Beating the Folklore Algorithm for Dynamic Matching

Mohammad Roghani
Stanford University, CA, USA
roghani@stanford.edu

Amin Saberi
Stanford University, CA, USA
saberi@stanford.edu

David Wajc
Stanford University, CA, USA
wajc@stanford.edu

Abstract

The maximum matching problem in dynamic graphs subject to edge updates (insertions and deletions) has received much attention over the last few years; a multitude of approximation/time tradeoffs were obtained, improving upon the folklore algorithm, which maintains a maximal (and hence 2-approximate) matching in $O(n)$ worst-case update time in n-node graphs.

We present the first deterministic algorithm which outperforms the folklore algorithm in terms of *both* approximation ratio and worst-case update time. Specifically, we give a $(2-\Omega(1))$-approximate algorithm with $O(m^{3/8}) = O(n^{3/4})$ worst-case update time in n-node, m-edge graphs. For sufficiently small constant $\epsilon > 0$, no deterministic $(2+\epsilon)$-approximate algorithm with worst-case update time $O(n^{0.99})$ was known. Our second result is the first deterministic $(2+\epsilon)$-approximate *weighted* matching algorithm with $O_\epsilon(1) \cdot O(\sqrt[4]{m}) = O_\epsilon(1) \cdot O(\sqrt{n})$ worst-case update time. Neither of our results were previously known to be achievable by a randomized algorithm against an adaptive adversary.

Our main technical contributions are threefold: first, we characterize the tight cases for *kernels*, which are the well-studied matching sparsifiers underlying much of the $(2+\epsilon)$-approximate dynamic matching literature. This characterization, together with multiple ideas – old and new – underlies our result for breaking the approximation barrier of 2. Our second technical contribution is the first example of a dynamic matching algorithm whose running time is improved due to improving the *recourse* of other dynamic matching algorithms. Finally, we show how to use dynamic bipartite matching algorithms as black-box subroutines for dynamic matching in general graphs without incurring the natural $\frac{3}{2}$ factor in the approximation ratio which such approaches naturally incur (reminiscent of the integrality gap of the fractional matching polytope in general graphs).

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases dynamic matching, dynamic graph algorithms, sublinear algorithms

Digital Object Identifier 10.4230/LIPIcs.ITCS.2022.111

Related Version *Full Version*: https://arxiv.org/abs/2106.10321

Funding This research is partially supported by NSF award 1812919, ONR award N000141912550, and a gift from Cisco Research.

Acknowledgements The authors thank Sayan Bhattacharya and Aaron Bernstein for helpful discussions, and Aaron Bernstein for sharing a preprint of [5].

1 Introduction

We study the dynamic (weighted) matching problem, where our goal is to maintain an approximately maximum (weight) matching subject to edge updates (insertions and deletions) in a dynamically evolving graph.

For approximation and update time, there are two natural barriers for the dynamic matching problem: an approximation ratio of 2, and an update time of $O(n)$. Both bounds are achieved by a folklore deterministic algorithm maintaining a maximal matching in worst-case update time $O(n)$ (see Figure 1). Moreover, both bounds stand at the frontier of known time/approximation tradeoffs for randomized and deterministic dynamic matching algorithms. For example, an approximation ratio of 2 is the best approximation ratio known to be achievable in polylogarithmic worst-case update time [6] or constant amortized time [33] for randomized algorithms.[1] Similarly, an approximation ratio of $2+\epsilon$ is the best approximation ratio known to be achievable in polylogarithmic amortized update time deterministically [13, 12] or worst-case polylogarithmic update time without the oblivious adversary assumption [35].[2] On the other hand, $O(\sqrt{m}) = O(n)$ is the current best worst-case update time for deterministic algorithms achieving better than 2 approximation [21, 31, 29]. Moreover, an update time of $\Omega(n)$ (ignoring sub-polynomial factors) is known to be necessary for (exact) dynamic matching, assuming any one of several widely believed conjectures, including the Strong Exponential Time Hypothesis, and the 3SUM, APSP, and OMv conjectures [16, 23, 26].

```
INIT: $M \leftarrow \emptyset$.
INSERT($e$): If both $u, v \in e$ are unmatched in $M$, then $M \leftarrow M \cup \{e\}$.
DELETE($e$): If $e \in M$, for each $v \in e$, if $v$ has an unmatched neighbor $w$, then
$M \leftarrow M \cup \{(v, w)\}$.
```

Figure 1 The Folklore Algorithm.

Given the above, breaking the approximation ratio of 2 and worst-case update time of $O(n)$ barriers stand as two recurring goals of the rich literature on the dynamic matching problem. A concentrated effort, starting with the influential work of Onak and Rubinfeld [30], has resulted in numerous algorithms breaking either one of these barriers individually [2, 30, 33, 14, 1, 6, 12, 11, 13, 21, 29, 7, 8, 31, 35, 3, 9]. However, despite this long line of work, both barriers were not known to be surpassable simultaneously without assuming sparsity or bipartiteness [21, 7, 29, 31, 35], settling for amortized update time [8], or using randomness and the oblivious adversary assumption [4]. Whether or not there exists a deterministic algorithm that beats the trivial folklore $O(n)$-time maximal matching algorithm both in terms of worst-case update time and approximation ratio, thus simultaneously breaking these two natural barriers for this problem in its full generality, remained a vexing open problem.

[1] A dynamic algorithm has amortized update time $f(n)$ if for any sequence of t updates starting with the empty graph, the algorithm takes $t \cdot f(n)$ time.

[2] The oblivious adversary assumption stipulates that the update sequence is generated non-adaptively, and in particular each update is independent of the algorithm's previous random coin tosses. This assumption, needed for the analysis of many randomized dynamic algorithms, rules out their black-box use in many applications. See [28, 35] for discussions.

1.1 Results

We resolve the above open problem, and present the first deterministic algorithm which outperforms the folklore dynamic matching algorithm, in terms of both approximation and worst-case update time. Our main result is the following.

▶ **Theorem 1.** *(Beating the Folkore Algorithm) There exists a deterministic* 1.999999*-approximate dynamic matching algorithm with worst-case update time $O(m^{3/8})$.*

Prior to this work, no deterministic algorithm was known to achieve $2+\epsilon$ approximation in $O_\epsilon(n^{0.99})$ worst-case update time, where $O_\epsilon(\cdot)$ suppresses dependencies on ϵ. As a byproduct of the algorithm given for Theorem 1, we obtain a secondary result, rectifying this state of affairs for both the matching and weighted matching problem and present the first $(2+\epsilon)$-approximate matching algorithm with (polynomially) sub-linear update time for any constant $\epsilon > 0$.

▶ **Theorem 1.1** (Informal for Weighted)**.** *There exist deterministic $(2+\epsilon)$-approximate dynamic matching and weighted matching algorithms with worst-case update time $O_\epsilon(1) \cdot O(\sqrt[4]{m}) = O_\epsilon(1) \cdot O(\sqrt{n})$.*

For constant $\epsilon > 0$, the update time of the algorithms of Theorem 1.1 is $O(\sqrt[4]{m})$. Such worst-case update time was only previously known to yield a $(9/4+\epsilon)$ approximation (see [5, Section 7] and the full version of the paper for details), or worse [35].

We contrast our results with prior linear- and sublinear-time deterministic algorithms in Table 1.

Table 1 Known $O(n)$ Time Deterministic Dynamic Matching Algorithms in General Graphs (References are to the latest publication, with the first publication venue in parentheses).

Approx.	Update Time	Worst Case	Notes	Reference
$4+\epsilon$	$O(\sqrt[3]{m}/\epsilon^2)$	✓		Bhattacharya et al. (SODA '15) [11]
$3+\epsilon$	$O(\sqrt{n}/\epsilon)$	✗		
$9/4+\epsilon$	$O(\sqrt[4]{m}) \cdot \text{poly}(1/\epsilon)$	✓		Bernstein et al. (STOC '21) [5]
$(2+\epsilon)\cdot c$	$\tilde{O}(n^{1/c}) \cdot \text{poly}(1/\epsilon)$	✓	$\forall c \geq 1$	Wajc (STOC '20) [35]
$2+\epsilon$	$O(\sqrt[4]{m}/\sqrt{\epsilon})$	✓		**This Work**
$2+\epsilon$	$\text{poly}(\log n, 1/\epsilon)$	✗		Bhattacharya et al. (STOC '16) [12] Bhattacharya and Kiss (ICALP'21) [13]
2	$O(n)$	✓		Folklore
1.999999	$O(\sqrt{n}\cdot\sqrt[8]{m})$	✓		**This Work**
$3/2+\epsilon$	$O(\sqrt[4]{m}/\epsilon^{2.5})$	✗		Bernstein and Stein (SODA '16) [8]
$3/2$	$O(\sqrt{m})$	✓		Neiman and Solomon (STOC '13) [29]
$1+\epsilon$	$O(\sqrt{m}/\epsilon^2)$	✓		Gupta and Peng (FOCS '13) [21]

1.2 Our Approach in a Nutshell

Warm-up: Faster $(2+\epsilon)$-Approximate Matching. Our starting point is the highly successful matching sparsifiers of Bhattacharya, Henzinger, and Italiano [11] – termed *kernels* – previously used in numerous dynamic matching algorithms [11, 1, 35, 12, 5]. Kernels are $(2+\epsilon)$-approximate matching sparsifiers (i.e., these subgraphs contain a $(2+\epsilon)$-approximate matching), of small maximum degree, $d = o(n)$. To obtain approximation ratios close to $(2+\epsilon)$, it is therefore natural to combine kernel-maintenance algorithms with known near-maximum (i.e., $(1+\epsilon)$-approximate) algorithms with worst-case update time linear in the maximum degree [21, 31]. This seems to suggest an $O(T+d)$ time $(2+O(\epsilon))$-approximate algorithm, where T is the update time for maintaining the kernel.

