sqrt{2}$ and $k \geq \frac{8}{\epsilon b}$ run on bipartite graph G computes a $(1 + \frac{1}{\sqrt{2}} + \epsilon) \approx (1.707+\epsilon)$-approximation to $\mu(G)$.*

Proof. Fix a maximum matching M^* in G. Next, for $i \in \{0,1,2\}$, let M_i^* denote the edges of M^* with i endpoints matched in M_1. By definition, and since $|M_1| = \frac{1}{2} \cdot |V(M_1)|$, we have that

$$|M_1| = |M_2^*| + (1/2) \cdot |M_1^*|. \tag{1}$$

Furthermore, since M_1 is an ϵ'-AMM in G for $\epsilon' = \epsilon/8$, we have that $|M_0^*| \leq \epsilon' \cdot \mu(G)$, since at most $\epsilon' \cdot \mu(G)$ nodes of G must be removed from G to make M_1 maximal, and at least one endpoint of each M_0^* edge must be removed to achieve the same effect. But then, since M_0^*, M_1^*, M_2^* partition M^*, whose cardinality is $|M^*| = \mu(G)$, this implies that

$$|M_1^*| + |M_2^*| \geq (1 - \epsilon') \cdot \mu(G). \tag{2}$$

Now, by Lemma 3.2, since M_1^* is a matching in graph $G' := G[V(M_1), \overline{V(M_1)}]$ and $k \geq \frac{1}{\epsilon' b}$, we have

$$|M_2| \geq \mu(G') \cdot \frac{k \cdot \lfloor kb \rfloor}{k + \lfloor kb \rfloor} \geq \frac{\lfloor kb \rfloor}{b+1} \cdot |M_1^*| \geq \frac{kb(1-\epsilon')}{b+1} \cdot |M_1^*|. \tag{3}$$

Combining equations (1), (2) and (3), we obtain the following lower bound on our output estimate.

$$\begin{aligned}
(1-\delta) \cdot |M_1| + (\delta/k) \cdot |M_2| &\overset{(1),(3)}{\geq} (1-\delta) \cdot \left(|M_2^*| + (1/2) \cdot |M_1^*|\right) + (\delta/k) \cdot \frac{kb(1-\epsilon')}{b+1} \cdot |M_1^*| \\
&= (1 - 1/b) \cdot |M_2^*| + \left(\frac{1-1/b}{2} + \frac{1-\epsilon'}{b+1}\right) \cdot |M_1^*| \\
&\geq (|M_1^*| + |M_2^*|) \cdot \min\left\{1 - 1/b, \frac{1-1/b}{2} + \frac{1-\epsilon'}{b+1}\right\} \\
&\overset{(2)}{\geq} (1-\epsilon') \cdot \mu(G) \cdot \min\left\{1 - 1/b, \frac{1-1/b}{2} + \frac{1-\epsilon'}{b+1}\right\} \\
&\geq (1 - 2\epsilon') \cdot \mu(G) \cdot \min\left\{1 - 1/b, \frac{1-1/b}{2} + \frac{1}{b+1}\right\} \\
&= (1 - 2\epsilon') \cdot (2 - \sqrt{2}) \cdot \mu(G),
\end{aligned}$$

where the last equality follows by our choice of $b = 1 + \sqrt{2}$.

Thus, combining with Lemma 3.1, and using that $\epsilon' = \epsilon/8 < 1/8$, we find that the output matching size estimate $\nu := (1-\delta) \cdot |M_1| + (\delta/k) \cdot |M_2|$ is indeed a $\left(1 + \frac{1}{\sqrt{2}} + \epsilon\right)$-approximation, since

$$\nu \leq \mu(G) \leq \nu \cdot \left(\frac{1}{(2-\sqrt{2}) \cdot (1-2\epsilon')}\right) \leq \nu \cdot \left(\left(1 + \frac{1}{\sqrt{2}}\right) \cdot (1 + 4\epsilon')\right) \leq \nu \cdot \left(1 + \frac{1}{\sqrt{2}} + \epsilon\right),$$

as desired. □

Remark 3.4. *A direct extension of the tight example of [65] proves that this analysis is tight, up to the exact dependence on ϵ.*

Lemma 3.1 implies a two-pass streaming algorithm for computing a $(1 + \frac{1}{\sqrt{2}} + \epsilon)$-approximate maximum matching: simply store $G[M_1 \cup M_2]$ and output a maximum matching in this subgraph by the stream's end. The space used in the first and second passes are $\tilde{O}(n)$ and $\tilde{O}(nkb) = \tilde{O}(n/\epsilon)$, respectively. More interestingly for our goals, we show in the next section that Lemma 3.3 can be used to obtain a *dynamic* approximation of the same quality, in polylogarithmic update time.

3.2 Dynamic Algorithm

In this section, we show how to (approximately) implement Algorithm 1 in polylogarithmic update time in a dynamic setting.

THEOREM 3.5. *Let $\epsilon \in (0,1)$. There exists a robust dynamic algorithm $\mathcal{A}$ with worst-case update time $t_u = \tilde{O}_\epsilon(1)$ w.h.p. and query time $t_q = \tilde{O}_\epsilon(n)$ that outputs w.h.p. a value $\nu \in [\mu(G)/(1+\frac{1}{\sqrt{2}}+\epsilon), \mu(G)]$. That is, it answers $(1+\frac{1}{\sqrt{2}}+\epsilon)$ approximate matching size estimate queries.*

PROOF. The dynamic algorithm $\mathcal{A}$ is based on Lemma 1. Let $\epsilon' = \epsilon/12$. Throughout the updates, Algorithm $\mathcal{A}$ simply maintains an $(\epsilon'/8)$-AMM in G, denoted by M_1, invoking Lemma 2.5. This immediately implies the desired update time of $t_u = \tilde{O}_\epsilon(1)$.

We now describe how Algorithm $\mathcal{A}$ responds to a query about the maximum matching size. To answer this query, the algorithm considers a new auxiliary graph $G^* = (V^*, E^*)$, which is defined as follows. Set $b = 1+\sqrt{2}$. For each $u \in V(M_1)$, create $k := \lceil \frac{8}{\epsilon' b} \rceil$ copies of the node u in G^*. Next, for each $v \in \overline{V(M_1)}$, create $\lfloor k \cdot b \rfloor$ copies of the node v in G^*. Finally, for every edge $(u,v) \in G\left[V(M_1), \overline{V(M_1)}\right]$, create an edge in G^* between every pair (u^*, v^*) of copies of the nodes u, v. Note that there is a one-to-one mapping between maximal matchings in the new graph G^* and maximal b-matchings in $G\left[V(M_1), \overline{V(M_1)}\right]$.

We emphasize that our dynamic algorithm $\mathcal{A}$ does not explicitly maintain the auxiliary graph G^*. When we receive a query about the maximum matching size in G, we explicitly construct only the node-set V^* of G^*, based on the matching M_1. This takes only $O_\epsilon(n)$ time since $|V^*| = O_\epsilon(n)$. We can, however, access the edges of G^* by using adjacency matrix queries: there exists an edge (u^*, v^*) in G^* iff there exists an edge (u,v) between the corresponding nodes in G.

At this point, we invoke Proposition 2.2 with $G = G^*$ and precision parameter $\epsilon'' = (\epsilon')^3$. This gives us a value ψ, which is an estimate of $|M_2|$. We now return $\nu := (1-1/b)\cdot|M_1| + (1/bk)\cdot\psi$ as our estimate of $\mu(G)$. Clearly, this entire procedure for answering a query takes $\tilde{O}_\epsilon(n)$ time. It now remains to analyze the approximation ratio. Towards this end, we again appeal to Proposition 2.2. This proposition asserts that the value ψ satisfies

$$|M_2| \geq \psi \geq |M_2| - \epsilon''\cdot|V^*| \geq |M_2| - (\epsilon')^2\cdot 16n \geq |M_2| - \epsilon'\cdot\mu(G), \tag{4}$$

Therefore, our estimate ν satisfies that $\nu' \geq \nu \geq \nu' - \epsilon\cdot\mu(G)$, where

$$\nu' := (1-1/b)\cdot|M_1| + (1/bk)\cdot|M_2|. \tag{5}$$

Now, by Lemma 3.3 and our choice of $k = \lceil \frac{8}{\epsilon' b} \rceil$ and $b = 1+\sqrt{2}$, we have that $\nu' \leq \mu(G) \leq \nu'\cdot\left(1+\frac{1}{\sqrt{2}}+\epsilon\right)$, from which we obtain that $\nu \leq \nu' \leq \mu(G)$ and moreover

$$\mu(G) \leq \nu'\cdot\left(1+\frac{1}{\sqrt{2}}+\epsilon'\right) \leq (\nu+\epsilon'\cdot\mu(G))\cdot\left(1+\frac{1}{\sqrt{2}}+\epsilon'\right) \leq \nu\cdot\left(1+\frac{1}{\sqrt{2}}+\epsilon'\right) + 3\epsilon'\cdot\mu(G).$$

Rearranging terms, and using that $\epsilon' = \epsilon/16 \leq 1/12$, we have that

$$\nu \leq \mu(G) \leq \nu\cdot\left(1+\frac{1}{\sqrt{2}}+\epsilon'\right)/(1-3\epsilon) \leq \nu\cdot\left(1+\frac{1}{\sqrt{2}}+\epsilon\right)\cdot(1+4\epsilon) \leq \nu\cdot\left(1+\frac{1}{\sqrt{2}}+16\epsilon'\right).$$

That is, since $\epsilon' = \epsilon/16$, the estimate ν output after a query is $\left(1+\frac{1}{\sqrt{2}}+\epsilon\right)$-approximate, w.h.p. □

Combining Theorem 3.5 and Propostion 2.1, we obtain our result for bipartite graphs.

THEOREM 3.6. *For any $\epsilon \in (0,1)$, there exists a $(1+\frac{1}{\sqrt{2}}+\epsilon) \approx (1.707+\epsilon)$-approximate randomized dynamic bipartite matching size algorithm with $\tilde{O}_\epsilon(1)$-update time. The algorithm's approximation ratio holds w.h.p. against an adaptive adversary.*

Remark 3.7. *By the same approach as the recent dynamic weighted matching framework of [20] restricted to bipartite graphs, Theorem 3.6 implies a* $\left(1+\frac{1}{\sqrt{2}}+\epsilon\right)$ *robust approximation for* weighted *bipartite matching with the same update time, up to an exponential blowup in the dependence on* ϵ.[5]

4 ALGORITHMS ON GENERAL GRAPHS

In this section we present our main result: a robust dynamic algorithm maintaining a $(2-\Omega(1))$-approximation to the size of the maximum matching in a general graph in worst-case $\tilde{O}_\epsilon(1)$ update time. As with our algorithm for bipartite graphs, we start with a two-pass semi-streaming algorithm in Section 4.1, and then show how to approximately implement it dynamically in Section 4.2.1. Note that unlike in our implementation for bipartite graphs, we will not be able to refer to Proposition 2.2 in a black-box way to estimate the size of the second matching, defined by the matched and unmatched node sets of the dynamic matching we explicitly maintain. Instead, we unbox the proof of Proposition 2.2 in Lemma 4.6, so that it can handle queries suitable for our implementation for non-bipartite graphs. Finally, in Section 4.2.2 we show that our approach allows us to improve the approximation of any algorithm with approximation ratio in the range $(1.5, 2]$.

4.1 Two-Pass Streaming Algorithm

The key challenge in extending Algorithm 1 and its analysis to non-bipartite graphs is its reliance on the integrality of the fractional matching polytope in bipartite graphs. This allowed us to focus on proving the existence of a large fractional matching, which guarantees the existence of a large integral matching of (at least) the same size. For general graphs this argument fails, and so instead we search for length-three augmenting paths (3-augmenting paths, for short) with respect to our first matching, M_1, by computing some large b-matching M_2 in the edge set $V(M_1) \times \overline{V(M_1)}$. The main difficulty with this approach in general graphs is that both the endpoints of an edge $(u,v) \in M_1$ might get matched (in M_2) to the same node w, and the resulting triangle $w-u-v-w$ does not help us in any way to create a 3-augmenting path involving the edge $(u,v) \in M_1$.

We overcome this difficulty using *random bipartitions* (see Algorithm 2). As before, in the first pass we compute an $(\epsilon/4)$-AMM M_1 in the input graph G.[6] Next, we define the following random bipartition (L,R) of the node-set V. For each matched edge $(u,v) \in M_1$, we arbitrarily include one of its endpoints in L and the other in R. Next, for each unmatched node $v \in \overline{V(M_1)}$, we include the node v in into one of L and R chosen uniformly at random. Subsequently, we assign a capacity $b_v := 1$ to all nodes $v \in V(M_1)$ and a capacity $b_v := b$ to all nodes $v \in \overline{V(M_1)}$, for some integer b to be chosen later. Let $B = (V, E_2)$ be the bipartite subgraph spanned by edges with a single node matched in M_1 and endpoints in opposite sides, i.e.,

$$E_2 := \{(u,v) \in E \mid u \in V(M_1),\ v \in \overline{V(M_1)},\ |\{u,v\} \cap L| = |\{u,v\} \cap R| = 1\}).$$

In the second pass, we compute a maximal b-matching M_2 in B w.r.t. the capacities $\{b_v\}$. Finally, we return the maximum matching in the subgraph $G[M_1 \cup M_2]$.

[5] This extension is not obtained by using the framework of [20] directly, as the latter requires explicit dynamic matchings. Nonetheless, their arguments can be extended to the value version of the problem.

[6] As with the bipartite Algorithm 1, we suggest the reader think of M_1 as being maximal for now (i.e., $\epsilon = 0$).