Unfortunately, combining these two ideas does not immediately result in an $O(T+d)$ update time. This is because near-maximum matching algorithms take $O(d)$ *time per update to the kernel*, and not per update to the input dynamic graph G. Consequently, if c is the number of changes to the kernel per update to G, then combining these algorithms yields an update time of $O(T + c \cdot d)$. Prior deterministic kernel maintenance algorithms [11] all had $c \cdot d = \Omega(n)$ in the worst case. Consequently, no deterministic $(2+\epsilon)$-approximate algorithm with worst-case $o(n)$ time was previously known.

In Section 4, we show how to de-amortize an $O_\epsilon(n/d)$ update time kernel maintenance algorithm of Bhattacharya et al. [11] (and speed it up by an $O(n/\sqrt{m})$ factor in the full version of the paper), while making only $c = O(1)$ updates to the kernel K per update to G. Setting the kernel's (tunable) maximum degree to $d = \sqrt[4]{m}$ yields our sublinear $(2+\epsilon)$-approximate unweighted algorithm of Theorem 1.1. Our weighted algorithm of the same theorem (see the full version) uses our kernel algorithm and the recent dynamic weighted matching framework of Bernstein et al. [5].

Breaking the Barrier of Two. If the kernel K which we maintain *happens to be* a $(2-\epsilon)$-approximate matching sparsifier, then a near-maximum matching in K would be a $(2-\Omega(\epsilon))$-approximate matching in G, maintainable in worst-case $o(n)$ update time. Unfortunately, the $(2+\epsilon)$ approximation ratio can be tight for kernels. Nonetheless, we show that it is possible to use kernels to obtain better-than-two-approximate matchings dynamically.

Our high-level approach is a natural one: we find a bounded-degree subgraph A, whose union with the kernel contains, say, $2\epsilon \cdot \mu(G)$ disjoint augmenting paths with respect to some maximum matching in the kernel K. Since the kernel K is $(2+\epsilon)$-approximate, the augmented kernel $AK := A \cup K$ is therefore $(2-\epsilon)$-approximate, whether or not the kernel is $(2-\epsilon)$-approximate. Therefore, we can maintain in $o(n)$ worst-case update time, a $(1+O(\epsilon))$-approximate matching in this low-degree subgraph, AK, giving a $(2-O(\epsilon))$-approximate matching in G.

The above approach seems simple, but its dynamic implementation presents two challenges. The first challenge is to identify a *dynamically maintainable* sparse subgraph A whose edges "augment" the kernel. The second challenge is to guarantee that following each update to G, the worst-case number of changes to the augmented kernel $AK = A \cup K$ is small. This is crucial, since the time to update the output matching, i.e., a near-maximum matching in AK, is proportional to the product of the maximum degree of AK and the number of edge updates to AK *per update to G.*

Identifying Good Augmentations. Our first key observation is a structural characterization of kernels which are worse than $(2-\epsilon)$-approximate. First, we show that such a kernel K contains a maximum matching M such that nearly all connected components of $M \cup M^*$

(where M^* is any maximum matching in G) are augmenting paths of length three w.r.t. M. Moreover, in the vast majority of these augmenting paths, the edges of M^* connect a high- and a low-degree node in K. This motivates us to compute a large matching M' in the *bipartite* subgraph between high- and low-degree nodes in K, in the hope that such an M' contains edges of $G \setminus K$ whose addition to the kernel K increases its maximum matching size. (Foreshadowing our use of the dynamic bipartite matching algorithm of Bernstein and Stein [7], we focus on $(3/2+\epsilon)$-approximate matchings in this subgraph.) While a single such matching M' is insufficient, we show in Section 3.3 that the union of two $(3/2+\epsilon)$-approximate matchings in two similarly-defined bipartite subgraphs do constitute a good bounded-degree augmentation A of the kernel. That is, the augmented kernel, $AK = A \cup K$, contains a $(2-\epsilon)$-approximate matching. This allows us to use kernels and $(3/2+\epsilon)$-approximate bipartite matching algorithms to obtain $(2-\epsilon)$-approximate matchings in general graphs in a *static* setting.

Dynamizing Our Approach. An intricate part of this work is in dynamizing our outlined approach. For this, we must (i) make few changes to the bipartite subgraphs derived from K (as each such change will cost the update time of the bipartite matching algorithm), and (ii) make few changes to the matchings computed in these subgraphs (as each such change will cost the update time of the bounded-degree algorithm in K). However, the bipartite subgraphs discussed in our static approach above are defined by degrees in the kernel K, which may change abruptly, resulting in *vertex updates* (vertex insertions and deletions) in these bipartite subgraphs. This is problematic, as no $(2-\Omega(1))$-approximate algorithm is known with $o(n)$ vertex update time, even in bipartite graphs.

Approximate Degrees, and Star Updates. To overcome the above bottleneck, we extend our static approach, and prove that a sparse augmentation A of the kernel K is also obtained from the union of large matchings in some $O(1)$ bipartite subgraphs, $B_1, B_2, \dots, B_{O(1)}$, defined based on *approximate degrees* in K (see Section 3.4 for precise definition). The key advantage of basing these subgraphs on approximate degrees is that (i) we can maintain such approximate degrees, and hence these subgraphs, in $o(n)$ update time (see the full version), and (ii) these bipartite subgraphs $\{B_i\}_i$ change less abruptly than their exact-degree-based counterparts. In particular, each change in G (and hence in K) results in *local* changes to the approximate degrees. Specifically, each update to G causes the bipartite subgraphs $\{B_i\}_i$ to change by *star updates* – addition/removal edges of a star graph with $b = o(n)$ edges.

These star updates are doubly beneficial: first, they allow us to maintain, using the bipartite algorithm of Bernstein and Stein [7], a $(3/2+\epsilon)$-approximate matching M_i in each B_i in worst-case update time $b \cdot O_\epsilon(\sqrt[4]{m})$, which for our values of b is sublinear in n. Second, since a star update only increases or decreases the size of any matching by at most one, we can appeal to (a slight extension of) the recent framework of Solomon and Solomon [32] for decreasing *recourse* (number of changes to the output per update) to guarantee that each matching M_i only changes by $O(1/\epsilon) = O(1)$ edges per star update, and hence per update to G. We then take the union of these dynamic matchings $\{M_i\}_i$ to be our dynamic augmentation, $A = \bigcup_i M_i$, which changes by $O(1)$ edges per update to G. Since the kernel likewise changes by $O(1)$ edges per update to G, we have that the augmented kernel $AK = A \cup K$, which has maximum degree $d + O(1)$, has $c = O(1)$ edge changes per update to G. This then allows us to maintain a near-maximum matching in AK in an additional $O(c \cdot d) = O(d) = o(n)$ worst-case time per update in G. By our structural results of Section 3, this last matching is a $(2-\Omega(\epsilon))$-approximate matching in G, yielding our main result, Theorem 1.

1.2.1 Conceptual Contributions

As the above overview suggests, our main result requires a careful combination of a number of ideas. Here we outline the key novel ideas.

Our first new contribution is a structural characterization of "tight" kernels, i.e., kernels whose approximation is no better than roughly two approximate. Given the use of kernels in the dynamic matching literature [11, 1, 35, 5], it seems plausible that this characterization will prove useful in subsequent developments in the area.

Our second contribution is in exhibiting the power of dynamic matching algorithms for restricted class families – bipartite graphs – to improve algorithms in general graphs. Prior work has relied on fast algorithms for bounded-degree [21] or bounded-arboricity graphs [31] to improve their running time [7, 8, 1, 35, 13]. We present the first use of dynamic matching algorithms for *bipartite* graphs to improve dynamic matching algorithms in general graphs (beyond the trivial, yet overlooked reduction which incurs a loss factor of $\frac{3}{2}$ in the approximation ratio (see details in the full version). This adds to the list of tools for dynamic matching algorithms.

Finally, our work reinforces the message of prior work, that the study of dynamic algorithms with bounded recourse (number of changes to the output) may be of interest beyond its fundamental nature: reducing recourse may prove useful in reducing algorithms' *update time* (as we show – even for the same problem!). This suggests that some slow but recourse-bounded algorithms, may prove useful in designing fast algorithms for problems for which no such algorithms are known (e.g., [20, 10]).

1.3 Related Work

We briefly review the most relevant algorithmic results for the dynamic matching problem. For a more detailed survey of the rich literature on dynamic matching and dynamic algorithms more broadly, we refer to the recent survey of Hanauer et al. [22].

Breaking the Approximation Barrier of Two. For many computational models with dynamic inputs, such as streaming and online algorithms, an approximation ratio of two is easy to achieve for the maximum matching problem, and beating this bound is either a major open problem, or is provably impossible [18]. In the dynamic graph setting, Neiman and Solomon [29] were the first to break the natural approximation ratio of two, giving an $O(\sqrt{m}) = O(n)$ worst-case time algorithm with approximation ratio of $\frac{3}{2}$, later improved to $(1+\epsilon)$ [21]. Bernstein and Stein [7, 8] then showed how to maintain a $\frac{3}{2}+\epsilon$ approximation in $O(\sqrt[4]{m}/\text{poly}(\epsilon)) = O(\sqrt{n}/\text{poly}(\epsilon))$ time – worst-case for bipartite graphs, and amortized for general graphs. In a recent work, Behnezhad et al. [4] showed that for any $\epsilon > 0$, there exists a *randomized* algorithm with worst-case $O(\Delta^\epsilon) = O(n^\epsilon)$ update time, and approximation ratio of $2 - \frac{1}{1000 \cdot 2^{13/\epsilon}}$, also relying on the search for short augmenting paths. For this, their algorithm crucially relies on the oblivious adversary assumption. Whether the barrier of 2 can be broken in $o(n)$ worst-case time by an algorithm (whether deterministic or randomized) which does not require this assumption remained a tantalizing open question, which we resolve in the affirmative.