Algorithm 2 General Two-Pass Streaming Algorithm

$M_1 \leftarrow (\epsilon/4)$-AMM computed from first pass
for edge $(u, v) \in M_1$ **do**
    $s(u) \leftarrow \ell$ and $s(v) \leftarrow r$
for vertex $w \in \overline{V(M_1)}$ **do**
    $s(w) \sim \text{Uni}\{\ell, r\}$
let $L \leftarrow \{v \mid s(v) = \ell\}$ and $R \leftarrow \{v \mid s(v) = r\}$
Assign each vertex v capacity $b_v = \begin{cases} 1 & v \in V(M_1) \\ b & v \in \overline{V(M_1)}. \end{cases}$
let $B \leftarrow (V, E_2)$, for $E_2 = \{(u, v) \in E \mid u \in V(M_1), v \in \overline{V(M_1)}, |\{u, v\} \cap L| = |\{u, v\} \cap R| = 1\})$.
$M_2 \leftarrow$ maximal b-matching in B computed from second pass
Output maximum matching in $G[M_1 \cup M_2]$

Intuition: The intuition behind Algorithm 2 is as follows: if M_1 is only roughly 2-approximate, then many 3-augmenting paths exist in G w.r.t. M_1, by Proposition 2.6. Now, a constant fraction of these (specifically, a quarter) "survive" the random bipartition and have their extreme edges belong to B. Now, for each augmenting path $u' - u - v - v'$ that survives, either an augmenting path containing $u - v$ is found in $M_1 \cup M_2$, or at least one of u' or v' is matched b times to nodes in $V(M_1)$ other than u or v. Next, since nodes in $V(M_1)$ can only be matched once in M_2, this limits the number of paths where u and v do not participate in an augmenting path. This implies a large number of augmenting paths in $M_1 \cup M_2$ that are disjoint in their $V(M_1)$ nodes. Finally, since each node in $\overline{V(M_1)}$ belongs to at most b such paths, some large $\Omega(1/b)$ fraction of these augmenting paths are also disjoint in their $\overline{V(M_1)}$ nodes, from which we conclude that $M_1 \cup M_2$ contains a large set of node-disjoint augmenting paths w.r.t. M_1, and that $G[M_1 \cup M_2]$ contains a large matching.

We now substantiate the above intuition. The first lemma in this vein asserts that $M_1 \cup M_2$ contains many 3-augmenting paths w.r.t. M_1 (assuming M_1 is not already near maximum in size).

Lemma 4.1. *If $|M_1| = \left(\frac{1}{2} + c\right) \cdot \mu(G)$, then $G[M_1 \cup M_2]$ contains a set $\mathcal{P}$ of 3-augmenting paths w.r.t. M_1 that are disjoint in their $V(M_1)$ nodes, with expected cardinality at least*

$$\mathbb{E}[|\mathcal{P}|] \geq \left(\frac{b}{b+1}\right) \cdot \left(\frac{1}{4} \cdot \left(\frac{1}{2} - 3c\right) - \frac{1}{b} \cdot \left(\frac{1}{2} + c\right) - \frac{7\epsilon}{8}\right) \cdot \mu(G).$$

As the proof of Lemma 4.1 is a little calculation heavy, we defer its proof to Appendix C, and instead prove the following slightly weaker but simpler bound here.

Lemma 4.2. *If $|M_1| = \left(\frac{1}{2} + c\right) \cdot \mu(G)$, then $G[M_1 \cup M_2]$ contains a set $\mathcal{P}$ of 3-augmenting paths w.r.t. M_1 that are disjoint in their $V(M_1)$ nodes, with expected cardinality at least*

$$\mathbb{E}[|\mathcal{P}|] \geq \left(\frac{1}{4} \cdot \left(\frac{1}{2} - 3c - \frac{7\epsilon}{2}\right) - \frac{2}{b} \cdot \left(\frac{1}{2} + c\right)\right) \cdot \mu(G).$$

Proof. Fix a maximum set of node-disjoint 3-augmenting paths in G w.r.t. M_1, denoted by $\mathcal{P}^*$. By Proposition 2.6, we have $|\mathcal{P}^*| \geq \left(\frac{1}{2} - 3c - \frac{7\epsilon}{2}\right) \cdot \mu(G)$. Next, let $S \subseteq \mathcal{P}^*$ be the paths $u' - u - v - v'$ who "survive" the bipartition, in that $(u, u'), (v, v') \in E_2$. By construction, each path in $\mathcal{P}^*$ survives with probability exactly $\frac{1}{4}$. Therefore, $\mathbb{E}[|S|] \geq \frac{1}{4} \cdot \left(\frac{1}{2} - 3c - \frac{7\epsilon}{2}\right) \cdot \mu(G)$.

Now, for each survived path $u' - u - v - v' \in S$, either both u and v are matched (exactly once) in M_2, thus contributing an augmenting path, or at least one of u' and v' must be matched in M_2 to b distinct nodes in $V(M_1)$. But since each vertex in $V(M_1)$ is matched at most once in M_2,

there are thus at most $|V(M_1)|/b = 2|M_1|/b$ paths in S whose middle edges do not belong to a 3-augmenting path in $M_1 \cup M_2$. Therefore, there are at least $|S| - 2|M_1|/b$ many edges (u, v) in M_1 whose endpoints are both matched in M_2 to some (different) nodes u'' and v'', respectively. Each such edge contributes an augmenting path to a set $\mathcal{P}$ of the desired size,

$$\mathbb{E}[|\mathcal{P}|] = \mathbb{E}[|S|] - \frac{2|M_1|}{b} \geq \left(\frac{1}{4} \cdot \left(\frac{1}{2} - 3c - \frac{7\epsilon}{2}\right) - \frac{2}{b} \cdot \left(\frac{1}{2} + c\right)\right) \cdot \mu(G). \qquad \square$$

The preceding two lemmas imply the existence of a multitude of 3-augmenting paths that are disjoint in their $V(M_1)$ nodes. We now use these augmenting paths to prove the existence of numerous (though possibly fewer) augmenting paths that are disjoint in *all* their nodes. Since each of the two $\overline{V(M_1)}$ nodes of a 3-augmenting path belong to at most b such paths, it is easy to find some $1/(2b-1)$ fraction of these augmenting paths that are disjoint in all their nodes. The following lemma, resembling [48, Lemma 6], increases this fraction to $1/b$.

Lemma 4.3. *Let $\mathcal{P}$ be a set of 3-augmenting paths w.r.t. M_1 in $G[M_1 \cup M_2]$ such that each $V(M_1)$ (resp. $\overline{V(M_1)}$) node belongs to at most one (resp., b) paths in $\mathcal{P}$. Then $\mathcal{P}$ contains a set of node-disjoint 3-augmenting paths $\mathcal{P}' \subseteq \mathcal{P}$ of cardinality at least $|\mathcal{P}'| \geq \frac{1}{b} \cdot |\mathcal{P}|$.*

Proof. Consider the graph $G' = (\overline{V(M_1)}, E')$ obtained by replacing each path $u' - u - v - v'$ in $\mathcal{P}$ with a single edge $u' - v'$. This graph G' is bipartite, by virtue of our random bipartition of G. Now, since this bipartite graph G' has maximum degree b, it contains a matching of size at least $|E'|/b = |\mathcal{P}|/b$: the fractional matching assigning values $1/b$ to each edge has value $|E'|/b$, and so G' contains an *integral* matching of at least the same value. On the other hand, disjoint edges in G' have a one-to-one mapping to node-disjoint paths in $\mathcal{P}$, since each node in $V(M_1)$ belongs to at most one such path. Thus, the maximum matching in G' corresponds to a collection $\mathcal{P}' \subseteq \mathcal{P}$ of node-disjoint augmenting paths in $G[M_1 \cup M_2]$ w.r.t. M_1, of cardinality at least $|\mathcal{P}'| \geq |\mathcal{P}|/b$. $\square$

The three preceding lemmas imply that $G[M_1 \cup M_2]$ contains a large set of vertex-disjoint 3-augmenting paths w.r.t. M_1, assuming this latter matching is not already large. As we now show, this implies that $G[M_1 \cup M_2]$ contains a better-than-2-approximate matching.

Theorem 4.4. *Let $\epsilon \in (0, 1/4)$. Then, Algorithm 2 with $b = 9$ satisfies $\mu(G) \geq \mathbb{E}[\mu(G[M_1 \cup M_2])] \geq \left(\frac{1}{2} + \frac{1}{144} - \epsilon\right) \cdot \mu(G)$, and is thus $\left(\frac{1}{2} + \frac{1}{144} - \epsilon\right)^{-1} < 1.973(1 + 2\epsilon)$-approximate in expectation.*

Proof. Let $|M_1| = \left(\frac{1}{2} + c\right) \cdot \mu(G)$, where $c \in [-\epsilon/8, 1/2]$, with the lower bound on c following from Observation 2.4 and M being an $(\epsilon/4)$-AMM. Let $\mathcal{P}'$ be a maximum set of vertex-disjoint 3-augmenting paths w.r.t. M_1 in $G[M_1 \cup M_2]$. Then, augmenting along these paths, we find that $M_1 \bigoplus \mathcal{P}'$ contains a matching (hence of size at most $\mu(G)$) of the desired expected cardinality.

$$\begin{aligned}
\mathbb{E}[|M_1 \bigoplus \mathcal{P}'|] &= \mathbb{E}[|M_1| + |\mathcal{P}'|] \\
&\geq \left(\frac{1}{2} + c\right) \cdot \mu(G) + \left(\frac{1}{b+1} \cdot \left(\frac{1}{4} \cdot \left(\frac{1}{2} - 3c\right) - \frac{1}{b} \cdot \left(\frac{1}{2} + c\right) - \frac{7\epsilon}{8}\right)\right) \cdot \mu(G) \\
&\geq \left(\frac{1}{2} - \frac{\epsilon}{8} + \frac{1}{b+1} \cdot \left(\frac{1}{4} \cdot \frac{1}{2} - \frac{1}{b} \cdot \frac{1}{2}\right) - \frac{7\epsilon}{8}\right) \cdot \mu(G) \\
&= \left(\frac{1}{2} + \frac{1}{144} - \epsilon\right) \cdot \mu(G).
\end{aligned}$$

Above, the first inequality follows from Lemma 4.1 and Lemma 4.3, the second inequality mainly relies on the parenthetical expression being increasing in $c \geq -\epsilon/8$ (for our choice of $b = 9$). Finally, the equality holds by our choice of $b = 9$. $\square$

4.2 Dynamic Algorithms

In this section we provide the dynamic algorithms yielding our main results, theorems 1.1 and 1.2. As with the bipartite case, our general approach is to approximately implement our two-pass streaming algorithm in a dynamic setting. Unlike the algorithm for bipartite graphs, here we need to (slightly) unbox the sublinear-time algorithm of [12] to find a large set of edges in M_1 which belong to 3-augmenting paths in $G[M_1 \cup M_2]$, as explained below.

4.2.1 Breaking the barrier of two in polylog time. In this section, we present a robust dynamic $(1.973+\epsilon)$-approximate maximum matching size with worst case update time of $t_u = \tilde{O}_\epsilon(1)$, and a query time of $t_q = \tilde{O}_\epsilon(n)$, provided $\mu(G) \geq \epsilon n$. This, combined with Proposition 2.1, implies our main result, Theorem 1.1.

For our dynamic (approximate) implementation of Algorithm 1, which works on bipartite graphs, all we needed was to estimate $|M_2|$. In contrast, for our dynamic (approximate) implementation of Algorithm 2, we will need to estimate the size of the set $\mathcal{P}$ as guaranteed by Lemma 4.1. Specifically, we note that the proofs of Lemmas 4.1, 4.3 and 4.4 imply the following observation.

Observation 4.5. *Let $\widehat{M}_1 \subseteq M_1$ be the set of edges in M_1 whose two endpoints are matched in M_2 in Algorithm 2 run with $b = 9$. Then, $|M_1| + \frac{1}{b} \cdot \mathbb{E}\left[|\widehat{M}_1|\right] \geq \frac{\mu(G)}{1.973 \cdot (1+2\epsilon)}$, and also $\mu(G) \geq |M_1| + \frac{1}{b} \cdot |\widehat{M}_1|$.*

To estimate $|\widehat{M}_1|$ efficiently, we make use of the following extension of the algorithm of [12].

Lemma 4.6. *Consider a graph $G' = (V', E')$ with $|V'| = n'$, and a matching M with $V(M) \subseteq V'$ that is not necessarily part of G' (i.e., we might have $M \nsubseteq E'$). For any matching M' in G', let $k_{M'}$ denote the number of edges in M both of whose endpoints are matched in M'. There is an algorithm which, given adjacency matrix query access to the edges of G', w.h.p. runs in $\tilde{O}_\epsilon(n')$ time and returns an estimate $\kappa \in [k_{M'} - \epsilon^2 n', k_{M'}]$ for some maximal matching M' in G'.*

This lemma follows from the work of [12] rather directly, though it requires some unboxing of the results there, due to the organization of that work. We substantiate this lemma in Appendix B.

Given the above, we are now ready to prove the main result of this section, which is summarized in the theorem below.

Theorem 4.7. *For any $\epsilon \in (0, 1/4)$, there exists a robust dynamic matching size estimator algorithm $\mathcal{A}$ with worst-case update time $\tilde{O}_\epsilon(1)$ that, provided $\mu(G) \geq \epsilon \cdot n$, supports $\tilde{O}_\epsilon(n)$-time queries and outputs a $(1.973+\epsilon)$-approximate estimate w.h.p.*

Proof. The dynamic algorithm $\mathcal{A}$ is based on Algorithm 2. For its updates, it maintains an $(\epsilon/4)$-AMM M_1 in the input graph G, using Lemma 2.5, and a balanced binary search tree (BST) of edges in the graph, allowing for logarithmic-time insertion, deletion and edge queries. This immediately implies a worst-case update time of $t_u = \tilde{O}_\epsilon(1)$.