Breaking the $O(n)$ Worst-Case Time Barrier. As mentioned above, $\Omega(n)$ update time is likely impossible for exact dynamic matching algorithms [16, 23, 26]. On the other hand, the dynamic matching literature abounds with (approximate) algorithms breaking the $O(n)$ time barrier [35, 1, 14, 6, 13, 11, 12, 33, 30]. For example, Wajc [35] presented a family of deterministic algorithms trading off worst-case update time and approximation, requiring $\tilde{O}(n^{1/c} \cdot \text{poly}(1/\epsilon))$ update time for $(2+\epsilon)\cdot c$ approximation, for any $c \geq 1$ and $\epsilon > 0$. This, however, does not result in $o(n)$ update time for all $2+\epsilon$ approximation, let alone for $2-\Omega(1)$ approximation. The only sub-linear time sub-2-approximate algorithms known in general graphs are the aforementioned *randomized* algorithm of Behnezhad et al. [4] and the deterministic *amortized* time algorithm of Bernstein and Stein [8]. Indeed, for 9/4-approximate matching (or better), no deterministic algorithm for general graphs with $o(n)$ worst-case update time was previously known. On the other hand, the extent of the usefulness of randomization and amortization are key questions in the dynamic algorithms literature. Fittingly, much effort has been spent on de-randomizing and de-amortizing dynamic matching results (see, e.g., [13, 6]). In this work, we show that neither randomization nor amortization are needed to achieve sub-linear-time sub-two-approximate dynamic matching algorithms.

Subsequent work. Following the posting of this work, two papers obtaining improved $3/2+\epsilon$ approximation in deterministic $o(n)$ worst-case update time (respectively, in $O_\epsilon(\sqrt[4]{m})$ and $O_\epsilon(\sqrt{n})$) were posted [19, 24]. Those later papers' techniques are orthogonal to ours.

2 Preliminaries

Problem statement. Our input is a dynamically changing graph $G = (V, E)$, initially empty, undergoing edge *updates* (insertions and deletions). We denote by G_t this graph after t updates. The objective is to maintain a matching, i.e., a node-disjoint set of edges, of size close to the maximum matching size, $\mu(G)$, spending little time following each update. We denote a maximum (weight) matching in G by M^*. For a weighted graph $G = (V, E, w)$, we wish to compute a matching whose weight approximate the maximum weight matching in G, denoted by $MWM(G) := w(M^*) := \sum_{e\in M^*} w(e)$.

Matching Theory Basics. A matching M is (inclusion-wise) maximal in $G = (V, E)$ if G contains no matching $M' \supsetneq M$. For a matching M in $G = (V, E)$, an alternating path P in G is a path whose edges alternate between M and $E \setminus M$. An augmenting path is an alternating path whose endpoints are unmatched in M. For a set of k disjoint augmenting paths $P_1, P_2, \ldots, P_k$ with respect to M, the symmetric difference between M and $P := \bigcup_i P_i$, denoted by $M \bigoplus P$, is a matching of cardinality $|M| + k$, matching all nodes matched by M (and others). The symmetric difference of M and M^* consists of even length paths and cycles, as well as $|M^*| - |M|$ odd-length augmenting paths. We say an edge $e \in M$ is *3-augmentable* if the connected component of $M \bigoplus M^*$ containing e is an augmenting path of length three. A standard result in matching theory (see, e.g., [25, Lemma 1]) is that a maximal matching which is not much better than 2-approximate must consist mostly of 3-augmentable edges.

▶ **Proposition 2.1.** *Let $\epsilon \geq 0$. Let M be a maximal matching of G s.t. $|M| \leq \left(\frac{1}{2}+\epsilon\right)\cdot \mu(G)$. Then M contains at least $\left(\frac{1}{2}-3\epsilon\right)\cdot\mu(G)$ edges which are 3-augmentable.*

2.1 Dynamic Matching Background

In addition to our new structural results and algorithms, our work will rely on a number of previous algorithms from the dynamic matching literature, which we now outline.

Bounded-Degree Algorithms. One useful property for approximate dynamic matching is the "stability" of the matching problem, first used in a dynamic setting by Gupta and Peng [21], which implies that for an α-approximate matching M in G_t, the undeleted edges of

M by time $t' \in [t+1, t+\epsilon \cdot \mu(G_t)]$ constitute an $\alpha(1+\epsilon)$-approximate matching in $G_{t'}$. Combined with static linear-time near-maximum matching algorithms [27, 17], this yields near-maximum algorithms with worst-case update time linear in the maximum degree. This is implied by the work of Gupta and Peng [21] (first observed in Bernstein and Stein [7]) and by the arboricity-time algorithm of Peleg and Solomon [31]. For completeness, we provide a short proof in the full version of the paper.

▶ **Proposition 2.2** (Bounded-Degree Algorithm). *For any $\epsilon \leq 1/3$, there exists a deterministic $(1+\epsilon)$-approximate matching algorithm with worst-case update time $O(\Delta/\epsilon^2)$ in dynamic graphs with maximum degree at most Δ.*

Kernels. Proposition 2.2 motivates the study of dynamic *matching sparsifiers* – sparse (specifically, low-degree) subgraphs which approximately preserve the maximum matching in the dynamic graph G. This is the approach followed by numerous works in this area [1, 11, 7, 8, 5, 21, 35, 13]. One family of sparsifiers which have proven useful in a number of these works are *kernels*, introduced by [11].

▶ **Definition 2.3** (Kernels [11]). *An (ϵ, d)-*kernel *of a graph $G = (V, E)$ is an edge-induced subgraph $K = (V, E_K)$ satisfying the following properties:*
(P₁) $\max_{v \in V} d_H(v) \leq d$.
(P₂) $\max_{v \in e} d_H(v) \geq d(1-\epsilon)$ *for each edge $e \in E \setminus E_K$.*

By definition, kernels are low-degre subgraphs, and hence one can maintain a near-maximum degree in these graphs in time $O_\epsilon(d)$, by Proposition 2.2. The following proposition implies that such a near-maximum matching in a kernel K is also large in G. (See [1, 35, 11] for proofs.)

▶ **Proposition 2.4** (The Basic Kernel Lemma). *Let $\epsilon \in (0, 1/2)$ and $d \geq 1/\epsilon$. If $K = (V, E_K)$ is an (ϵ, d)-kernel of $G = (V, E)$, then $\mu(G) \leq \frac{2(1+\epsilon)}{1-\epsilon} \cdot \mu(K) \leq (2+8\epsilon) \cdot \mu(K)$.*

Combining propositions 2.2 and 2.4, one obtains $(2+O(\epsilon))$-approximate matching algorithms with update time $O((d/\epsilon^2) \cdot C(\epsilon, d) + T(\epsilon, d))$ from (ϵ, d)-kernel maintenance algorithms with $T(\epsilon, d)$ update time and $C(\epsilon, d)$ changes to the kernel per update to G. Our result of Theorem 1.1 follows precisely this well-trodden path. The key novelty in our work is showing how to surpass the approximation factor of $2+\epsilon$ obtainable using kernels alone.

Fast Bipartite Matching. In our main algorithm we dynamically maintain large matchings in dynamically changing bipartite subgraphs of G, using the following result of Bernstein and Stein [7].

▶ **Proposition 2.5.** *(Dynamic Bipartite Matching) Let $\epsilon > 0$. There exists a $(3/2+\epsilon)$-approximate dynamic bipartite matching algorithm with worst-case update time $O(\sqrt[4]{m} \cdot \mathrm{poly}(1/\epsilon))$.*

Bounding Recourse. Recall that to dynamize our main approach, we need that the augmenting edges A, which are the union of large matching M_i in appropriately-chosen bipartite subgraphs B_i, must change slowly. That is, the number of changes to these M_i per update, also referred to as *recourse*, should be small. For this, we rely on the following (minor extension) of the recent black-box reduction of Solomon and Solomon [32, Theorem 3], which we prove in the full version of the paper for completeness.

▶ **Proposition 2.6** (Bounded Recourse)**.** *For any $\alpha \geq 1$ and $\epsilon \in (0, 1/6)$, if there exist an α-approximate dynamic matching algorithm with worst-case update time T, then there exists an $\alpha(1+\epsilon)$-approximate dynamic matching algorithm with worst-case update time $T + O(1/\epsilon)$ and worst-case recourse bound $O(1/\epsilon)$. This applies under edge and/or node updates* ***and/or star updates.***

Solomon and Solomon [32] note that in some settings, bounded recourse translates to bounded running time. Our work provides another concrete example of this phenomenon, in a dynamic setting.

3 Identifying Augmentations to Kernels

In this section we characterize a bounded-degree subgraph A whose addition to K results in a better-than-two-approximate matching sparsifier, $AK := A \cup K$. In following sections we will show that this subgraph AK is also efficiently maintainable. To focus on the key ideas, we mostly provide brief proof sketches here, deferring most full proofs with detailed calculations to the full version of the paper.

Our starting point, in Section 3.1, is an extension of the proof of the approximation ratio of kernels, to obtain bounds on degrees in K of endpoints of any maximum matching M^* in G, whenever such K is worse than $(2-\delta)$-approximate. In Section 3.2 we leverage these degree bounds to characterize (most) augmenting paths of some maximum matching M in K in terms of the K-degree of their nodes. In Section 3.3 we use this characterization to prove that large matchings in two bipartite subgraphs, defined by degrees in K (broadly: low-degree and high-degree nodes on opposite sides), are precisely the desired set A. Unfortunately, defining these subgraphs based on degrees in K, which may change abruptly, causes updates to G (and hence to K) to result in *node updates* in these subgraphs, requiring $\Omega(n)$ update time. In Section 3.4 we show that large matchings in some $O(1)$ bipartite subgraphs defined by *approximate* degrees in K (causing these graphs to only change by small star updates) still result in a sufficient $\Omega(\delta \cdot \mu(G))$ augmenting paths with respect to M.