We now describe how Algorithm $\mathcal{A}$ responds to a query about the maximum matching size. To answer this query, the algorithm considers a new auxiliary graph $G^* = (V^*, E^*)$, which is defined as follows. For each node $u \in V(M_1)$, create a node 0_u in G^*. Next, for each node $v \in \overline{V(M_1)}$, create b nodes $1_v, \ldots, b_v$ in G^*. Finally, for every edge $(u, v) \in E_2$, with $u \in V(M_1)$ and $v \in \overline{V(M_1)}$, create an edge $(0_u, i_v)$ in G^* for all $i \in \{1, \ldots, b\}$. Note that there is a one-to-one mapping between maximal matchings in the new graph G^* and maximal b-matchings in B.

We emphasize that our dynamic algorithm $\mathcal{A}$ does not explicitly maintain the auxiliary graph G^*. When we receive a query about the maximum matching size in G, we explicitly construct only the node-set V^* of G^*, based on the matching M_1. This takes only $O(n)$ time. We can, however, simulate adjacency matrix queries in G^* efficiently: there exists an edge $(0_u, i_v)$ in G^* iff there exists

an edge (u, v) between the corresponding nodes in G, verifiable in $O(\log n)$ time using our edge-set BST.

At this point, we estimate the size of $\widehat{M}_1$ by invoking Lemma 4.6 with $G' = G^*$ and $M = M_1$. This gives us, in time $\tilde{O}_\epsilon(n)$ a value κ satisfying $\kappa \in [|\widehat{M}_1| - \epsilon^2 n, |\widehat{M}_1|]$, w.h.p. We now return $\nu := |M_1| + \frac{1}{b} \cdot \kappa$ as our estimate of $\mu(G)$. All in all, our algorithm has query time $t_q = \tilde{O}_\epsilon(n)$.

It remains to analyze the approximation ratio. For this, we observe that, by our hypothesis that $\mu(G) \geq \epsilon \cdot n$,

$$\begin{aligned}
\mathbb{E}[\nu] &\geq \mathbb{E}\left[|M_1| + \frac{1}{b} \cdot (|\widehat{M}_1| - \epsilon^2 n)\right] \\
&= |M_1| + \frac{1}{b} \cdot \mathbb{E}\left[\widehat{M}_1\right] - \frac{\epsilon^2 n}{b} \\
&\geq \frac{\mu(G)}{1.973 \cdot (1 + 2\epsilon)} - \epsilon^2 n \\
&\geq \frac{\mu(G)}{1.973 \cdot (1 + 2\epsilon)} - \epsilon \cdot \mu(G) \\
&\geq \frac{\mu(G)}{1.973 \cdot (1 + 5\epsilon)}. \qquad (6)
\end{aligned}$$

In the above derivation, the second inequality follows from Observation 4.5. Similarly, we have w.h.p.:

$$\nu \leq |M_1| + \frac{1}{b} \cdot |\widehat{M}_1| \leq \mu(G). \qquad (7)$$

The second inequality in the above derivation again follows from Observation 4.5. From (6) and (7), we conclude that we return in response to each query a $1.973(1 + 5\epsilon)$-approximation to $\mu(G)$ in expectation. Therefore, by standard Chernoff bounds, running $O_\epsilon(\log n)$ copies of this algorithm (increasing update and query time appropriately) and taking the average of these will then result in a $1.973(1 + O(\epsilon))$ approximation of the desired value, w.h.p. Reparameterizing ϵ appropriately, the theorem follows. □

Combined with Observation 2.1, the above theorem implies our main result, Theorem 4.7.

4.2.2 New time/approximation tradeoffs. In this section we show our secondary result: a black-box method to improve dynamic matching algorithm's approximation ratio, at the cost of only outputting a size estimate. We start with the following observation.

Proposition 4.8. *Let G be an n-node graph, $\epsilon \in (0, 1)$ and $\alpha \geq 1$. Then, given an ϵ-AMM M' and α-approximate maximum matching M'' in G, one can compute in $O(n)$ time a matching M in G which is both α-approximate and an ϵ-AMM.*

Proof. The subgraph $G[M' \cup M'']$ has maximum degree two, and is thus the union of paths and cycles. Let M be the matching obtained by taking from each connected component C in $G[M' \cup M'']$ either the set of edges of M' or M'' that are most plentiful in C, breaking ties in favor of M'. By construction, it is clear that M is a matching, and that moreover $|M| \geq |M''|$, and so M is α-approximate. On the other hand, M matches all nodes of M' in each component, and therefore overall. That is, after removing at most $\epsilon\mu(G)$ nodes in $V \setminus V(M) \subseteq V \setminus V(M')$, we obtain a graph in which M is maximal. That is, the matching M is also an ϵ-AMM. □

We are now ready to prove Theorem 1.2, restated below for ease of reference.

Theorem 1.2. *For any $\alpha > 1.5$, a dynamic α-approximate matching algorithm with update time t_u implies a dynamic $\left(\alpha - \Omega\left(\left(1 - 6\left(\frac{1}{\alpha} - \frac{1}{2}\right)\right)^2\right)\right)$-approximate matching* size estimator *with update time $\tilde{O}(t_u)$.*

Proof. Let $\epsilon > 0$ be some sufficiently small constant. We describe how to obtain a new dynamic matching size estimator $\mathcal{A}$ for G, with update time $\tilde{O}_\epsilon(t_u)$ that, provided $\mu(G) \geq \epsilon \cdot n$, supports $\tilde{O}_\epsilon(n)$-time queries and outputs a β-approximate estimate w.h.p., for some $\beta = \alpha - \Omega((1-6(1/\alpha - 1/2))^2))$. The theorem then immediately follows from Proposition 2.1.

The algorithm $\mathcal{A}$ works as follows. It maintains an ϵ-AMM M_1', invoking Lemma 2.5, taking $\tilde{O}_\epsilon(1)$ update time. It also maintains α-approximate matching M_1'', by running the dynamic algorithm guaranteed by the theorem's hypothesis, taking t_u update time. Therefore, Algorithm $\mathcal{A}$ has an overall update time of $t_u + \tilde{O}_\epsilon(1) = \tilde{O}_\epsilon(t_u)$.

Upon receiving a query, Algorithm $\mathcal{A}$ first invokes Proposition 4.8 to obtain a matching M_1 (based on M_1' and M_1'') that is simultaneously an ϵ-AMM and an α-approximate maximum matching in G. This takes $O(n)$ time. The rest of the query algorithm remains exactly the same as in Section 4.2. This implies that Algorithm $\mathcal{A}$ has an overall query time of $\tilde{O}_\epsilon(n)$.

We now analyze the approximation guarantee of $\mathcal{A}$. Towards this end, observe that as $\alpha > 1.5$, we can write $\frac{1}{\alpha} = \frac{1}{2} + c$ where $0 < c < 1/6$. So, we have that $|M_1| = \left(\frac{1}{2} + z\right) \cdot \mu(G)$ for some $z \geq c$. Therefore, by Lemma 4.2 and Lemma 4.3, there exists some set $\mathcal{P}'$ of node-disjoint length-three augmenting paths w.r.t. M_1 in $G[M_1 \cup M_2]$ whose cardinality satisfies

$$\frac{|\mathcal{P}'|}{\mu(G)} \geq f_b(z) := \frac{1}{b} \cdot \left(\frac{1}{4} \cdot \left(\frac{1}{2} - 3z - \frac{7\epsilon}{2}\right) - \frac{2}{b} \cdot \left(\frac{1}{2} + z\right)\right).$$

Augmenting along these paths with respect to M_1, we obtain a new matching in $G[M_1 \cup M_2]$ of cardinality at least $\left(\frac{1}{2} + z + f_b(z)\right) \cdot \mu(G)$. Now, for $b \geq 2$ (as we will choose), this matching size is decreasing in z, as observed by taking the derivative of $1/2 + z + f_b(z)$ w.r.t. z. Therefore, the matching size is minimized at $z = c$, and we find that $\mu(G[M_1 \cup M_2]) \geq 1/2 + c + f_b(c)$. Taking another derivative, this time with respect to b, we find that this expression is minimized (ignoring the ϵ dependence) at $b^* = \frac{16(1+2c)}{1-6c}$. Note that $b \geq 16$, as $c \in (0, 1/6)$. This optimal b^* need not be an integer, however, and so we take $b = \lceil b^* \rceil \leq \frac{17}{16} b^*$ in our algorithm, and find that $M_1 \cup M_2$ contains a matching of size at least $\mu(G)$ times $1/2+z+f_b(z) \geq 1/2+c+f_{\frac{17b}{16}b^*}(c) \geq 1/2+c+\frac{9(1-6c)^2}{2312(1+2c))} - O(\epsilon/b)$. Moreover, some $b \cdot f_b(z)$ many edges $\hat{M}_1 \subseteq M_1$ have both of their endpoints matched in the b-matching M_2.

We conclude that $\mathbb{E}[\mu(G[M_1 \cup M_2])]$ gives a strictly-better-than-α approximation to $\mu(G)$ (again using that $c < 1/6$). Specifically, the gain we get in the approximation ratio is of the order of $\Theta((1-6c)^2) = \Theta((1-6(1/\alpha - 1/2))^2)$. Now, using the fact that $\mu(G) \geq \epsilon n$ and we are running the same query algorithm as in Section 4.2, our estimation using the sub-linear-time algorithm (Lemma 4.6) gives a strictly-better-than-α approximation to $\mu(G)$ in expectation. As before, taking the average of $O(\log n)$ copies of this algorithm will provide the same bound w.h.p., at an additional logarithmic multiplicative overhead to the update and query times. □

Remark 4.9. *We note that the reduction of Theorem 1.2 preserves robustness and worst-case update time.*

5 AMMS AGAINST ADAPTIVE ADVERSARIES

In this section we prove Lemma 2.5. That is, we provide a robust dynamic algorithm for maintaining an ϵ-AMM in worst-case polylogarithmic update time. But first, we motivate our algorithm, and characterize the kind of matching we wish to compute.

We first recall a useful tool in the literature, namely *edge* sparsification: maintaining a sparse subgraph of G containing a large matching–a so-called *matching sparsifier*. Such sparsifiers naturally allow to achieve speedups in the algorithms needed for Proposition 2.1, as a large matching in a sparsifier can be computed quickly. One influential such sparsifier that we will use are *kernels*, introduced by Bhattacharya et al. [25].

Definition 5.1. *For $\epsilon \geq [0,1]$ and $d \in \mathbb{N}$, a subgraph $K = (V, E_K)$ of graph $G = (V, E)$ is an (ϵ, d)-*kernel *if K's maximum degree is at most d and each edge $e \in E \setminus E_K$ has at least one endpoint of degree at least $d(1-\epsilon)$ in K.*

These sparsifiers will play an integral role in robustly and efficiently maintaining an AMM in this section. We start by motivating their use in computing AMMs in a *static* setting.

5.1 From kernels to AMMs

To motivate the interest in bounded-degree graphs, we recall the following observation, which follows from the fact that the $2\mu(G)$ endpoints of a maximum matching form a vertex cover (i.e., are incident on each edge of the graph).

Fact 5.2. *Let $G = (V, E)$ be a graph of maximum degree Δ. Then $|E| \leq 2\mu(G) \cdot \Delta$.*

When d and $\mu(G)$ are small, Fact 5.2 and Definition 5.1 imply that (ϵ, d)-kernels of G are quite sparse subgraphs compared to the size of a maximum matching contained within them. In particular, since maximal matchings are computable in linear time, in kernel K computing a maximal matching requires only $|E_K| \leq 2\mu(K) \cdot d = O(\mu(G) \cdot d)$ time. The following result of [41] implies that essentially the same amount of time is needed to compute a *near-maximum-weight* matching in K.

Proposition 5.3. *Let $G = (V, E, w)$ be a weighted graph. There exist a deterministic algorithm for computing a $(1+\epsilon)$-approximate maximum weight matching of G in time $O_\epsilon(|E|)$.*

We now explain how to find large matchings in a kernel K that allow us to obtain an AMM of G. For this, we will need to upper bound the number of high-degree nodes in K. Specifically, for an (ϵ, d)-kernel K of graph G, we denote by $H_K := \{v \mid d_K(v) \geq d(1-\epsilon)\}$ the set of high-degree nodes in K. We will wish to argue that a removal of few high-degree nodes in the kernel yields a subgraph in which our matching is maximal. We therefore need to prove that the number of high-degree nodes is itself small in terms of $\mu(G)$.

Lemma 5.4. *Let $K = (V, E_K)$ be an (ϵ, d)-kernel K of G with $\epsilon \leq 1/4$. Then $|H_K| \leq 4\mu(G)$.*

Proof. We consider the fractional matching $x \in \mathbb{R}^E$ where $x_e = \mathbb{1}[e \in E_K]/d$. By the degree bound of K, this is a feasible fractional matching in G. Using this fractional matching, we can show that

$$\frac{1}{2} \cdot (1-\epsilon) \cdot |H_K| \leq \sum_{v \in H_K} \sum_{e \ni v} x_e \leq \sum_e x_e \leq \frac{3}{2} \cdot \mu(G),$$

where the first inequality follows from the definition of H_K and possible double counting of edges, and the last inequality follows from the integrality gap of $\frac{3}{2}$ of the factional matching polytope. Simplifying the above and using $\epsilon \leq 1/4$, we have that indeed $|H_K| \leq (3/(1-\epsilon)) \cdot \mu(G) \leq 4\mu(G)$. □

We now show that it is sufficient to match the $(1-\Theta_\epsilon(1))$-fraction of vertices of H_K in order to find an *AMM* in G through Lemma 5.5. Lemma 5.6 then proves that there exists a static algorithm with running time $\tilde{O}_\epsilon(k \cdot \mu(G))$ for computing an *AMM* of G given access to kernel K.

Lemma 5.5. *Let $K = (V, E_K)$ be an (ϵ, d)-kernel of $G = (V, E)$, for $\epsilon \in (0, 1/4)$ and $d \geq \frac{1}{\epsilon}$. Then, a maximal matching M in K that matches at least a $(1 - c \cdot \epsilon)$-fraction of H_K is a $4c\epsilon$-AMM in G. Moreover, such a matching exists for $c = 2$.*

Proof. First, we argue that such a matching M, if it exists, is indeed a $4c\epsilon$-AMM in G. We recall that every edge in $E \setminus E_K$ has a high-degree endpoint in K. Therefore, if we remove the $c\epsilon$ fraction of high-degree nodes H_K unmatched by M, each edge in $E \setminus E_K$ in the resulting graph G' has at least one endpoint matched in M. On the other hand, every edge in E_K has an endpoint matched in M, by maximality of M in K. We conclude that after removing $c\epsilon \cdot |H_K| \leq 4c\epsilon \cdot \mu(G)$ nodes in G (with the inequality relying on Lemma 5.4), we obtain a graph G' where M is maximal. That is, M is a $4c\epsilon$-AMM.

We now argue the existence of such a matching M for $c = 2$. Since H has maximum degree $d \geq \frac{1}{\epsilon}$, by Vizing's theorem [81] it can be $(d+1)$-edge-colored, i.e., decomposed into $(d+1)$ matchings. A randomly-chosen color in this edge coloring is a matching M' that matches each edge with probability $\frac{1}{d+1}$, and thus it matches each high-degree vertex v with probability at least

$$\Pr[v \text{ matched}] \geq d(1-\epsilon)/(d+1) \geq (1-\epsilon)/(1+\epsilon) \geq 1 - 2\epsilon.$$

Finally, extending this matching M' to also be maximal in K by adding edges of K greedily then proves the existence of the desired 8ϵ-AMM contained in the kernel K. □

We now turn to making the above proof of existence constructive.

Lemma 5.6. *Given an (ϵ, d)-kernel $K = (V, E_K)$ of $G = (V, E)$, one can compute an ϵ-AMM in G in deterministic time $O_\epsilon(d \cdot \mu(G))$.*

Proof. Let $\epsilon' = \epsilon/12$. By Fact 5.2, the number of edges in K is at most $|E_K| = O(d \cdot \mu(G))$. We then compute a $(1+\epsilon')$-max weight matching M' in the graph G with edge weights equaling the number of high-degree nodes incident on them, $w_e = \sum_{v \in e} \mathbb{1}[v \in H_K] \in \{0, 1, 2\}$. By Proposition 5.3, this can be done in deterministic time $O_\epsilon(d \cdot \mu(G))$. By Lemma 5.5, this guarantees that at least a $(1-2\epsilon')/(1+\epsilon') \geq (1-3\epsilon')$ fraction of high-degree nodes in K are unmatched by this dynamic subroutine. We then extend M' to also be maximal in K, by scanning over the $|E_K| = O(d \cdot \mu(G))$ edges of K (in the same deterministic time) and adding them to M' where possible. By Lemma 5.5, this results in a $12\epsilon'$-AMM, i.e., an ϵ-AMM, after a total of $O_\epsilon(d \cdot \mu(G))$ deterministic time. □

So far, we have provided a *static* AMM algorithm with deterministic time $O_\epsilon(d \cdot \mu(G))$, *provided we have access to a kernel*. To dynamize the above, we first show how to maintain a kernel dynamically through periodic recomputation.

5.2 Periodic kernels and AMMs

In [82], Wajc provided a method for rounding dynamic fractional matchings to matching sparsifiers, and from these (by methods underlying Algorithm 3), we can obtain integral matchings. Crucially for our needs, his framework was robust, and allowed for worst-case update times. Unfortunately for us, the lemma statements in his work do not immediately imply a robust dynamic kernel maintenance. However, they do allow for *kernel queries*, with running time $\tilde{O}(d \cdot \mu(G))$.

Lemma 5.7. *Let $\epsilon \in (0, 1)$ and $d = \tilde{O}_\epsilon(1)$ be sufficiently large. Then, there exists a robust algorithm with worst-case update time $t_u = \tilde{O}_\epsilon(1)$ allowing for (ϵ, d)-kernel and ϵ-AMM queries in worst-case query time $t_q = \tilde{O}_\epsilon(d \cdot \mu(G))$. The query's outputs are a kernel and an ϵ-AMM w.h.p.*

Given the ability to query a kernel, the ability to query an AMM then follows directly from Lemma 5.6. As the proof and presentation of an algorithm allowing for kernel queries essentially requires repeating verbatim numerous lemmas in [82], we defer its proof to Appendix D.

We now turn to designing a robust dynamic algorithm that *always* maintains an AMM.

5.3 Robust dynamic AMMs

So far, we have provided a method to answer AMM queries in a dynamic setting. Lemma 5.8 proves that once an AMM is computed it remains an *AMM* for $\sim \epsilon \cdot \mu(G)$ updates.

Lemma 5.8. *Let $\epsilon \in (0, 1/2)$. If M is an ϵ-AMM in G, then the non-deleted edges of M during any sequence of at most $\epsilon \cdot \mu(G)$ updates constitute a 6ϵ-AMM in G (during the updates).*

Proof. Let G, M and G', M' be the graph and matching before and after the updates, respectively. Since each update can decrease the size of the maximum matching size by at most one, we have

$$\frac{1}{2} \cdot \mu(G) \leq (1-\epsilon) \cdot \mu(G) \leq \mu(G').$$

Now, recall that for some set of vertices $U \subseteq V$ of size at most $|U| \leq \epsilon \cdot \mu(G)$ nodes from G, the matching M is maximal in $G[V \setminus U]$. Now, after these $\epsilon \cdot \mu(G)$ updates, it might be that $2\epsilon \cdot \mu(G)$ edges in G' are now not incident on edges in M'. (The factor of two arises due to edges of M that are deleted leaving two uncovered edges, addressable by removing two more nodes). That is, after removing a node set $U' \subseteq V$ of size at most $|U'| \leq 3\epsilon \cdot \mu(G) \leq 6\epsilon \cdot \mu(G')$ nodes from G', we obtain a graph $G'[V \setminus U']$ where M' is maximal. That is, M' is an $O(\epsilon)$-AMM in G'. □

This "stability" of AMMs again lends itself to the periodic re-computation framework of [57], which, together with our algorithms for querying AMMs, allow us to maintain AMMs (always).

Lemma 2.5. *For any $\epsilon \in (0, 1/2)$, there exists a robust dynamic algorithm that w.h.p. maintains an ϵ-AMM in worst-case update time $\tilde{O}_\epsilon(1)$.*

Proof. We will run the dynamic AMM query algorithm $\mathcal{A}$ of Lemma 5.7, whose update fits within our update time budget. We will periodically query $\mathcal{A}$, and spread this computation over these periods to guarantee low worst-case update time. Specifically, we will divide the update sequence into *epochs*, where if the graph G at the start of epoch i is G_i, then the epoch has length $\ell_i \in [\epsilon \cdot \mu(G)/3, \epsilon \cdot \mu(G_i)]$. In order to determine the length of the epochs, we run the deterministic dynamic $(2+\epsilon)$-approximate fractional matching algorithm of [27], which in particular gives us a $2 + \epsilon \leq 3$-approximation of $\mu(G_i)$ in worst-case update time $\tilde{O}_\epsilon(1)$, again fitting within our time budgets. Now, during phase i, we spend the time t_q for the ϵ-AMM query subroutine of $\mathcal{A}$, so as to finish computing M_i. The amount of time spent per update to achieve this goal is at most $\frac{\tilde{O}_\epsilon(\mu(G_i))}{\lfloor \epsilon \cdot \mu(G_i)/10 \rfloor} = \tilde{O}_\epsilon(1)$, again fitting within our time updates. We now describe and analyze the matchings maintained by this algorithm (these are not always M_i).

By Lemma 5.8, we need to provide a matching M'_{i+1} at the start of each phase $i+1$ which is an $O(\epsilon)$-AMM in G_{i+1}, thus guaranteeing that the non-deleted edges of M'_{i+1} remain an $O(\epsilon)$-AMM. Reparameterizing appropriately will then yield the desired result. It remains to define our matchings M'_i. Using our estimate of $\mu(G)$ obtained by the dynamic fractional matching, we test whether $\mu(G_i) \in [1/\epsilon, 10/\epsilon]$. If this is the case, then M'_{i+1} is obtained by querying the AMM algorithm $\mathcal{A}$ at the beginning of phase $i+1$, in time $O_\epsilon(1)$. (This relied on $\mu(G_{i+1}) \leq \mu(G_i) + \ell_i \leq \mu(G_i) \cdot (1+\epsilon) = \tilde{O}_\epsilon(1)$.) By the properties of $\mathcal{A}$, the matching M'_{i+1} is an ϵ-AMM in G_{i+1} w.h.p. Now, if conversely $\mu(G_i) \geq 10/\epsilon$, then we have that

$$\frac{1}{2} \cdot \mu(G_i) \leq (1-\epsilon) \cdot \mu(G_i) \leq \mu(G_i) - \ell_i \leq \mu(G_{i+1}).$$

Now, M_i, is an ϵ-AMM in G_i, which is obtained from G_{i+1} by at most $\epsilon \cdot \mu(G_i) \leq 2\epsilon \cdot \mu(G_{i+1})$ updates. Therefore, by Lemma 5.8 M_i is a 12ϵ-AMM in G_{i+1}. We therefore take M'_{i+1} to be M_i. Reparameterizing ϵ appropriately, the lemma follows. □

6 CONCLUSION AND FUTURE DIRECTIONS

We presented the first dynamic matching (size estimation) algorithm breaking the approximation barrier of 2 in polylogarithmic update time. While this presents a major advance in our understanding of the dynamic matching problem, many questions remain. We mention a few such questions which we find particularly intriguing, and some progress on these following the publication of this paper's conference version.

Explicit fast matching. In our work we show how to maintain a better-than-two approximate estimate of the maximum matching size. Can one also maintain an explicit matching of similar approximation ratio within the same time bounds? We note that unless all greater than 1.5 approximation ratios are possible for explicit matching in polylog time, then Theorem 1.2 would imply a separation between the attainable time/approximate tradeoffs for explicit dynamic matching and its size estimation counterpart.

Better approximation in $o(n)$ update time? Known conditional impossibility results rule out an exact algorithm with $n^{1-\Omega(1)}$ update time [2, 40, 58], but the best approximation ratios currently known are $\frac{3}{2} + \epsilon$ [22, 23, 56, 62]. Can one do better in $o(n)$ time? Following the publication of the conference version of this paper, the sublinear+streaming connection to dynamic algorithms has resulted in better time/approximation tradeoffs for the value version of the problem [10, 18, 29, 30]. Again, one may ask, can explicit matching algorithms with such improved guarantees be obtained? In a similar vein, sub-logarithmic speedups were obtained for $(1 + o(1))$-approximate matching [5]. Can one obtain stronger speedups for $(1 + \epsilon)$-approximate matching? On the flip side, can we show any (conditional) hardness of *approximate* dynamic matching, for any approximation ratio?

Unconditional impossibility results. With this work we bring dynamic matching with better-than-two approximation into the polylogarithmic update time regime—the range where *unconditional* impossibility are known for numerous data structures and dynamic algorithms. Can such unconditional impossibility results be proven for (approximate) dynamic matching?

More applications of robust AMMs. A plethora of recent developments in fast (static) graph algorithms rely on invocations of adversarially-robust dynamic algorithms (i.e., ones that work against an adaptive adversary). Our work adds almost-maximal matchings to the list of algorithmic subroutines useful for this approach. (Subsequently, [32] derandomized this result, though at the cost of amortization.) Robust AMMs played a key role for dynamic problems, both here, in the follow-up [10], and in the bounded-arboricity matching algorithms of [77]. What further applications can our robust dynamic AMM algorithms for arbitrary graphs find for other (dynamic and static) algorithmic problems?

Acknowledgements. We thank the anonymous SODA and J.ACM reviewers for helpful comments.

REFERENCES

[1] Amir Abboud and Søren Dahlgaard. 2016. Popular conjectures as a barrier for dynamic planar graph algorithms. In *Proceedings of the 57th Symposium on Foundations of Computer Science (FOCS)*. 477–486.

[2] Amir Abboud and Virginia Vassilevska Williams. 2014. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 55th Symposium on Foundations of Computer Science (FOCS)*. 434–443.

[3] Kook Jin Ahn and Sudipto Guha. 2013. Linear programming in the semi-streaming model with application to the maximum matching problem. *Information and Computation* 222 (2013), 59–79.

[4] Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. 2018. Dynamic Matching: Reducing Integral Algorithms to Approximately-Maximal Fractional Algorithms. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming (ICALP)*. 79:1–79:16.

[5] Sepehr Assadi, Soheil Behnezhad, Sanjeev Khanna, and Huan Li. 2023. On regularity lemma and barriers in streaming and dynamic matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*. 131–144.

[6] Sepehr Assadi, Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. 2022. Semi-Streaming Bipartite Matching in Fewer Passes and Optimal Space. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 627–669.

[7] Sepehr Assadi and Sanjeev Khanna. 2024. Improved Bounds for Fully Dynamic Matching via Ordered Ruzsa-Szemer\'edi Graphs. *arXiv preprint arXiv:2406.13573* (2024).

[8] Sepehr Assadi, Sanjeev Khanna, and Yang Li. 2019. The stochastic matching problem with (very) few queries. *ACM Transactions on Economics and Computation (TEAC)* 7, 3 (2019), 1–19.

[9] Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. 2016. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1345–1364.