Notation and parameters. In this section we fix an (ϵ, d)-kernel $K = (V, E_H)$ of graph $G = (V, E)$ with $\epsilon = 2 \times 10^{-8}$ and $d \geq 1/\epsilon$. We also fix two additional parameters $\delta = 2 \times 10^{-6}$ and $s = 2 \times 10^{-4}$. We assume that $\mu(G) \geq (2-\delta) \cdot \mu(K)$, and search for a sparse augmentation A of K, resulting in a sparse subgraph $AK := A \cup K$ with $\mu(G) \leq (2-\delta) \cdot \mu(AK)$. Our choices of values for ϵ, d, s and δ are taken to make these numbers rather round, while allowing δ to be close to its highest possible value. We indicate by $f(\epsilon, s, \delta) \overset{\star}{\leq} g(\epsilon, s, \delta)$ inequalities which hold for sufficiently small ϵ, s, δ, ϵ/δ and δ/s, and which specifically hold for our particular choices of ϵ, s and δ.

3.1 The Extended Kernel Lemma

By Proposition 2.4, kernels are $(2 + O(\epsilon))$-approximate sparsifiers. In this section we prove that for a kernel K to be not much better than a 2-approximation, most edges of any maximum matching M^* in G must have their endpoint degrees in K sum to roughly d. More precisely, we prove the following.

▶ **Lemma 3.1** (Extended Kernel Lemma)**.** *For any maximum matching M^* in G, at most $\frac{\delta}{s} \cdot |M^*|$ edges $e \in M^*$ satisfy either of the following conditions.*

1. $\sum_{v \in e} d_K(v) \geq d \cdot (1 + s - 2\epsilon)$.
2. $\sum_{v \in e} d_K(v) \leq d \cdot (1 - s + 2\epsilon)$ *and* $e \in E_K$.

We now outline the proof approach, deferring a full proof with detailed calculations to the full version of the paper.

Proof (Sketch). The basic kernel lemma, Proposition 2.4, which this lemma extends, is obtained by considering a fractional matching $\vec{x}$ in K, with values determined by a maximum matching M^* in G. For this fractional matching $\vec{x}$, nodes' fractional degrees, $y_v := \sum_{e \ni v} x_e$, satisfy the following bound.

$$\sum_{v \in e} y_v \geq 1 - \epsilon \qquad \forall e \in M^*. \tag{1}$$

This bound directly implies that this fractional matching x has large value in terms of $|M^*| = \mu(G)$.

$$\sum_e x_e = \frac{1}{2} \sum_v y_v \geq \frac{1}{2} \cdot \sum_{e \in M^*} \sum_{v \in e} y_v \geq \frac{1}{2} \cdot (1-\epsilon) \cdot \mu(G). \tag{2}$$

In bipartite graphs G, where the fractional matching polytope is integral, this immediately implies that $\mu(K) \geq \sum_e x_e \geq \frac{1-\epsilon}{2} \cdot \mu(G)$. In general graphs, using an application of Vizing's edge coloring theorem (see the full version), one can show that for this particular fractional matching $\vec{x}$, the kernel K contains a large integral matching M of nearly the same size as x's value, namely

$$\mu(K) \geq |M| \geq \frac{1}{1 + 1/d} \cdot \sum_e x_e \geq \frac{1}{1+\epsilon} \cdot x_e. \tag{3}$$

Combining equations (2) and (3) directly yields Proposition 2.4.

Our key observation is that the fractional degree bound of Equation (1) is loose if either condition (1) or (2) hold for edge $e \in M^*$. For such edge e we have that $\sum_{v \in e} y_v \geq 1 + s - 4\epsilon$. Consequently, if r is the number of edges of M^* satisfying either condition, then the above observation and Equation (1) imply that the fractional matching $\vec{x}$ has size at least $\sum_e x_e \geq \frac{1-\epsilon}{2} \cdot \mu(G) + r \cdot (s - 3\epsilon)$. By Equation (3), an increase in r therefore increases our lower bound on $\mu(K)$ in terms of $\mu(G)$. Thus, the inequality $\mu(G) \geq (2-\delta) \cdot \mu(K)$, and some simple calculations, implies the claimed upper bound on r. ◀

3.2 Characterizing augmenting path node classes

In this section, we characterize the structure of length-three augmenting paths based on node degree classes, which we now define.

► **Definition 3.2.** *A node v has* super-high/high/medium/low *degree in K for $i \in [\frac{1}{\epsilon}]$, if it belongs to the following sets, respectively.*[3]

$$\begin{aligned}
V_{SH}^{(i)} &:= \left\{ v \in V \mid d_K(v) \geq d \cdot \left(1 - \epsilon - i\epsilon^2\right) \right\} \\
V_{H}^{(i)} &:= \left\{ v \in V \mid d_K(v) \geq d \cdot \left(1 - 2s - i\epsilon^2\right) \right\} \\
V_{M}^{(i)} &:= \left\{ v \in V \mid d_K(v) \in \left(d \cdot \left(s + i\epsilon^2\right), d \cdot \left(1 - 2s - i\epsilon^2\right)\right) \right\} \\
V_{L}^{(i)} &:= \left\{ v \in V \mid d_K(v) \leq d \cdot \left(s + i\epsilon^2\right) \right\}.
\end{aligned}$$

[3] For notational simplicity, we assume that d is an integer multiple of $1/\epsilon$. Since $d =\geq 1/\epsilon$, this is WLOG, up to rounding of d, which at most increases our running time by an $O(1)$ factor.

The proofs of sections 3.2 and 3.3 will hold for any $i \in [\frac{1}{\epsilon}]$. Therefore, for notational simplicity, we fix some such i, and drop the notation (i) from the node's classes. The reader will be relieved to know that they need not commit to memory the exact formulae defining the sets V_{SH}, V_H, V_M and V_L. In particular, until Section 3.4, these formulae will only be used explicitly in the following two observations and lemma.

▶ **Observation 3.3.** *We have that $V_{SH} \subseteq V_H$. Also, V_H, V_M, and V_L are disjoint.*

Next, we note that Lemma 3.1 implies constraints on degree classes of endpoints of M^* edges.

▶ **Observation 3.4.** *At most $\frac{\delta}{s} \cdot \mu(K)$ edges $(u, v) \in M^*$ satisfy either of the following conditions.*
1. $(u, v) \in (V_M \times V_{SH}) \cup (V_H \times V_H)$.
2. $(u, v) \in V_L \times (V_L \cup V_M)$.

Proof. Condition (1) is a special cases of Condition (1) of Lemma 3.1. On the other hand, every edge $(u, v) \in V_L \times (V_L \cup V_M)$ has no super-high degree endpoint, and so by Property (P_2) of a kernel, must belong to E_K. Therefore, Condition (2) is a special case of Condition (2) of Lemma 3.1 too. The bound on the number of edge in M^* satisfying either condition above therefore follows from Lemma 3.1. ◀

Next, we note that some maximum matching M in K matches most V_H nodes.

▶ **Lemma 3.5.** *Some maximum matching M in K leaves at most $7s \cdot \mu(K)$ nodes of V_H unmatched.*

Proof (Sketch). First, we note that there exists a (possibly non-maximum) matching M' in K which leaves at most $O(s) \cdot \mu(K)$ high-degree nodes in K unmatched. This follows by considering a $(d + 1)$-edge-coloring of K (guaranteed to exist by $\Delta(K) \leq d$ and Vizing's theorem), and noting that a high degree node v in such a color is matched with probability at least $d_K(v)/(d+1) \geq 1-O(s)$. Linearity of expectation then implies a bound of $O(s) \cdot |V_H|$ on the number of nodes in V_H unmatched by M'. However, $|V_H|$ can be shown to be $O(\mu(K))$, and so we find that M' leaves at most $O(s) \cdot \mu(K)$ unmatched nodes in V_H. The proof is completed by augmenting M' to be a maximum matching in M while only adding to the set of matched nodes. ◀

For the remainder of Section 3, we let M be the maximum matching in K guaranteed by Lemma 3.5. That lemma, and Property (P_2) of kernel K implies that M is an $O(s)$-*approximately maximal matching*, i.e., it is a matching which is maximal in a subgraph obtained by removing some $O(s) \cdot \mu(G)$ nodes (and their edges). These type of matchings were useful for Peleg and Solomon [31] when studying bounded-arboricity graphs, and will prove useful for us, too. In particular, since such a matching is maximal in a graph $G[V \setminus U]$ with essentially the same matching size as G, we can appeal to Proposition 2.1 to argue that M has many 3-augmentable edges (see the full version).

▶ **Lemma 3.6.** *At least $(1 - 88s) \cdot \mu(K)$ edges of M are 3-augmentable.*

In what follows, we will show that most length-three augmenting paths in $M \cup M^*$ belong to one of a small number of *types*, which we now define. Note that since M is maximum in K, it admits no augmenting paths in E_K, and hence an augmenting path $v_1 - v_2 - v_3 - v_4$ cannot have both of its M^* edges (v_1, v_2) and (v_3, v_4)) in E_K. Therefore, we assume WLOG that $(v_3, v_4) \notin E_K$.