[10] Amir Azarmehr, Soheil Behnezhad, and Mohammad Roghani. 2024. Fully Dynamic Matching: $(2-\sqrt{2})$-Approximation in Polylog Update Time. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 3040–3061.

[11] Surender Baswana, Manoj Gupta, and Sandeep Sen. 2015. Fully Dynamic Maximal Matching in $O(\log n)$ Update Time. *SIAM Journal on Computing (SICOMP)* 44, 1 (2015), 88–113.

[12] Soheil Behnezhad. 2022. Time-optimal sublinear algorithms for matching and vertex cover. In *Proceedings of the 62nd Symposium on Foundations of Computer Science (FOCS)*. 873–884.

[13] Soheil Behnezhad. 2023. Dynamic Algorithms for Maximum Matching Size. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 129–162.

[14] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. 2019. Fully dynamic maximal independent set with polylogarithmic update time. In *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*. 382–405.

[15] Soheil Behnezhad and Alma Ghafari. 2024. Fully Dynamic Matching and Ordered Ruzsa-Szemerédi Graphs. In *Proceedings of the 65th Symposium on Foundations of Computer Science (FOCS)*. To appear.

[16] Soheil Behnezhad and Sanjeev Khanna. 2022. New Trade-Offs for Fully Dynamic Matching via Hierarchical EDCS. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 3529–3566.

[17] Soheil Behnezhad, Jakub Łącki, and Vahab Mirrokni. 2020. Fully Dynamic Matching: Beating 2-Approximation in Δ^{ϵ} Update Time. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2492–2508.

[18] Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. 2023. Sublinear time algorithms and complexity of approximate maximum matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*. 267–280.

[19] Amos Beimel, Haim Kaplan, Yishay Mansour, Kobbi Nissim, Thatchaphol Saranurak, and Uri Stemmer. 2022. Dynamic algorithms against an adaptive adversary: Generic constructions and lower bounds. In *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC)*. 1671–1684.

[20] Aaron Bernstein, Aditi Dudeja, and Zachary Langley. 2021. A Framework for Dynamic Matching in Weighted Graphs. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing (STOC)*. 668–681.

[21] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. 2019. A deamortization approach for dynamic spanner and dynamic maximal matching. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1899–1918.

[22] Aaron Bernstein and Cliff Stein. 2015. Fully Dynamic Matching in Bipartite Graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*. 167–179.

[23] Aaron Bernstein and Cliff Stein. 2016. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 692–711.

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

[25] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F Italiano. 2018. Deterministic fully dynamic data structures for vertex cover and matching. *SIAM Journal on Computing (SICOMP)* 47, 3 (2018), 859–887.

[26] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 2016. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*. 398–411.

[27] 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 *Proceedings of the 28th Annual*

ACM-SIAM Symposium on Discrete Algorithms (SODA). 470–489.

[28] Sayan Bhattacharya and Peter Kiss. 2021. Deterministic Rounding of Dynamic Fractional Matchings. In *Proceedings of the 48th International Colloquium on Automata, Languages and Programming (ICALP)*. 27:1–27:14.

[29] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. 2023. Sublinear Algorithms for (1.5+ ϵ)-Approximate Matching. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*. 254–266.

[30] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. 2024. Dynamic $(1+\epsilon)$-Approximate Matching Size in Truly Sublinear Update Time. In *Proceedings of the 56th Symposium on Foundations of Computer Science (FOCS)*. 1563–1588.

[31] Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. 2023. Dynamic matching with better-than-2 approximation in polylogarithmic update time. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 100–128.

[32] Sayan Bhattacharya, Peter Kiss, Aaron Sidford, and David Wajc. 2024. Near-Optimal Dynamic Rounding of Fractional Matchings in Bipartite Graphs. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC)*. 59–70.

[33] Sayan Bhattacharya and Janardhan Kulkarni. 2019. Deterministically Maintaining a $(2+\epsilon)$-Approximate Minimum Vertex Cover in $O(1/\epsilon^2)$ Amortized Update Time. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1872–1885.

[34] Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. 2019. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*. 456–480.

[35] Moses Charikar and Shay Solomon. 2018. Fully Dynamic Almost-Maximal Matching: Breaking the Polynomial Barrier for Worst-Case Time Bounds. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming (ICALP)*. 33:1–33:14.

[36] Shiri Chechik and Tianyi Zhang. 2019. Fully dynamic maximal independent set in expected poly-log update time. In *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*. 370–381.

[37] Jiale Chen, Aaron Sidford, and Ta-Wei Tu. 2023. Entropy Regularization and Faster Decremental Matching in General Graphs. *arXiv preprint arXiv:2312.09077* (2023).

[38] Julia Chuzhoy and Sanjeev Khanna. 2019. A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut problems. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*. 389–400.

[39] Michael Crouch and Daniel M Stubbs. 2014. Improved Streaming Algorithms for Weighted Matching, via Unweighted Matching. In *Proceedings of the 17th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. 96.

[40] Søren Dahlgaard. 2016. On the Hardness of Partially Dynamic Graph Problems and Connections to Diameter. In *Proceedings of the 43rd International Colloquium on Automata, Languages and Programming (ICALP)*. 48:1–48:14.

[41] Ran Duan and Seth Pettie. 2014. Linear-time approximation for maximum weight matching. *Journal of the ACM (JACM)* 61, 1 (2014), 1.

[42] Devdatt P. Dubhashi and Desh Ranjan. 1998. Balls and bins: A study in negative dependence. *Random Struct. Algorithms* 13, 2 (1998), 99–124.

[43] Aditi Dudeja. 2024. A Note on Rounding Matchings in General Graphs. *arXiv preprint arXiv:2402.03068* (2024).

[44] Jack Edmonds. 1965. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B* 69, 125-130 (1965), 55–56.

[45] Jack Edmonds. 1965. Paths, trees, and flowers. *Canadian Journal of mathematics* 17, 3 (1965), 449–467.

[46] Sebastian Eggert, Lasse Kliemann, and Anand Srivastav. 2009. Bipartite graph matchings in the semi-streaming model. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA)*. 492–503.

[47] Leah Epstein, Asaf Levin, Danny Segev, and Oren Weimann. 2013. Improved Bounds for Online Preemptive Matching. In *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS)*. 389.

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

[49] Matthew Fahrbach, Gary L Miller, Richard Peng, Saurabh Sawlani, Junxing Wang, and Shen Chen Xu. 2018. Graph sketching against adaptive adversaries applied to the minimum degree algorithm. In *Proceedings of the 59th Symposium on Foundations of Computer Science (FOCS)*. 101–112.

[50] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. 2005. On graph problems in a semi-streaming model. *Theoretical Computer Science (TCS)* 348, 2-3 (2005), 207–216.

[51] Moran Feldman and Ariel Szarf. 2022. Maximum Matching sans Maximal Matching: A New Approach for Finding Maximum Matchings in the Data Stream Model. In *Proceedings of the 25th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. 33:1–33:24.

[52] Manuela Fischer, Slobodan Mitrović, and Jara Uitto. 2022. Deterministic $(1+\epsilon)$-approximate maximum matching with poly $(1/\epsilon)$ passes in the semi-streaming model and beyond. In *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC)*. 248–260.

[53] Mohsen Ghaffari and David Wajc. 2019. Simplified and Space-Optimal Semi-Streaming $(2+\epsilon)$-Approximate Matching. In *Proceedings of the 2nd Symposium on Simplicity in Algorithms (SOSA)*. 13:1–13:8.

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

[55] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. 2013. Perfect Matchings in $O(n \log n)$ Time in Regular Bipartite Graphs. *SIAM Journal on Computing (SICOMP)* 42, 3 (2013), 1392–1404.

[56] Fabrizio Grandoni, Chris Schwiegelshohn, Shay Solomon, and Amitai Uzrad. 2022. Maintaining an EDCS in General Graphs: Simpler, Density-Sensitive and with Worst-Case Time Bounds. *Proceedings of the 5th Symposium on Simplicity in Algorithms (SOSA)* (2022), 12–23.

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

[58] 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 *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*. 21–30.

[59] Sagar Kale and Sumedh Tirodkar. 2017. Maximum Matching in Two, Three, and a Few More Passes Over Graph Streams. In *Proceedings of the 20th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. 15:1–15:21.

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

[61] Michael Kapralov. 2021. Space lower bounds for approximating maximum matching in the edge arrival model. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1874–1893.

[62] Peter Kiss. 2022. Improving update times of dynamic matching algorithms from amortized to worst case. *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS)* (2022), 94:1–94:21.

[63] Christian Konrad. 2018. A simple augmentation method for matchings with applications to streaming algorithms. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*. 74:1–74:16.

[64] Christian Konrad, Frédéric Magniez, and Claire Mathieu. 2012. Maximum matching in semi-streaming with few passes. In *Proceedings of the 15th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. 231–242.

[65] Christian Konrad and Kheeran K Naidu. 2021. On Two-Pass Streaming Algorithms for Maximum Bipartite Matching. In *Proceedings of the 24th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. 19:1–19:18.

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

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

[68] Hung Le, Lazar Milenković, Shay Solomon, and Virginia Vassilevska Williams. 2022. Dynamic Matching Algorithms Under Vertex Updates. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS)*. 96:1–96:24.

[69] Yang P Liu. 2024. On approximate fully-dynamic matching and online matrix-vector multiplication. In *Proceedings of the 65th Symposium on Foundations of Computer Science (FOCS)*. To appear.

[70] Andrew McGregor. 2005. Finding graph matchings in data streams. In *Proceedings of the 8th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. 170–181.

[71] Danupon Nanongkai and Thatchaphol Saranurak. 2017. Dynamic spanning forest with worst-case update time: adaptive, Las Vegas, and $O(n^{1/2-\epsilon})$-time. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC)*. 1122–1129.

[72] Huy N Nguyen and Krzysztof Onak. 2008. Constant-time approximation algorithms via local improvements. In *Proceedings of the 49th Symposium on Foundations of Computer Science (FOCS)*. 327–336.

[73] Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. 2012. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1123–1131.

[74] Krzysztof Onak and Ronitt Rubinfeld. 2010. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*. 457–464.

[75] Michal Parnas and Dana Ron. 2007. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science (TCS)* 381, 1-3 (2007), 183–196.
[76] Ami Paz and Gregory Schwartzman. 2018. A $(2+\epsilon)$-Approximation for Maximum Weight Matching in the Semi-streaming Model. *ACM Transactions on Algorithms (TALG)* 15, 2 (2018), 18.
[77] David Peleg and Shay Solomon. 2016. Dynamic $(1+\epsilon)$-approximate matchings: a density-sensitive approach. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 712–729.
[78] Mohammad Roghani, Amin Saberi, and David Wajc. 2022. Beating the Folklore Algorithm for Dynamic Matching. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS)*. 111:1–111:23.
[79] Piotr Sankowski. 2007. Faster dynamic matchings and vertex connectivity. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 118–126.
[80] Shay Solomon. 2016. Fully dynamic maximal matching in constant update time. In *Proceedings of the 57th Symposium on Foundations of Computer Science (FOCS)*. 325–334.
[81] Vadim G Vizing. 1964. On an estimate of the chromatic class of a p-graph. *Diskret analiz* 3 (1964), 25–30.
[82] David Wajc. 2020. Rounding dynamic matchings against an adaptive adversary. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC)*. 194–207.
[83] Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. 2012. Improved constant-time approximation algorithms for maximum matchings and other optimization problems. *SIAM Journal on Computing (SICOMP)* 41, 4 (2012), 1074–1093.

APPENDIX

A PROOFS OF BASIC BUILDING BLOCKS

Here we substantiate some key propositions implied by prior work. We stress that we provide proofs mostly for completeness, due to our propositions being slight variants or being differently organized than their previous counterparts. That is, we do not claim novelty of the underlying ideas of this section.

A.1 Proof of Proposition 2.1

A key component of Proposition 2.1 is the following vertex sparsification technique for dynamic settings by Kiss [62], adapted from such a vertex sparsification of Assadi et al. [9] in the context of stochastic matching.

Proposition A.1. *There exists a randomized algorithm which for each update to G makes an update to $O\left(\frac{\log^2 n}{\epsilon^3}\right)$ contracted subgraphs, such that w.h.p. throughout any (possibly adaptively generated) update sequence, one subgraph G' has a matching of cardinality $\mu(G)\cdot(1-O(\epsilon))$ and nodeset of size $n' \leq \mu(G)/\epsilon$. Moreover, any matching M' in G' can be transformed into a matching in G of cardinality $|M'|$ in time $O(|M'|)$. For any matching M' in any G' undergoing edge updates we can maintain a matching of cardinality $|M'|$ in G with $O(1)$ worst-case update time.*

Proof. Consider a random graph G' obtained by hashing each node into one of k/ϵ buckets, for some integer k, and contracting all nodes that are hashed into the same bin. That is, two contracted nodes neighbor in G' if their corresponding bins contain neighboring nodes in G. By storing for each edge e in G' a list of edges inducing e, we can easily transform a matching M' in G' to a matching in G of the same cardinality in time $O(|M'|)$. The majority of this proof is thus dedicated to showing that $O\left(\frac{\log n}{\epsilon^2}\right)$ such contractions for each value $k = \lceil(1+\epsilon)^i\rceil$ with $i \in [\log_{1+\epsilon}(n)] \subseteq \left[O\left(\frac{\log n}{\epsilon}\right)\right]$ suffice to guarantee that one of these G' contains a matching of cardinality at least $\mu(G)\cdot(1-3\epsilon)$.

Fix an integer i and $k = \lceil(1+\epsilon)^i\rceil \leq n$. Fix a matching M in G of cardinality $|M| \leq k$. The probability that a vertex v incident on some edge of M is contracted into a separate bin than the other $2|M|-1$ endpoints can be expressed as follows:

$$(1-1/(k/\epsilon-1))^{2|M|-1} \geq \left(1-\frac{2\cdot\epsilon}{k}\right)^{2\cdot k} \geq (1-5\cdot\epsilon).$$

Thus, by linearity, the number X of such endpoints of edges of M satisfy that $\mathbb{E}[X] \geq (1-5\epsilon)\cdot 2|M|$. Observe that X is the sum of negatively associated random variables, by [42], since the hashing of vertices is equivalent to the folklore balls and bins experiment, so by standard Chernoff Bounds,

$$\Pr[X \leq 2\cdot|M|\cdot(1-6\epsilon)] \leq \exp\left(-\Theta(\epsilon^2|M|)\right).$$

If at least $2|M|\cdot(1-6\cdot\epsilon)$ endpoints of M are hashed to unique vertices then at least $|M|-2|M|\cdot 6\epsilon \geq |M|\cdot(1-12\epsilon)$ edges of M had both of their endpoints assigned to unique vertices in G' hence are present in G'.

We say that the contraction is *bad* if for *some* matching M of cardinality in the range $[k, k(1+\epsilon)+1]$ if the number of edges of M that are not present in G' is lesser than $|M|\cdot(1-12\epsilon)$. Otherwise, it is *good*. Now, there are $\sum_{i=k}^{k(1+\epsilon)+1}\binom{n}{i} \leq k\epsilon\cdot n^{k(1+\epsilon)+1} \leq n^{k(1+\epsilon)+2}$ possible matchings of size $|M|\in[k,k(1+\epsilon)]$. Therefore, by randomly contracting the graph for range $[k,k(1+\epsilon)+1]$ some $\frac{C\log n}{\epsilon^2}$ many times, for a sufficiently large C, we have that the probability that all contractions for range $[k,k(1+\epsilon)+1]$ are bad is

$$\Pr[\text{all contractions are bad}] \leq n^{k(1+\epsilon)+2}\cdot\exp\left(-\Theta(\epsilon^2 k)\cdot\frac{C\log n}{\epsilon^2}\right) \leq n^{-3}.$$

Therefore, taking union bound over the $\log_{1+\epsilon}(n)$ possible value of k, we find that with high probability, each range $[k,k(1+\epsilon)+1]$ has some good contraction.

We conclude that, w.h.p., among the $O\left(\frac{\log^2 n}{\epsilon^3}\right)$ contracted graphs, there exists a good contraction for every $k=\lceil(1+\epsilon)^i\rceil$, and in particular for $k\leq\mu(G)\leq k(1+\epsilon)+1$. That is, one of the contracted graphs contains a large matching, $\mu(G')\geq\mu(G)\cdot(1-12\epsilon)$, and has few nodes, $n'\leq k/\epsilon\leq\mu(G)/\epsilon$, as desired. □

We now proceed towards proving Proposition 2.1, restated below for ease of reference.

Proposition 2.1. *Let $\epsilon\in(0,1)$ and $\alpha\geq 1$. Suppose there exists an algorithm $\mathcal{A}$ on a dynamic n-node graph G with update time t_u, that, provided $\mu(G)\geq\epsilon\cdot n$, supports t_q-time α-approximate size estimate queries w.h.p. Then, there is another algorithm $\mathcal{A}'$ on G that always maintains an $(\alpha+O(\epsilon))$-approximate estimate ν' in $\tilde{O}_\epsilon(t_u+t_q/n)$ update time. Moreover if the update time of $\mathcal{A}$ is worst-case, so is that of $\mathcal{A}'$, and if $\mathcal{A}$ works against an adaptive adversary, then so does $\mathcal{A}'$.*

Proof. Let $\epsilon'=\alpha'\cdot\epsilon\cdot 2$ (here α' is some $O(1)$ factor). Using the algorithm described by Proposition A.1 we can generate $T=\tilde{O}_\epsilon(1)$ graphs $G_i : i\in[T]$ with the following properties: A) $\mu(G_i)\leq\mu(G)$ for all $i\in[T]$, B) There is an $i\in[T]$ satisfying that $\mu(G_i)\geq(1-\epsilon')\cdot\mu(G)$ and $\mu(G_i)\geq n\cdot\epsilon'$, C) All sub-graphs G_i undergo a single update when G undergoes an update.

Our algorithm proceeds as follows: on all T generated sub-graphs we run algorithm $\mathcal{A}$ at all times. Furthermore, on each sub-graph we maintain an $O(1)=\alpha'$-approximate estimate on the maximum matching size $\tilde{\mu}_i$ using algorithms from literature (randomized against an adaptive adversary) in $\tilde{O}_\epsilon(1)$ worst-case time. For all sub-graphs we monitor the relationship of $\tilde{\mu}_i$ and $|V_i|$. If $\tilde{\mu}_i$ increases above the threshold of $|V_i|\cdot\epsilon$ we start a run of the query algorithm on G_i returning us an α-approximate estimate of $\mu(G_i)$ which will define ν'_i. We distribute the work of this query over $|V_i|\cdot(\epsilon)^2$ updates and re-initiate the query every $|V_i|\cdot(\epsilon)^2$ updates. The matching size queries of G_i always run on the state of G_i at the start of the query (even though G_i undergoes updates during it's run). If $\tilde{\mu}_i$ decreases bellow the threshold of $|V_i|\cdot\epsilon$ we stop the querying process and set $\nu'_i=0$. Note that at initialization we just set $\nu'_i=\mu(G_i)$ for all $i\in[T]$ statically.

At all times we maintain the output $\max_{i\in R}\nu'_i$, the maximum of our matching size estimates.

Algorithm 3 Vertex set sparsification

Initialize $\nu'_i = \mu(G_i)$
Maintain contracted sub-graphs G_i and α'-approximate matching size estimates $\tilde{\mu}_i$
Run algorithm $\mathcal{A}$ on every G_i
for $i \in [T]$ **do**
  if $\tilde{\mu}_i$ becomes at least $|V_i| \cdot \epsilon$ **then**
    Initiate a matching size query of G_i in $O(t_q)$ time on the current state of G_i
    Distribute the work over the next $|V_i| \cdot \epsilon^2$ updates
    Repeatedly recompute distributed over every $|V_i| \cdot \epsilon^2$ updates
    Let ν'_i be the latest finished estimate
  if $\tilde{\mu}_i$ reduces below $|V_i| \cdot \epsilon$ **then**
    Terminate the querying process of $\mu(G_i)$
    Set $\nu'_i \leftarrow 0$
At all times return $\max_{i\in[T]} \nu'_i$

We first discuss the update time of Algorithm 3. The maintenance of the T contracted sub-graphs and matching size estimates $\tilde{\mu}_i$ takes $\tilde{O}_\epsilon(1)$ w.c. time. Running algorithm $\mathcal{A}$ on each of the contracted sub-graphs takes update time $\tilde{O}_\epsilon(t_u)$ (and is worst case if $\mathcal{A}$ has worst-case update time). A matching size query will only be initiated and run on contracted sub-graph G_i if $\tilde{\mu}_i \geq \mu(G_i)/\alpha' \geq |V_i| \cdot \epsilon$, that is if $\mu(G_i) \geq |V_i| \cdot \epsilon/2$. Each re-computation of the estimate ν'_i will be distributed over some $|V_i| \cdot \epsilon^2$ updates, that is, it will take $O(t_q/(|V_i| \cdot \epsilon^2) = O_\epsilon(t_q/|V_i|)$ worst-case time. Finding and returning the maximum matching size estimate ν'_i takes $\tilde{O}_\epsilon(1) = O(T)$ time. Therefore, the total update time of the algorithm is $\tilde{O}_\epsilon(t_u + t_q \cdot \beta/n)$ and is worst-case if $\mathcal{A}$ has worst-case update time. Furthermore, all components of the algorithm but $\mathcal{A}$ are randomized against an adaptive adversary..

It remains to argue that the algorithm maintains ν' such that $\nu' \leq \mu(G) \leq \nu' \cdot (\alpha + O(\epsilon))$ at all times. Say that G_i is a 'successful' contraction if G_i satisfies property B). By Proposition A.1, w.h.p., there is a successful contraction at all times, at time point τ_1 let that contraction be G_i. We will separate two instances:

i) Throughout the run of the algorithm at all times it held that $\tilde{\mu}_i \geq |V_i| \cdot \epsilon$: The algorithm has ran the matching size query sub-routine on G_i after every $|V_i| \cdot \epsilon^2$ edge updates. Let $G_i^{\tau_0}$ be the past state of the graph G_i when the algorithm started calculating the current estimate $(\nu_i^{\tau_1})'$. By the scheduling of this calculation we know that $\tau_0 \geq \tau_1 - \epsilon^2 \cdot |V_i|$. Hence, $\mu(G_i^{\tau_0}) \geq \mu(G_i^{\tau_1}) - \epsilon^2 \cdot |V_i|$, where $\mu(G_i^{\tau_1}) \geq \mu(G) \cdot (1 - \epsilon')$ and $\mu(G_i^{\tau_1}) \geq |V|_i \cdot \epsilon'$. Hence, $(\nu_i^{\tau_1})' \cdot \alpha \cdot (1 + O(\epsilon)) \geq \mu(G)$.

ii) At time τ_1 G_i is a successful contraction but at some prior point during the run of the algorithm $\tilde{\mu}_i$ became less than $|V_i| \cdot \epsilon$: we know that at some point τ_0 prior to τ_1 $\tilde{\mu}_i$ must have increased above $|V_i| \cdot \epsilon$. Define the state of G_i at the two time points as $G_i^{\tau_0}$ and $G_i^{\tau_1}$ respectively. As at τ_1 $G_i^{\tau_1}$ is a successful contraction we know that $\mu(G_i^{\tau_1}) \geq |V_i| \cdot \epsilon'$. When $\tilde{\mu}_i$ crossed the threshold at τ_0 it held that $\tilde{\mu}_i = |V_i| \cdot \epsilon$ that is $\mu(G_i^{\tau_0})_i \leq |V_i| \cdot \epsilon \cdot \alpha$. As per each update the maximum matching size may only change by 1 we have that $\tau_1 - \tau_0 \geq |V_i| \cdot \epsilon \cdot \alpha$. Hence, by time τ_1 the algorithm already had an updated estimate of ν'_i (that is one calculated in the previous $\epsilon^2 \cdot |V_i|$ updates such that $\mu(G_i) \geq |V_i| \cdot \epsilon$ during these updates). Here we can refer back to the previous case (pretending the algorithm initialized at τ_0). □

A.2 Proof of Proposition 2.6

We now give a proof extending standard arguments that small maximal matchings contain many length-three augmenting paths to showing that small ϵ-AMM likewise contain many such paths.

Proposition 2.6. *Let $\epsilon > 0$ and $c \in \mathbb{R}$ and let M be an ϵ-AMM in G such that $|M| \leq \left(\frac{1}{2} + c\right) \cdot \mu(G)$. Then M admits a collection of at least $\left(\frac{1}{2} - 3c - \frac{7\epsilon}{2}\right) \cdot \mu(G)$ node-disjoint 3-augmenting paths.*

Proof. The above bound for $\epsilon = 0$ is well-known (see, e.g., [64]). We reduce to this case by removing the at most $\epsilon \cdot \mu(G)$ nodes in $V \setminus V(M)$ needed to make M maximal. This yields a graph G' with $\mu(G') \geq \mu(G) \cdot (1 - \epsilon)$, and therefore

$$|M| \leq \left(\frac{1}{2} + c\right) \cdot \mu(G) \leq \frac{\frac{1}{2} + c}{1 - \epsilon} \cdot \mu(G') \leq \left(\frac{1}{2} + c + \epsilon\right) \cdot \mu(G'),$$

Consequently, by the special case of this proposition with $\epsilon = 0$, we have that the maximum number of disjoint 3-augmenting paths that M admits in G' (and hence also in G) is at least

$$\left(\frac{1}{2} - 3(c + \epsilon)\right) \cdot \mu(G') \geq \left(\frac{1}{2} - 3(c + \epsilon)\right)(1 - \epsilon) \cdot \mu(G) \geq \left(\frac{1}{2} - 3(c + \epsilon) - \frac{\epsilon}{2}\right) \cdot \mu(G),$$

as claimed. □

B PROOF OF LEMMA 4.6

In this section we prove Lemma 4.6 which we re-state for convenience.

Lemma 4.6. *Consider a graph $G' = (V', E')$ with $|V'| = n'$, and a matching M with $V(M) \subseteq V'$ that is not necessarily part of G' (i.e., we might have $M \nsubseteq E'$). For any matching M' in G', let $k_{M'}$ denote the number of edges in M both of whose endpoints are matched in M'. There is an algorithm which, given adjacency matrix query access to the edges of G', w.h.p. runs in $\tilde{O}_\epsilon(n')$ time and returns an estimate $\kappa \in [k_{M'} - \epsilon^2 n', k_{M'}]$ for some maximal matching M' in G'.*

Our proof of this lemma is a minor modification of the argument from [12, Section 5]. We claim no novelty for this proof. To make our notations consistent with the ones used by [12], we will focus on an n-node graph $G = (V, E)$ (different from our dynamic input graph). Let π be a permutation of the edges of graph $G = (V, E)$. Let $GMM(G, \pi)$ stand for the output of the greedy maximum matching algorithm when run on graph G with edge ordering π.

B.1 Building blocks

Lemma B.1 is explicitly concluded by [12], whereas Lemma B.2 is a slight modification of a construction appearing in Section 5 of [12] we need to fit our arguments.

Lemma B.1. *There is a randomized algorithm that in $\tilde{O}(|E|/|V|)$ expected time returns the matched status of a random v under $GMM(G, \pi)$, for random π. This algorithm relies on list access to the edges of G.*

In order to prove Lemma 4.6 we have to work with adjacency matrix queries. Based on a slight modification of Section 5 of [12] we can derive the following tool for this purpose.