▶ **Definition 3.7.** *We say a length-three augmenting path $p : v_1 - v_2 - v_3 - v_4$ with edges $e_i = (v_i, v_{i+1})$ for $i \in [3]$ and $e_3 = (v_3, v_4) \notin E_K$ is* frequent *if it belongs to one of the following* types *(see Figure 2):*

- *Type 1: $(v_1, v_2, v_3, v_4) \in V_L \times V_{SH} \times V_{SH} \times V_L$.*
- *Type 2: $(v_1, v_2, v_3, v_4) \in V_L \times (V_H \setminus V_{SH}) \times V_{SH} \times V_L$ and $e_1 \in E_K$.*
- *Type 3: $(v_1, v_2, v_3, v_4) \in V_M \times (V_H \setminus V_{SH}) \times V_{SH} \times V_L$ and $e_1 \in E_K$.*
- *Type 4: $(v_1, v_2, v_3, v_4) \in V_M \times V_M \times V_{SH} \times V_L$ and $e_1 \in E_K$.*

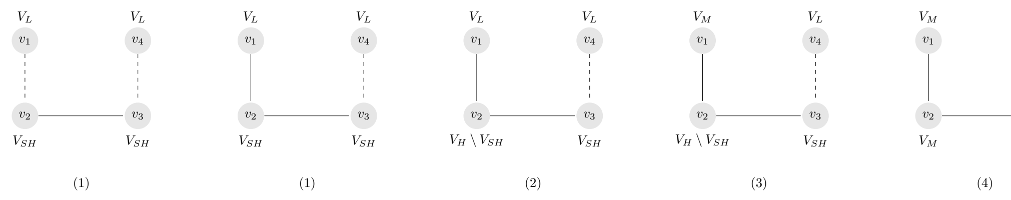

Figure 2 Frequent length-three augmenting paths.
Subfigures are labeled by their path's type number. (Note the two type (1) paths.)
Kernel edges are solid and non-kernel edges are dashed. Nodes are labeled by their class.

We now prove that these path types are indeed frequent, as their name suggests. Let n_f and n_{if} denote the number of frequent and infrequent (i.e., not frequent) length-three augmenting paths in $M \cup M^*$, respectively. We prove the following.

▶ **Lemma 3.8.** *We have that $n_{if} \leq \left(7s + \frac{\delta}{s}\right) \cdot \mu(K)$ and $n_f \geq (1 - 100s - \frac{\delta}{s}) \cdot \mu(K)$.*

Proof. In what follows, whenever discussing a length-three augmenting path, we denote it by $p : v_1 - v_2 - v_3 - v_4$, and denote its edges by $e_i = (v_i, v_{i+1})$ for $i \in [3]$. Since M is a maximum matching in K, we cannot have $e_1, e_3 \in E_K$, since the converse would entail an augmenting path for M in K, contradicting $|M| = \mu(K)$. So, we assume WLOG that $e_3 \notin E_K$. We start by upper bounding the number of infrequent paths, first considering them by their first node, v_1.

By Observation 3.4, at most $\frac{\delta}{s} \cdot \mu(K)$ length-three augmenting paths are infrequent paths due to $(v_1, v_2) \in V_L \times (V_L \cup V_M)$ or $(v_1, v_2) \in V_M \times (V_L \cup V_{SH})$. Now, by Lemma 3.5, at most $7s \cdot \mu(K)$ length-three augmenting paths have a V_H node as one of their endpoints (i.e., their nodes v_1, v_4, which are unmatched in M). This accounts for infrequent paths with $v_1 \in V_H$, as well infrequent paths with $v_4 \in H$. In all remaining paths, since $e_3 = (v_3, v_4) \notin E_K$, we have by Property (P_2) of the kernel that $v_3 \in V_{SH}$, and $v_4 \in V_L$ (the case $(v_3, v_4) \in V_{SH} \times (V \setminus V_L)$ is accounted for by the at most $\frac{\delta}{s} \cdot \mu(K)$ edges satisfying conditions 1 or 2 of Observation 3.4). All remaining paths with $(v_3, v_4) \in (V_{SH} \times V_L) \setminus E_K$ correspond to frequent paths of types (1), (2), (3) and (4), since they satisfy, respectively,

$$(v_1, v_2) \in (V_L \times V_{SH}) \cup (V_L \times (V_H \setminus V_{SH})) \cup (V_M \times (V_H \setminus V_{SH})) \cup (V_M \times V_M).$$

To summarize, the number of infrequent length-three augmenting paths is at most $n_{if} \leq \left(7s + \frac{\delta}{s}\right) \cdot \mu(K)$. By Lemma 3.6, the number of length-three augmenting paths is at least $(1 - 88s) \cdot \mu(K)$. Combined, these two bounds imply the second claim, i.e., $n_f \geq (1 - 95s) \cdot \mu(K) \geq (1 - 100s) \cdot \mu(K)$. ◀

By Lemma 3.8, many edges of length-three augmenting paths in $M \cup M^*$ have a (possibly super-)high-degree node as one endpoint, and a low-degree node as the other endpoint. This motivates us to compute large bipartite matchings between high- and low-degree nodes in K. We address this strategy in the following section.

3.3 Two bipartite matchings suffice

In this section, we prove that large matchings in judiciously-chose bipartite subgraphs contain many augmenting paths. We start by noting that one such naturally-chosen matching does not yield such augmenting paths.

One matching does not suffice. Let $G[A, B] := G[A \times B]$ denote the bipartite subgraph of G induced by bipartition (A, B). The previous section's characterization of most augmenting paths suggests that a large matching in $G[V_{SH}, V_L]$ together with edges of K might contain many augmenting paths with respect to M. However, this is not the case. For example, if all augmenting paths are of type (2), then matching all copies of v_3 to v_1 in the same path (the unmatched V_L neighbor of $v_2 \in V_H$) would not result in *any* augmenting paths. This example might suggest to instead compute a large matching in the subgraph $G[V_H, V_L]$. However, a similar problem then arises in paths of type (3).

To conclude, a large matching in either $G[V_{SH}, V_L]$ or $G[V_H, V_L]$ may not constitute an augmentation A of the kernel K. We will prove that a large matching in $G[V_{SH}, V_L]$ and a large matching in $G[V_H, V_L]$ *together* form the desired augmentation A of the kernel. Formally, we prove the following.

▶ **Lemma 3.9.** *Let M_H and M_{SH} be a $(3/2 + \epsilon)$-approximate matching in $G[V_H, V_L]$ and $G[V_{SH}, V_L]$, respectively. If $A(M_H)$ and $A(M_{SH})$ denote the minimum number of disjoint augmenting paths induced by the union of K and each of M_H and M_{SH}, respectively, then* $\max(A(M_H), A(M_{SH})) \geq (\delta + \epsilon) \cdot \mu(K)$.

The proof of the above lemma will require some setting up, starting with the following definition.

▶ **Definition 3.10.** *Node v is* bad *if $v \in V(M)$ and v is not in a frequent length-three augmenting path.*

▶ **Observation 3.11.** *If node $v \in V_L$ is not bad, then it is unmatched in M.*

▶ **Observation 3.12.** *The number of bad nodes is at most $b \leq (200s + \frac{2\delta}{s}) \cdot \mu(K)$.*

Proof. First, by Lemma 3.6, there are at most $88s \cdot \mu(K)$ edges of M that are not 3-augmentable. On the other hand, by Lemma 3.8 we have that $n_{if} \leq (7s + \frac{\delta}{s}) \cdot \mu(K)$. Combining both bounds on the number of edges containing a bad node, and noting that each such edge contributes two bad nodes, we have that $b \leq 2 \cdot (88s + 7s + \frac{\delta}{s}) \cdot \mu(K) \leq (200s + \frac{\delta}{s}) \cdot \mu(K)$, as claimed. ◀

We now lower bound the number of augmemting paths in $M_H \cup E_K$ and $M_{SH} \cup E_K$ in terms of numbers of frequent augmenting paths of the various types.

▶ **Lemma 3.13.** *Let $M_H, M_{SH}, A(M_H)$ and $A(M_{SH})$ be as in Theorem 3.19. Let n_i the number of frequent augmenting paths of type $i \in [4]$ (noting that $n_f = n_1 + n_2 + n_3 + n_4$). Then,*

$$A(M_H) \geq |M_H| - b - n_f.$$
$$A(M_{SH}) \geq |M_{SH}| - b - (n_1 + n_2).$$

Proof. Whenever discussing a frequent length-three augmenting path in this proof, we denote it by $v_1 - v_2 - v_3 - v_4$, as in Definition 3.7. We say an edge in M_H or M_{SH} is *good* if neither of its endpoints is bad. Consequently, we have at least $|M_H| - b$ and $|M_{SH}| - b$ good edges in M_H and M_{SH}, respectively. We say a node is *free* if it is not matched in M. By Observation 3.11, for any good edge $e \in M_H \cup M_{SH}$, the endpoint $v \in e \cap V_L$ must be free. We will show that these good edges form part of large sets of disjoint augmenting paths in M_H and M_{SH}.