Lemma B.2. *Let $\delta \in (0, 1/2)$. For a given n-node graph $G = (V, E)$ there exists a supergraph $H = (V_H, E_H)$ of G (i.e., $V_H \supseteq V$ and $E_H \supseteq E$) satisfying the following:*

- *$|E_H| = \Theta_\delta(n^2)$.*
- *$|V_H| = \Theta_\delta(n^2)$.*
- *At most $\delta \cdot n$ nodes of V are matched to nodes in $V_H \setminus V$ by $GMM(H, \pi)$, w.h.p. over π.*
- *$GMM(H, \pi) \cap E$ is a maximal matching in $G[V \setminus V_{slack}]$, where $V_{slack} \subseteq V$ are nodes in V that are matched to nodes in $V \setminus V_H$.*
- *Any adjacency list query to E_H (querying the i-th neighbour of a vertex according to some ordering of neighbours) can be implemented using one adjacency matrix query to E (querying the existence of any edge (u, v)).*

Informally, the main change in our construction compared to that of [12] is that our construction will allow us to argue that the random matching in the constructed graph H is, w.h.p., a maximal matching after ignoring a small set of nodes. In contrast, the construction in [12] resulted in an "expected" version of this guarantee. As the high-probability bounds will simplify our discussion later, we modify this construction below. The second change we make is in externalizing the fact that the matching computed this way is maximal, rather than 2-approximate, as stated in [12]. We now turn to proving the above lemma.

Proof of Lemma B.2. The node-set of H is $V_H := V \cup V^* \cup (W_1 \ldots, W_n) \cup (U_1, \ldots, U_n)$, where $V = \{v_1, v_2, \ldots, v_n\}$ (note that $G = (V, E_H[V])$), $V^* = \{v_2^*, v_2^*, \ldots, v_2^*\}$, $W_i = \{w_i^1, w_i^2, \ldots w_i^n\}$, and the set $U_i = \{u_i^1, u_i^2, \ldots, u_i^s\}$ is of size $s := 10n/\delta$ for all $i \in [n]$. To specify the edge-set E_H, we now define the *ordered* adjacency list for every node $v \in V_H$.

- Every node $v_i \in V$ has degree exactly n: For any $j \in [n]$, if $(v_i, v_j) \in E$ then the j^{th} neighbor of v_i is the node $v_j \in V$, otherwise it is the node $v_j^* \in V^*$.
- Every node $v_i^* \in V^*$ has degree exactly $n + s$: For any $j \in [n]$, if $(v_i, v_j) \in E$ then the j^{th} neighbor of v_i^* is the node $w_i^j \in W_i$, otherwise it is the node $v_j \in V$. Furthermore, for all $j \in [s]$, the $(n + j)^{th}$ neighbor of v_i^* is the node $u_i^j \in U_i$.
- Each node in U_j, for any $j \in [n]$, has only one neighbor (which is v_j^*).
- Node w_i^j may have degree at most one: if $(v_i, v_j) \in E$ then w_i^j is a neighbour of $v_i^* \in V^*$, otherwise it is an isolated vertex of H.

Note that $|V_H| = 2n + n^2 + ns = \Theta(n^2/\delta)$ and that similarly $|E_H| = n^2 + |E| + ns = \Theta(n^2/\delta)$. Furthermore, from the above discussion it is immediate that an adjacency list query to E_H (i.e., querying for the j-th neighbor of a vertex) can be implemented using at most one adjacency matrix query to E. It remains to prove the remaining two properties of H.

To this end, recall that V_{slack} denotes the set of vertices in V matched to V^* nodes. Then, by maximality of $GMM(H, \pi)$, we have that $GMM(H, \pi) \cap E$ is indeed maximal matching of $G[V \setminus V_{slack}]$. We now turn to bound $|V_{slack}|$. To this end, we say a node $v^* \in V^*$ is *occupied* if its earliest edge in π has its other endpoint in W_i or U_i. Trivially, such an occupied vertex $v^* \in V^*$ is matched to a vertex of $U_{v^*} \cup W_{v^*}$ under $GMM(H, \pi)$. The following simple claim, which follows by a Chernoff bound together with the simple observation that it is unlikely for a node in $V*$ to be matched in V (and thus contribute to $|V_{slack}|$).

Claim B.3. *Let π be a uniformly random permutation of E_H. Let $X_{v^*} : v^* \in V^*$ represent the indicator variable of v^* being occupied and $X_O = \sum X_{v^*}$. Then $X \geq n \cdot (1 - \delta)$ w.h.p.*

Proof. Note that each $v^* \in V^*$ has at most n edges with vertices of V' and has at least $10n/\delta$ edges with vertices in U_{v^*} and W_{v^*}. Therefore,

$$\mathbb{E}[X_{v^*}] = \Pr(X_{v^*} = 1) \geq \frac{n \cdot 10/\delta}{n \cdot (10/\delta + 1)} = 1 - \frac{1}{10/\delta + 1} \geq 1 - \delta/10.$$

On the other hand, the variables $\{X_{v^*} \mid v \in V\}$ are independent binary variables. Therefore, by Chernoff's bound, we have that

$$\begin{aligned}
\Pr(X_O \leq n \cdot (1 - \delta)) &\leq \Pr\left(X_O \leq n \cdot (1 - \delta/10) - \frac{n \cdot \delta}{2}\right) \\
&\leq \Pr\left(X_O \leq \mathbb{E}[X_O] - \mathbb{E}[X_O] \cdot \frac{\delta}{2}\right) \qquad (8) \\
&\leq 2 \cdot \exp\left(-\frac{(\delta/2)^2 \cdot \mathbb{E}[X_O]}{3}\right) \qquad (9)
\end{aligned}$$

$$\leq \quad n^{-\Theta(1)} \tag{10}$$

Inequality 8 follows from the fact that $n \geq E[X_O] \geq n \cdot (1-\delta)$. Inequality 9 is an application of Chernoff's bound. Inequality 10 follows as long as $n \cdot \delta^2 \in \Omega(\log(n))$. □

The above claim completes the proof of the last requirement of Lemma B.2. □

B.2 The algorithm

We now introduce the algorithm that will build on the previous two lemmas and inform the proof of Lemma 4.6, given in Algorithm 4. Recall that we wish to estimate the number of edges in some input matching, which here, to avoid confusion, we denote by M^*, that are both matched in some maximal matching in G.

Let $H = (V_H, E_H)$ be the supergraph of G defined by Lemma B.2 of G with $\delta = \epsilon^2/8$. For a permutation π of E_H, define $M'(\pi) := \text{GMM}(H, \pi) \cap (V \times V)$ to be the set of edges in $\text{GMM}(H, \pi)$ both of whose endpoints are in V. Let $M(\pi)$ be a maximal matching in G that is obtained by augmenting $M'(\pi)$, i.e., we start with $M = M'(\pi)$, visit the edges $e \in E$ in an arbitrarily fixed order, and obtain the matching $M(\pi)$ by greedily adding as many edges to M as possible. Note that $M'(\pi) \subseteq M(\pi) \subseteq E'$. Note also that $M(\pi)$ will be the maximal matching that Lemma 4.6 refers to as M'. We now slightly overload our notations and let $k_{M'(\pi)}$ denote the number of edges in M^* both of whose endpoints are matched in $M'(\pi)$.[7]

Algorithm 4 Extended Sub-Linear Algorithm

1: **if** $|M^*| \leq \epsilon^2 \cdot n$ **then**
2: **Return** $\kappa = 0$
3: (Implicitly) construct $H = (V_H, E_H)$ as in Lemma B.2 with $\delta = \epsilon^2/8$
4: Sample a permutation π of E_H uniformly at random
5: $L \leftarrow \frac{10^5 \cdot \log(n)}{\epsilon^5}$
6: Sample L edges $e_1, \ldots, e_L \in M^*$ uniformly at random with replacement
7: Let X_i be one if both endpoints of edge e_i are matched by $GMM(H, \pi)$ and $X = \sum_i X_i$
8: **Return** $\kappa = \frac{X \cdot |M|}{L} - \frac{n \cdot \epsilon^2}{2}$

Claim B.4. *Algorithm 4 can be implemented in time $\tilde{O}_\epsilon(n)$ in expectation.*

Proof. The construction of H is implicit, and as such takes no time. Let $T_H(v, \pi)$ stand for the time it takes to calculate the matched status of vertex $v \in V_H$ in $GMM(H, \pi)$ using the algorithm of [16]. By Lemma B.1 we have that $\mathop{\mathbb{E}}_{v \sim V_H}[T_H(v, \pi)] = \tilde{O}_\epsilon(|E_H|/|V_H|) = \tilde{O}_\epsilon(1)$. Therefore, since the endpoints of the sampled edges $S = \bigcup_{i=1}^{L} e_i \subseteq V_H$ are a subset of of vertices of cardinality $|S| \geq \epsilon^2 \cdot n$, and since $|V_H| = \Theta_\epsilon(n^2)$ we have the expected time to calculate their matched status (using adjacency matrix queries, using the construction of H) is

$$\mathop{\mathbb{E}}_{v \sim S}[T_H(v, \pi)] \leq \mathop{\mathbb{E}}_{v \sim V_H}[T_H(v, \pi)] \cdot \frac{|V_H|}{|S|} \leq \tilde{O}_\epsilon(n)$$

□

We now argue that Algorithm 4 provides a good approximation of the number of nodes in M^* both of whose endpoints are matched by $GMM(H, \pi)$. But first, we recall the basic Chernoff bounds that we will rely on here.

[7] Recall that in the statement of Lemma 4.6 we defined the notation $k_{M'}$ only if M' is a matching in G', which is not the case with $M'(\pi)$. Nevertheless, for ease of exposition, we use the notation $k_{M'(\pi)}$.

Lemma B.5. *Chernoff bound: Let X be the sum of independently distributed (or negatively associated) random variables $X_1, \ldots, X_m$ with $X_i \in [0,1]$ for each $i \in [m]$. Then for all $\delta \in (0,1)$:*

$$\Pr(|X - E[X]| \geq \delta \cdot E[X]) \leq 2 \cdot \exp\left(-\frac{\delta^2 \cdot E[X]}{3}\right).$$

Lemma B.6. *W.h.p., The output κ of Algorithm 4 satisfies*

$$\kappa_{M(\pi)} \geq \kappa \geq \kappa_{M(\pi)} - n \cdot \epsilon^2.$$

Proof. First, by Lemma B.2, we have that w.h.p., the set of nodes $V_{slack} \subseteq V$ that are matched to nodes in $V_H \setminus V$ have cardinality at most $|V_{slack}| \leq n\epsilon^2/8$. Moreover, $GMM(H, \pi) \cap E$ is a maximal matching in $G[V \setminus V_{slack}]$.

Observe that whenever $\kappa = 0$ is returned by the algorithm due to $|M^*|$ being small, the algorithm returns a trivially correct solution. As $M'(\pi)$ is an $\epsilon^2/8$-AMM, we conclude that:

$$|M'(\pi)| \leq |M(\pi)| \leq |M'(\pi)| + \frac{n \cdot \epsilon^2}{8}. \tag{11}$$

Define M^*_H to be the set of edges of M^* such that both of their endpoints are matched by $GMM(H, \pi)$. By the guarantees of the construction of H we know that there can be at most $n \cdot \epsilon^2/8$ vertices of V matched by an edge not in $M'(\pi)$. Therefore,

$$|M^*_H| \geq \kappa_{M'(\pi)} \geq |M^*_H| - \frac{n \cdot \epsilon^2}{8}. \tag{12}$$

Note that using the Algorithm 4 is sampling from edges of M^* and determining if they are in M^*_H (hence approximating $\kappa_{M^*_H}$). Specifically, by inequalities (11) and (12), we get the following.

$$\kappa_{M^*_H} \in \left[\kappa_{M'(\pi)} \pm \tfrac{n\cdot\epsilon^2}{8}\right] \subseteq \left[\kappa_{M(\pi)} \pm \tfrac{n\cdot\epsilon^2}{8} \pm |M(\pi)| - |M'(\pi)|\right] \subseteq \left[\kappa_{M(\pi)} \pm \tfrac{n\cdot\epsilon^2}{4}\right].$$

We will argue that with high probability $\frac{X\cdot|M|}{L} \in \left[\kappa_{M^*_H} \pm \frac{n\cdot\epsilon^2}{8}\right]$, dependent on the randomization of M^*_L. Observe that X_i are independently distributed random variables taking values in $[0,1]$ and X is a binomial variable with parameters $k, |\kappa_{M^*_H}|/|M|$. We will consider two cases:

Case (A): $\kappa_{M^*_H} \leq \frac{n\cdot\epsilon^3}{8}$. In this case, we derive that

$$\begin{aligned}
\Pr\left(\frac{X\cdot|M|}{L} \notin \left[\kappa_{M^*_H} \pm \frac{n\cdot\epsilon^2}{8}\right]\right) &= \Pr\left(\frac{X\cdot|M|}{L} \geq \kappa_{M^*_H} + \frac{n\cdot\epsilon^2}{8}\right) \\
&\leq \Pr\left(\frac{X\cdot|M|}{L} \geq \frac{n\cdot\epsilon^2}{8}\right) \\
&= \Pr\left(B(L, \kappa_{M^*_H}/|M|) \geq \frac{n\cdot\epsilon^2}{8}\right) \qquad (13) \\
&\leq \Pr\left(B(L,\epsilon) \geq \frac{n\cdot\epsilon^2}{8}\right) \qquad (14) \\
&\leq \Pr\left(B(L,\epsilon) \geq 2\cdot\mathbb{E}[B(L,\epsilon)]\right) \\
&\leq 2\cdot\exp\left(-\frac{\mathbb{E}[B(L,\epsilon)]}{3}\right) \qquad (15) \\
&\leq n^{-\Theta(1)}.
\end{aligned}$$

In the above derivation, (13) holds as $\kappa_{M^*_H}/|M| \leq \epsilon$ (as otherwise $\kappa = 0$ would have been returned by the algorithm), and (14) is true assuming $n \cdot \epsilon^2/8 \geq 2 \cdot L \cdot \epsilon = \tilde{O}(1)$. Finally, (15) follows from Chernoff bound (Lemma B.5) on a binomial random variable.

Case (B): $\kappa_{M^*_H} \geq n \cdot \frac{\epsilon^3}{8}$. In this case, we derive that

$$\begin{aligned}
\Pr\left(\frac{X \cdot |M|}{L} \notin \left[\kappa_{M^*_H} \pm \frac{n \cdot \epsilon^2}{8}\right]\right) &= \Pr\left(|X - E[X]| \geq \frac{n \cdot \epsilon^2}{8} \cdot \frac{L}{|M|}\right) \\
&= \Pr\left(|X - \mathbb{E}[X]| \geq \mathbb{E}[X] \cdot \frac{n \cdot \epsilon^2}{8 \cdot \kappa_{M^*_H}}\right) \\
&\leq \Pr\left(|X - \mathbb{E}[X]| \geq \mathbb{E}[X] \cdot \frac{n \cdot \epsilon^2}{8 \cdot n}\right) \\
&\leq 2 \cdot \exp\left(-\frac{\left(\epsilon^2/8\right)^2 \cdot \mathbb{E}[X]}{3}\right) \qquad (16) \\
&= \exp\left(-\frac{L \cdot \kappa_{M^*_H} \cdot \epsilon^4}{|M| \cdot 194}\right) \qquad (17) \\
&\leq n^{-\Theta(1)}.
\end{aligned}$$

In the above derivation, (16) follows from Chernoff bound (Lemma B.5), and (17) holds due to on our assumptions on $|M|$ and $\kappa_{M^*_H}$. Therefore, with high probability $X \cdot |M|/k \in [\kappa_{M^*_H} \pm \epsilon^2 \cdot n/8] \in [\kappa_{M(\pi)} \pm \epsilon^2 \cdot n \cdot (1/8 + 1/4)]$ (recall that $\kappa_{M(\pi)} = \kappa_M$). This implies that $\kappa \leq \kappa_M + \epsilon^2 \cdot n \cdot (1/8 + 1/4 - 1/2) \leq \kappa_M$ and $\kappa \geq \kappa_M - \epsilon^2 \cdot n \cdot (1/8 + 1/4 - 1/2) \geq \kappa_M - \epsilon^2 \cdot n$. □

Having concluded that Algorithm 4 can be implement in low expected time, and is correct w.h.p., we are now ready to prove Lemma 4.6, restate below for ease of reference. (Note that here $G = (V, E)$ are renamed $G' = (V', E')$, and n' and $k_{M'}$ correspond respectively to n and $\kappa_{M(\pi)}$, whereas M^* in Algorithm 4 is renamed M.)

Lemma 4.6. *Consider a graph $G' = (V', E')$ with $|V'| = n'$, and a matching M with $V(M) \subseteq V'$ that is not necessarily part of G' (i.e., we might have $M \nsubseteq E'$). For any matching M' in G', let $k_{M'}$ denote the number of edges in M both of whose endpoints are matched in M'. There is an algorithm which, given adjacency matrix query access to the edges of G', w.h.p. runs in $\tilde{O}_\epsilon(n')$ time and returns an estimate $\kappa \in [k_{M'} - \epsilon^2 n', k_{M'}]$ for some maximal matching M' in G'.*

Proof. By Claim B.4, Algorithm 4 runs in expected $\tilde{O}_\epsilon(n)$ time and returns a correct solution with high probability. To improve its running time guarantee to a high probability bound we only need to incur a blowup of $O(\log(n))$ in running time: run the algorithm $O(\log(n))$ time in parallel and output the solution given by the first terminating copy. One of these algorithms will terminate within at most twice the expected time, by Markov's inequality, and so w.h.p., one of these completes after $\tilde{O}_\epsilon(n)$ time. Finally, by union bound and Lemma B.6, all of the $\log n$ algorithms' output satisfies the desired bounds with probability $1 - 1/poly(n)$, and so w.h.p., we obtain a solution satisfying the desired bounds after $\tilde{O}_\epsilon(n)$ time. □

C OMITTED PROOFS FROM SECTION 4

Here we prove the tighter bound on the number of $V(M_1)$-disjoint 3-augmenting paths in the subgraph $M_1 \cup M_2$ as output by Algorithm 2, restated below.

Lemma 4.1. *If $|M_1| = \left(\frac{1}{2}+c\right)\cdot\mu(G)$, then $G[M_1 \cup M_2]$ contains a set $\mathcal{P}$ of 3-augmenting paths w.r.t. M_1 that are disjoint in their $V(M_1)$ nodes, with expected cardinality at least*

$$\mathbb{E}[|\mathcal{P}|] \geq \left(\frac{b}{b+1}\right)\cdot\left(\frac{1}{4}\cdot\left(\frac{1}{2}-3c\right)-\frac{1}{b}\cdot\left(\frac{1}{2}+c\right)-\frac{7\epsilon}{8}\right)\cdot\mu(G).$$

Proof. Fix a maximum set of disjoint length-three augmenting paths w.r.t. M_1 in G, denoted by $\mathcal{P}^*$. By Proposition 2.6, we have $|\mathcal{P}^*| \geq \left(\frac{1}{2}-3c-\frac{7\epsilon}{2}\right)\cdot\mu(G)$. Next, let $S \subseteq \mathcal{P}^*$ be the paths $u'-u-v-v'$ that "survive" the bipartition, in the sense that $(u,u'),(v,v') \in E_2$. By construction, each path in $\mathcal{P}^*$ survives with probability exactly $\frac{1}{4}$. Therefore, $\mathbb{E}[|S|] \geq \frac{1}{4}\cdot\left(\frac{1}{2}-3c-\frac{7\epsilon}{2}\right)\cdot\mu(G)$. Let $D := \mathcal{P}^* \setminus S$ be the set of paths that did not survive this bipartition.

For $i \in \{0,1,2\}$, let $S_i \subseteq S$ and $D_i \subseteq D$ be the sets of paths $u'-u-v-v'$ in S and D (respectively) with i of their $V(M_1)$ nodes u and v matched in M_2. Now, by our bipartition, if $u'-u-v-v' \in S_2 \cup D_2$, i.e., if u and v are both matched in M_2, then they are matched to distinct nodes. Therefore, $G[M_1 \cup M_2]$ contains a set of augmenting paths $\mathcal{P}$ w.r.t. M_1 that are disjoint in their $V(M_1)$ nodes, of cardinality $|\mathcal{P}| = |S_2| + |D_2|$. We now turn to lower bounding $|S_2| + |D_2|$.

To bound $|S_2| + |D_2|$, we will double count the edges of M_2, once from their $V(M_1)$ endpoints, and once from their $\overline{V(M_1)}$ endpoints. First, by definition, since each edge in M_2 has exactly one endpoint in $V(M_1)$ and each node in $V(M_1)$ is matched at most once in M_2, we have that $|M_2| = 2|S_2| + |S_1| + 2|D_2| + |D_1| \leq |S_2| + |D_2| + |M_1|$, where the inequality follows from $|M_1| \geq \sum_{i=0}^{2}(|S_i| + |D_i|)$, by definition. On the other hand, for each of the $|S| - |S_2| = |S_0| + |S_1|$ survived paths $u'-u-v-v' \in S_0 \cup S_1$ that does not have both its internal nodes matched in M_2, we have by maximality of M_2 that u' and/or v' must contribute b distinct edges to M_2. Therefore, $|M_2| \geq b\cdot(|S_0| + |S_1|)$. Combining the above, we obtain

$$b\cdot(|S| - |S_2|) \leq |M_2| \leq |S_2| + |D_2| + |M_1|,$$

which after rearranging, yields

$$b\cdot|S| - |M_1| \leq (b+1)\cdot|S_2| + |D_2| \leq (b+1)\cdot(|S_2| + |D_2|).$$

Simplifying and combining with the lower bound on $\mathbb{E}[|S|]$, we obtain the claimed bound, as follows.

$$\begin{aligned}\mathbb{E}[|\mathcal{P}|] = \mathbb{E}[|S_2| + |D_2|] &\geq \frac{b}{b+1}\cdot\left(\mathbb{E}[|S|] - \frac{1}{b}\cdot|M_1|\right)\\ &\geq \frac{b}{b+1}\cdot\left(\frac{1}{4}\cdot\left(\frac{1}{2}-3c-\frac{7\epsilon}{2}\right)-\frac{1}{b}\cdot\left(\frac{1}{2}+c\right)\right)\cdot\mu(G).\\ &= \frac{b}{b+1}\cdot\left(\frac{1}{4}\cdot\left(\frac{1}{2}-3c\right)-\frac{1}{b}\cdot\left(\frac{1}{2}+c\right)-\frac{7\epsilon}{8}\right)\cdot\mu(G).\end{aligned}$$

□

D OMITTED PROOFS OF SECTION 5

We stress that the following is essentially implied by the work of [82], from which we now repeat significant amount of text essentially verbatim. The only difference here will be our final proof of Lemma 5.7, allowing us to efficiently periodically compute an ϵ-AMM, and the use of this lemma in the subsequent section. Readers familiar with [82] are encourage to read ahead to that lemma.

Overview. Briefly, [82] identified an edge-coloring-based approach to compute, based on the efficient maintenance of edge colorings and a particular fractional matching of [25], a kernel. (See Algorithm 5.) We start by recalling the type of fractional matching needed here, due to [4].

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

As proven in [4, Appendix A], the dynamic fractional matching of [27] is precisely such an approximately-maximal matching.

Lemma D.2. *For all $\epsilon \leq \frac{1}{2}$, there is a deterministic dynamic $(1+2\epsilon, \max\{54 \log n/\epsilon^3, (3/\epsilon)^{21}\})$-AMfM algorithm with $t_u = O(\log^3 n/\epsilon^7)$ worst-case update time, changing at most $O(\log n/\epsilon^2)$ edges' fractions per update in the worst case.*

Now, we turn to the sparsification procedure of [82], given in Algorithm 5. Briefly, this algorithm decomposes the graph into a logarithmic number of subgraphs, based on grouped x-values, edge colors these subgraphs using at most $\gamma = 2$ times their maximum degree, and then outputs the union of these subgraphs.

Algorithm 5 Edge-Color and Sparsify [82]

1: **for** $i \in \{1, 2, \ldots, \lceil 2 \log_{1+\epsilon}(n/\epsilon) \rceil\}$ **do**
2: let $E_i \triangleq \{e \mid x_e \in ((1+\epsilon)^{-i}, (1+\epsilon)^{-i+1}]\}$.
3: compute a $2\lceil (1+\epsilon)^i \rceil$-edge-coloring χ_i of $G_i \triangleq G[E_i]$. ▷ Note: $\Delta(G_i) < (1+\epsilon)^i$
4: Let S_i be a sample of $\min\{2\lceil d(1+\epsilon) \rceil, 2\lceil (1+\epsilon)^i \rceil\}$ colors without replacement in χ_i.
5: **Return** $K \triangleq (V, \bigcup_i \bigcup_{M \in S_i} M)$.

The following lemma of [82] allows us to compute kernels from AMfMs using Algorithm 5.

Lemma D.3. *Let $c \geq 1$, $\epsilon > 0$ and $d \geq \frac{9c(1+\epsilon)^2 \cdot \log n}{\epsilon^2}$. If $\vec{x}$ is a (c,d)-AMfM, then the subgraph K output by Algorithm 5 when run on $\vec{x}$ with ϵ and d is a $(c(1+O(\epsilon)), d(1+O(\epsilon)), 0)$-kernel, w.h.p.*

We are now ready to prove our (periodic) algorithmic kernel and AMM algorithm's guarantees, restated below for ease of reference.

Lemma 5.7. *Let $\epsilon \in (0,1)$ and $d = \tilde{O}_\epsilon(1)$ be sufficiently large. Then, there exists a robust algorithm with worst-case update time $t_u = \tilde{O}_\epsilon(1)$ allowing for (ϵ, d)-kernel and ϵ-AMM queries in worst-case query time $t_q = \tilde{O}_\epsilon(d \cdot \mu(G))$. The query's outputs are a kernel and an ϵ-AMM w.h.p.*

Proof. We maintain the dynamic $(1+2\epsilon, \tilde{O}_\epsilon(1))$-AMfM of Lemma D.2, using $\tilde{O}_\epsilon(1)$ deterministic w.c. update time and number of changes to edges per update. In addition, we maintain the subgraphs G_i in Algorithm 5. In each such subgraph we maintain $2\lceil (1+\epsilon)^i \rceil$-color edge colorings in each G_i in $O(\log n)$ deterministic w.c. time per change to $\vec{x}$, using the logarithmic-time $(2\Delta - 1)$-edge coloring algorithm of [24]. This concludes the description of the updates, which by the above take deterministic w.c. update time $t_u = \tilde{O}_\epsilon(1)$.

Next, to compute a kernel, we run the sampling step of Algorithm 5. As this is bottlenecked by the time to write down the $O(\log^2 n)$ colors (matchings), each of size no greater than $\mu(G)$ (by definition), this query takes deterministic $\tilde{O}(\mu(G))$. Finally, this output graph K is an $(O(\epsilon), d(1+O(\epsilon)))$-kernel w.h.p., by Lemma D.3. Finally, to output an ϵ-AMM, we appeal to the static algorithm Lemma 5.6, which runs in deterministic time $\tilde{O}_\epsilon(d \cdot \mu(G)) = \tilde{O}(\mu(G))$ and outputs an ϵ-AMM, provided K is a kernel, i.e., it also succeeds w.h.p. □