We first consider the good edges of matching M_H, which by definition are between a free node in V_L and a V_H node which is either free or belongs to a frequent length-three augmenting path. Any edge in M_H between a free V_H node and a free V_L node constitutes an augmenting path of length one. Also, if two V_H nodes v_2, v_3 of the same frequent path of types (1), (2), or (3) match to free V_L nodes, u_2, u_3, then $u_2 - v_2 - v_3 - u_3$ is an augmenting path in $M_H \cup E_K$. Finally, if the unique V_{SH} node v_3 of a frequent path $v_1 - v_2 - v_3 - v_4$ of type (4) is matched to a free node u in V_L, then $v_1 - v_2 - v_3 - u$ is an augmenting path in $M_H \cup E_K$. (See Figure 3 for an illustration of some of the above cases.) We note further that the augmenting paths above are node disjoint, since no V_H or V_L node is matched twice, and therefore none of these nodes appear in two of these augmenting paths, and moreover, the node v_1 in paths of type (4) is not matched in M_H, since it belongs to V_M, which by Observation 3.3 is disjoint from V_L and V_H. We conclude that any more than one good match per path of type (1), (2) or (3) and any other kind of good match all contribute one augmenting path to this disjoint set of augmenting paths. Therefore, the maximum number of disjoint augmenting paths in $M_H \cup E_K$ is, as claimed, at least

$$A(M_H) \geq |M_H| - b - (n_1 + n_2 + n_3) \geq |M_H| - b - n_f.$$

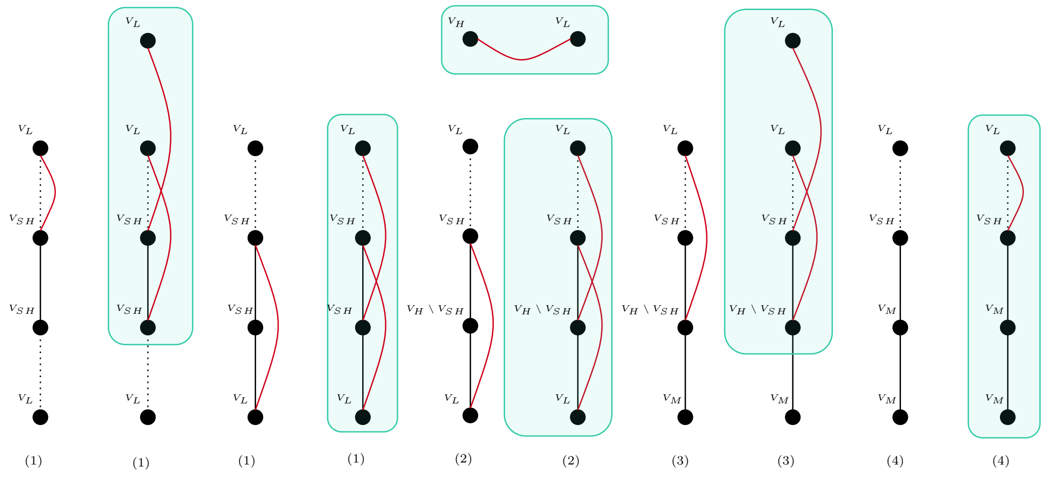

Figure 3 (Non-exhaustive) options for matches in M_H and relation to $A(M_H)$.
Kernel edges are solid and non-kernel edges are dashed, while edges in M_H are curved and red.
The number below each vertical line corresponds to the path type of the lowest four nodes in the line.
The blue boxes show augmenting paths in $E_K \cup M_H$.

We now consider the good edges of matching M_{SH}, only outlining the differences compared to the analysis for M_H. For matching M_{SH}, the only good matches which are guaranteed to form an augmenting paths are: (i) a good match between a free $u \in V_{SH}$ and a free $v \in V_L$ node, for which (u, v) is an augmenting path of length one, and (ii) matches between two V_{SH} nodes v_2, v_3 of the same frequent path of type (1) to free V_L nodes, u_2, u_3, for which $u_2 - v_2 - v_3 - u_3$ is an augmenting path. By similar reasoning to our study of M_H, we therefore conclude that the maximum number of disjoint augmenting paths in $M_{SH} \cup E_K$ is, as claimed, at least

$$A(M_{SH}) \geq |M_{SH}| - b - (n_1 + n_2). \qquad \blacktriangleleft$$

Armed with Lemma 3.13, we are now ready to prove that the matchings M_H and M_{SH} form a good augmentation of K. That is, we can now prove Lemma 3.9.

Proof of Lemma 3.9. By Observation 3.12, the number of bad nodes is at most $b \leq (200s + \frac{2\delta}{s}) \cdot \mu(K)$, and by Lemma 3.8, the number of frequent length-three augmenting paths is at least $n_f \geq (1 - 100s - \frac{\delta}{s}) \cdot \mu(K)$. However, in this proof, we will use the following looser bounds, $b \leq (200s + \frac{2\delta}{s} + 16\epsilon) \cdot \mu(K)$ and $n_f \geq (1 - 100s - \frac{\delta}{s} - 8\epsilon) \cdot \mu(K)$. Our use of these looser bounds will prove useful in Section 3.4.

We lower bound $|M_H|$ and $|M_{SH}|$, by first lower bounding the maximum matching sizes in the bipartite graphs $G[V_H, V_L]$. The matchings obtained by considering all $V_H \times V_L$ and $V_{SH} \times V_L$ edges in frequent length-three augmenting paths immediately imply that $\mu(G[V_H, V_L]) \geq n_f + n_1 + n_2$, and similarly $\mu(G[V_{SH}, V_L]) \geq n_f + n_1$. Here too, we will use the looser bounds $\mu(G[V_H, V_L]) \geq n_f + n_1 + n_2$, and similarly $\mu(G[V_{SH}, V_L]) \geq n_f + n_1$. On the other hand, M_H and M_{SH} are $(\frac{3}{2} + \epsilon) \leq \frac{1}{2/3-\epsilon}$-approximate matchings in these bipartite subgraphs. Combining the above with Lemma 3.13, we therefore get the following bounds on $A(M_H)$ and $A(M_{SH})$.

$$A(M_H) \geq \left(\frac{2}{3} - \epsilon\right) \cdot (n_1 + n_2 + n_f) - n_f - b \geq \frac{2}{9} \cdot n_1 + \frac{2}{3} \cdot n_2 - \frac{1}{3} \cdot n_f - b,$$

$$A(M_{SH}) \geq \left(\frac{2}{3} - \epsilon\right) \cdot (n_1 + n_f) - n_1 - n_2 - b \geq \frac{2}{3} \cdot n_f - \frac{1}{3} \cdot n_1 - n_2 - b.$$

Therefore, we have that

$$A(M_H) + A(M_{SH}) \geq A(M_H) + \frac{2}{3} \cdot A(M_{SH}) \geq \frac{1}{9} \cdot n_f - \frac{5}{3} \cdot b.$$

Combining the above with lemmas 3.8 and 3.12, we obtain the desired inequality

$$\begin{aligned}\max\left(A(M_H), A(M_{SH})\right) &\geq \frac{1}{2} \cdot \left(A(M_H) + A(M_{SH})\right)\\ &\geq \frac{1}{2} \cdot \left(\frac{1}{9} \cdot \left(1 - 100s - \frac{\delta}{s} - 8\epsilon\right) - \frac{5}{3} \cdot \left(200s + \frac{2\delta}{s} + 16\epsilon\right)\right) \cdot \mu(K)\\ &\overset{\star}{\geq} (\delta + \epsilon) \cdot \mu(K).\end{aligned}$$ ◀

3.4 Approximate degrees in the kernel

In the previous sections we show that the union of the kernel, a large matching in $G[V_{SH}, V_L]$, and a large matching in $G[V_H, V_L]$ contains a $(2 - \delta)$-approximate matching of G. However, just *maintaining subgraph* $G[V_{SH}, V_L]$ and $G[V_H, V_L]$ is costly and takes $\Omega(n)$ time to update in the dynamic setting as nodes join and leave these subgraphs when their degrees in K changes. In order to have less abrupt changes, we use approximate degrees for classifying nodes.

▶ **Definition 3.14.** *Let $\alpha \geq 0$. Elements of set $\{d^u(v) \mid u, v \in V\}$ are* α-approximate degrees *in K if for every pair of nodes $u, v \in V$,*

$$d_K(v) - \alpha \leq d^u(v) \leq d_K(v) + \alpha.$$

In the full version of the paper, we show how to maintain α-approximate degrees in K efficiently. For now, we will show that large matchings in bipartite subgraphs defined by $\epsilon^2 d$-approximate degrees contain many disjoint augmenting paths, mirroring Lemma 3.9 for bipartite subgraphs based on exact degrees in K. Specifically, we will consider the following bipartite subgraphs.

▶ **Definition 3.15.** *Let $\{d^u(v) \mid u, v \in V\}$ be $\epsilon^2 d$-approximate degrees in K. For $i \in [\frac{1}{\epsilon}]$, we denote by $B_H^{(i)} := (V, E_H^{(i)})$ and $B_{SH}^{(i)} := (V, E_{SH}^{(i)})$ bipartite subgraphs of G induced by the following edge sets.*

$$E_H^{(i)} = \left\{(u,v) \mid d^u(v) \geq d \cdot \left(1 - 2s - i\epsilon^2\right),\ d^v(u) \leq d \cdot \left(s + i\epsilon^2\right)\right\}.$$
$$E_{SH}^{(i)} = \left\{(u,v) \mid d^u(v) \geq d \cdot (1 - \epsilon - i\epsilon^2), \quad d^v(u) \leq d \cdot \left(s + i\epsilon^2\right)\right\}.$$

We recall that (informally) $\epsilon \ll s \ll 1$, and so the above subgraphs are indeed bipartite.

As in previous sections, we will drop the superscript (i) when i is clear from context. Intuitively, subgraphs B_H and B_{SH} are similar to the bipartite subgraphs $G[V_H, V_L]$ and $G[V_{SH}, V_L]$, respectively. However, since we use approximate degrees, these subgraphs are not equal to each other, since a node's degree class can be incorrectly classified by some of its neighbors, as in the following definition.

▶ **Definition 3.16.** *Fix $i \in [\frac{1}{\epsilon}]$. Let $v \in S$, where $S \in \{V_{SH}, V_H \setminus V_{SH}, V_M, V_L\}$. We say v is* misclassified *for i if some neighbor u of v has $d^u(v) = d'$, but if we had $d_K(v) = d'$, then we would have $v \notin S$.*

Intuitively, misclassifications may result a large number of matches in M_H and M_{SH} which do not increase $A(M_H)$ and $A(M_{SH})$ compared to the exact degree case. That is, such misclassifications result in a lower number of augmenting paths in the exact degree case. However, as we show, for some $i^* \in [\frac{1}{\epsilon}]$, such problematic miscalssifications are rare.

▶ **Observation 3.17.** *Let $\{d^u(v) \mid u, v \in V\}$ be $\epsilon^2 d$-approximate degrees in kernel K. Then there exists an $i^* \in [\frac{1}{\epsilon}]$ such that at most $8\epsilon \cdot \mu(K)$ nodes in length-three augmenting paths are misclassified for i^*.*

Proof. Since M is a maximum matching in K, length-three augmenting paths contain at most $4\mu(K)$ nodes. Since Let $\{d^u(v) \mid u, v \in V\}$ are $(\epsilon^2 d)$-approximate degrees, for a node v, we have that $\max_u |d^u(v) - d_K(v)| \leq \epsilon^2 d$. Hence, a node can be misclassified if the difference between its degree and the thresholds used to define a node's degree class is less than $\epsilon^2 d$. Since the difference between thresholds in Definition 3.2 for different i and j is at least $\epsilon^2 d$, each such node can be misclassified for at most two such indices $i \in [\frac{1}{\epsilon}]$. Therefore, there exists an i^* such that at most $8\mu(K)/(\frac{1}{\epsilon}) = 8\epsilon \cdot \mu(K)$ nodes are misclassified for i^*. ◀

We are now ready to generalize Lemma 3.9 to the approximate degree setting.

▶ **Lemma 3.18.** *Let $i^* \in [\frac{1}{\epsilon}]$ be the index guaranteed by Observation 3.17. Let M_H and M_{SH} be $(3/2 + \epsilon)$-approximate matchings in $B_H^{(i^*)}$ and $B_{SH}^{(i^*)}$, respectively. Let $A(M_H)$ and $A(M_{SH})$ denote the number of disjoint augmenting paths induced by the union of E_K and each of M_H and M_{SH}, respectively. Then,*

$$\max\left(A(M_H), A(M_{SH})\right) \geq (\delta + \epsilon) \cdot \mu(K).$$

Proof. We slightly abuse notation, and say a length-three augmenting path is frequent if it is of types (1), (2), (3), or (4) as in Definition 3.7 **and** none of its nodes are misclassified. By Lemma 3.8 and Observation 3.17, the number of frequent paths (under this new definition) is at least $n'_f \geq n_f - 8\epsilon \cdot \mu(K) \geq (1 - 100s - \frac{\delta}{s} - 8\epsilon) \cdot \mu(K)$. Now that with this new definition, a node v is *bad* if v belongs to both $V(M)$ and to an infrequent augmenting path, where infrequent paths also include paths with misclassified nodes. Therefore, by observations 3.12 and 3.17 the number of bad nodes (under this new definition) is at most $b' \leq b + 16\epsilon \cdot \mu(K) \leq (200s + \frac{2\delta}{s} + 16\epsilon) \cdot \mu(K)$. The maximum matching size in B_H and B_{SH} are the same as their counterparts without misclassifications, only replacig n_f and n_i for $i \in [4]$ by their tagged counterparts. That is, we have $\mu(B_{SH}) \geq n'_f + n'_1$ and $\mu(B_H) \geq n'_f + n'_1 + n'_2$. The above bounds on the number of bad nodes, frequent paths, and maximum matching sizes in B_H and B_{SH} are precisely the loose bounds in the proof of the desired inequality in Lemma 3.9. Repeating the same final calculations as in that proof, this lemma follows. ◀

Lemma 3.18 suggests a method for computing a sparse augmentation for K: compute large matchings in each $B_H^{(i)}$ and $B_{SH}^{(i)}$. Formally, we have the following.

▶ **Theorem 3.19.** *For $i \in [\frac{1}{\epsilon}]$, let $M_H^{(i)}$ and $M_{SH}^{(i)}$ be a $(3/2+\epsilon)$-approximate matching in $B_H^{(i)}$ and $B_{SH}^{(i)}$, respectively. The the edge set $A = \bigcup_{i \in [\frac{1}{\epsilon}]} \left(M_H^{(i)} \cup M_{SH}^{(i)} \right)$ augments K as follows.*

$$\mu(G) \leq (2 - \delta) \cdot \mu(G[E_K \cup A]).$$

Proof. By Lemma 3.18, there exists an $i^* \in [\frac{1}{\epsilon}]$ such that $G[E_K \cup M_H^{(i^*)} \cup M_{SH}^{(i^*)}]$ has at least $(\delta + \epsilon) \cdot \mu(K)$ augmenting paths with regards to M. Therefore, since $\mu(G) \leq (2 + 8\epsilon) \cdot \mu(K)$, by Proposition 2.4, and since $\delta \leq 1$, we obtain the desired inequality.

$$\begin{aligned}\mu(G[E_K \cup A]) &\geq \mu(G[E_K \cup M_H^{(i^*)} \cup M_{SH}^{(i^*)}]) \\ &\geq \mu(K) + (\delta + \epsilon) \cdot \mu(K) \geq \frac{1 + \delta + \epsilon}{2 + 8\epsilon} \cdot \mu(G) \overset{\star}{\geq} \frac{1}{2 - \delta} \cdot \mu(G).\end{aligned}$$ ◀

In Section 5 we show how to efficiently make use of the above structural characterizations of kernels and their augmentations to obtain the dynamic algorithm of our main result, Theorem 1. But first, we require an efficient algorithm for maintaining kernels, which we present in the following section.

4 Kernel Maintenance

In this section we describe a key component of our results: algorithms for kernel maintenance.

Bhattacharya et al. [11], in their paper introducing kernels, provided two algorithms for kernel maintenance, with amortized update times $O(\sqrt[3]{m}/\epsilon^2)$ and $O(\sqrt{n}/\epsilon)$, respectively.[4] They further showed how to de-amortize the former algorithm. Unfortunately for our needs, this algorithm maintains a kernel of maximum degree $d = \Omega(\sqrt[3]{m})$, and the number of changes to the kernel in this algorithm is $c = \Omega(\sqrt[3]{m})$. Consequently, maintaining near-maximum matchings in this kernel K would require $c \cdot d = \Omega(m^{2/3})$ update time, which is super-linear in n for moderately dense graphs.

In this section we show that the second algorithm of [11] can also be de-amortized (and sped up for sparse graphs). Crucially for our needs, the obtained algorithm will only cause $c = O(1)$ edge changes to the maintained kernel per edge update in G. This sub-linear update time and small number of edge changes in these kernels will prove useful for all of our results. We now turn to describing this kernel maintenance algorithm.

[4] Unlike in the preceding section, in this section $\epsilon > 0$ will not be a fixed constant.

4.1 The Basic Kernel Maintenance Algorithm

Next, we present a slightly simplified version of our kernel maintenance algorithm, with running time $O(n/(\epsilon d))$. For simplicity, we defer the details of our $O(\sqrt{m}/(\epsilon d))$ time algorithm to the full version of the paper.

Our algorithm maintains the edge set E_K using arrays and lists in a way to allow for insertion and deletions of edges in constant time. (See [15].) In addition, it maintains E in a similar structure, with the refinement that for each vertex $v \in V$, its list of neighbors in G, denoted by $N_G(v)$, is stored in a bidirectional circular list, and we maintain a pointer $ptr(v)$ in this list.

Our algorithm proceeds as follows: For insertion of an edge, we simply add it to K if both endpoints have degree below d in K (i.e., insertion does not violate Property (P_1)). Else, we do not change K. Regardless, for both endpoints of e, we insert the second endpoint into their linked list $N_G(v)$, *before* the node $ptr(v)$ points to. See the pseudocode in Algorithm 1. Removal is the more intricate procedure. For this procedure, we maintain, for each vertex v, a pointer $ptr(v)$ to its list of edges in G, stored in circular order. When v loses an edge in K due to a removal of an edge in G, we move $ptr(v)$ through the next neighbors of v in G, until we encounter a neighbor w of v in G with $d_K(w) < d$. In this case, we add (v, w) to K and terminate the procedure. Otherwise, the procedure terminates after $n/(\epsilon d)$ neighbors of v are inspected. See the pseudocode in Algorithm 2.

Algorithm 1 KERNEL-UPDATE:INSERT(e).

```
1: if $\max_{v \in e} d_K(V) < d$ then
2:     $E_K \leftarrow E_K \cup \{e\}$
3: for $v \in e$ do
4:     insert $u \in e \setminus \{v\}$ into $N_G(v)$ before the element $ptr(v)$ is pointing to
```

Algorithm 2 KERNEL-UPDATE:REMOVE(e).

```
1: for $v \in e$ do
2:     remove $u \in e \setminus \{v\}$ from $N_G(v)$
3: if $e \in E_K$ then
4:     $E_K \leftarrow E_K \setminus \{e\}$
5:     for $v \in e$ do
6:         for $i = 1, 2, \ldots, n/(\epsilon d)$ do
7:             advance $ptr(v)$ in $N_G(v)$    $\triangleright$ advance to next element in cyclic list of $N_G(v)$
8:             $w \leftarrow ptr(v).element$    $\triangleright$ get node in $N_G(v)$ pointed to by $ptr(v)$
9:             if $d_K(w) < d$ then
10:                $E_K \leftarrow E_K \cup \{(v, w)\}$
11:                return
```

The intuition behind Algorithm 2 is that for any edge not added to the kernel upon insertion, at least one of its endpoints v must have degree $d_K(v) = d$. The point of iterating through $n/(\epsilon d)$ neighbors of a vertex who lost an edge in E_K is to guarantee that by the time ϵd edges of this vertex have been deleted from E_K without a replacement edge of v being found in Algorithm 2, this vertex v must have considered each of its n (or fewer) neighbors u. When this happened, either (u, v) was added to E_K (and never removed), or $d_K(v) \geq d$. In either case, Property (P_2) is satisfied for edge (u, v).

Formalizing the above discussion, we now analyze the above algorithm.

For any $\epsilon > 0$ and d, the algorithm handling edge insertions and deletions using algorithms 1 and 2 is a deterministic dynamic algorithm for maintaining an (ϵ, d)-kernel using $O(n/(\epsilon d))$ worst-case update time and causing $O(1)$ changes to the kernel per update.

Proof. The running time and number of changes to E_K per update are immediate from the pseudocodode. Property (P_1) is also apparent, as we never add an edge to a vertex v which already has $d_K(v) = d$. The only non-trivial property is Property (P_2), which we now prove.

Fix an edge $e \in E$ and time point t. If e was added to E_K upon insertion of e, then, since edges are only removed from E_K when they are removed from G, edge e is still in E_K at time t, and so it vacuously satisfies Property (P_2). Suppose instead e was not added to E_K upon insertion. Therefore, by Algorithm 1, at that time we had that $\max_{v\in e} d_K(v) = d$. Let $v \in e$ be such that $d_K(v) = d$ at the latest time $t' < t$. Now, if $d_K(v) < d(1 - \epsilon)$ at time t, vertex v must have had at least ϵd edges deleted from E_K without a replacement edge found in lines 6-11. But then, during these ϵd applications of the $n/(\epsilon d)$ iterations of lines 6-11 for vertex v, we inspect all (at most) n neighbors of v which were not added after time t', and in particular we have had in one of these iterations $w = u$, where u is the second endpoint of $e = (v, u)$. Since we chose v to be the vertex in e which had $d_K(v)$ at the last point $t' < t$, we have that $d_K(w) < d$, and so we add e to E_K in Algorithm 2. Consequently, again relying on edges only being removed from E_K when they are removed from G, we have that edge e is in E_K at time t, and therefore vacuously satisfies Property (P_2) at time t. ◀

In full version of the paper, we refine the result of Section 4.1, by relying on the dynamic edge orientation algorithm of [7]. This allows us to speed up our kernel algorithm for sparse graphs, replacing the n factor in the running time of Section 4.1 by $\sqrt{m} \leq n$. From this, we obtain the following lemma. For any $\epsilon > 0$ and d, there exists a deterministic dynamic algorithm for maintaining an (ϵ, d)-kernel with $O(\sqrt{m}/(\epsilon d))$ worst-case update time, with $O(1)$ changes to the kernel per update.

4.2 Applications: Fast $(2 + \epsilon)$-Approximate (Weighted) Matching

Since (ϵ, d)-kernels have degree at most d (by property (P_1)), we can maintain near-maximum matchings in dynamic kernels in time $O_\epsilon(d)$ per edge change in kernels, by Proposition 2.2. Combined with Section 4.1 and Proposition 2.4, our deterministic kernel maintenance algorithms therefore yield the first deterministic $(2 + \epsilon)$-approximate dynamic matching algorithm with worst-case sublinear update time. Specifically, we obtain the following.

▶ **Theorem 4.1.** *There exists a deterministic $(2+\epsilon)$-approximate dynamic matching algorithm with worst-case update time $O(\sqrt[4]{m}/\sqrt{\epsilon}) = O(\sqrt{n}/\sqrt{\epsilon})$.*

Proof. Let $d = \sqrt[4]{m}/\sqrt{\epsilon}$. Our algorithm maintains an (ϵ, d)-kernel K in deterministic $O(\sqrt{m}/(\epsilon d))$ worst-case update time, Section 4.1. By Proposition 2.2 and property (P_1), we can maintain a $(1 + \epsilon)$-approximate matching in K in deterministic $O(d/\epsilon^2)$ worst-case update time per edge update in K. By Section 4.1, our kernel K has $O(1)$ edges added or removed per update, and so maintaining this matching in K contributes an $O(d/\epsilon^2)$ term to the (deterministic, worst-case) update time. Finally, by the kernel lemma 2.4, this matching satisfies $|M| \geq \frac{1}{1+\epsilon} \cdot \mu(K) \geq \frac{1}{1+\epsilon} \cdot \frac{1}{2+\epsilon} \cdot \mu(G)$. Consequently, this algorithm maintains a $(1 + \epsilon) \cdot (2 + \epsilon) = 2 + O(\epsilon)$ approximate matching in G in deterministic $O(\sqrt{m}/(\epsilon d) + d/\epsilon^2)$ worst-case update time. By our choice of $d = \sqrt[4]{m}/\sqrt{\epsilon}$, the theorem follows. ◀

For weighted matching, we obtain the same asymptotic running time in terms of n and m, albeit with an exponential dependence on ϵ, by relying on the recent weighted matching framework of Bernstein et al. [5]. Letting R denote the ratio of the maximum weight and minimum weights ever in the graph, we obtain the following. (See the full version for details, as well as a discussion on exponentially decreasing this dependence on R.)

▶ **Theorem 4.2.** *There exists a deterministic* $(2+\epsilon)$*-approximate dynamic* weighted *matching algorithm with worst-case update time* $O(\sqrt[4]{m}/\epsilon^{O(1/\epsilon)} \cdot \log R) = O(\sqrt{n}/\epsilon^{O(1/\epsilon)} \cdot \log R)$.

Theorem 1.1 follows by combining the preceding two theorems.

5 Putting it all Together: Beating the Folklore Algorithm

In this section we combine our ideas from the previous section to obtain our main result – a deterministic algorithm which outperforms the folklore algorithm in terms of both approximation ratio and worst-case update time. Specifically, we prove the following.

▶ **Theorem 1.** *(Beating the Folkore Algorithm) There exists a deterministic* 1.999999*-approximate dynamic matching algorithm with worst-case update time* $O(m^{3/8})$*.*

Proof. As in Section 3, we let $\epsilon = 2 \times 10^{-8}$ and $\delta = 2 \times 10^{-6}$. Set $d = m^{3/8} \geq 1/\epsilon$. This last inequality is WLOG, since if $m^{3/8} < 1/\epsilon$, then we can trivially maintain a 1-approximate matching in worst-case time $\text{poly}(n) = \text{poly}(1/\epsilon)$ per update, by recomputing a maximum matching after each update. Furthermore, in this proof we assume that m is fixed (up to some multiplicative constant). This too is WLOG, since whenever m increases or decreases by a constant factor, we can (gradually) re-run the dynamic matching algorithm with an appropriately larger/smaller value of m, incurring a constant multiplicative overhead.

Our algorithm consists of a number of subroutines applied to several subgraphs of the dynamic graph G. Our algorithm maintains an (ϵ, d)-kernel $K = (V, E_K)$. In addition, it maintains $\epsilon^2 d$-approximate degrees $\{d^u(v) \mid u, v \in V\}$ for all node pairs $u, v \in V$, and, for all $i \in [1/\epsilon]$, the subgraphs $B_{SH}^{(i)}$ and $B_H^{(i)}$ induced by these approximate degrees, as defined in Section 3.4, where all (star) updates to approximate degree counters $d^u(v)$ affect the $O(1)$ appropriate subgraphs $B_{SH}^{(i)}$ and $B_H^{(i)}$. In each of the bipartite subgraphs $B_{SH}^{(i)}$ and $B_H^{(i)}$, we maintain $(3/2+\epsilon)$-approximate matchings, $M_{SH}^{(i)}$ and $M_H^{(i)}$, respectively, using the bipartite matching algorithm of [7], making it incur only $O(1/\epsilon)$ recourse per star update in $B_{SH}^{(i)}$ and $B_H^{(i)}$, using Proposition 2.6. Finally, we maintain a $(1+\delta/8)$-approximate matching to use as our output, M^{out}, in the low-degree graph $AK := G\left[E_K \cup \bigcup_i \left(M_i^H \cup M_i^{SH}\right)\right]$, using Proposition 2.2.

As we shall show, the above is a $((2-\delta)\cdot(1+\delta/8))$-approximate algorithm with $O(\sqrt[4]{m}\cdot\sqrt{\Delta})$ update time. To speed up this algorithm for sparse graphs, we combine our algorithm with dynamic sparsification (essentially a corollary of the sparsification scheme of [34]), which allows us to decrease Δ to $O(\sqrt{m}/\epsilon)$ at the cost of increasing the approximation ratio by a $(1+\epsilon)$ factor.

Approximation Ratio. By Theorem 3.19, we have that $\mu(G) \leq (2-\delta)\cdot\mu(AK)$. Also, by applying dynamic sparsification (see details in the full version) to our algorithm, we lose a $(1+\epsilon)$ factor in approximation ratio. Therefore, the $(1+\delta/8)$ approximate matching M^{out} which we maintain satisfies the following.

$$\mu(G) \leq (2-\delta)\cdot\mu(AK) \leq (2-\delta)\cdot(1+\epsilon)\cdot(1+\delta/8)\cdot|M^{out}| \leq (2-\delta/2)\cdot|M^{out}|.$$

That is, this algorithm is $(2-10^{-6})$-approximate, as claimed.

Running Time.[5] By Section 4.1, the maintenance of the (ϵ, d)-kernel $K = (V, E_K)$ requires $O(\sqrt{m}/(\epsilon d))$ time and $O(1)$ changes to E_K per update. For each insertion or removal of an edge (u, v) in E_K, we update the approximate degrees, in $O(\Delta/(\epsilon^2 d))$ time, causing an insertion or removal of $O(1)$ stars (centered on u or v) with $O(\Delta/(\epsilon^2 d))$ leaves in the bipartite subgraphs of the form $B^{(i)}_{SH}$ and $B^{(i)}_{H}$ (see the full version). Consequently, the time to maintain the auxiliary approximate matchings $\{M^{(i)}_H, M^{(i)}_{SH}\}_i$ in these bipartite subgraphs $B^{(i)}_H$ and $B^{(i)}_{SH}$ following a star update is $O(\Delta/(\epsilon^2 d)) \cdot O(\sqrt[4]{m}/\text{poly}(\epsilon)) = O(\Delta \cdot \sqrt[4]{m}/d)$, by Proposition 2.5. Therefore, since only $O(1)$ such star updates occur per update in G, the time per update to maintain the $1/\epsilon = O(1)$ matchings $M^{(i)}_{SH}$ and $M^{(i)}_H$ is $O(\Delta \cdot \sqrt[4]{m}/d)$. Moreover, each star update results in $O(1/\epsilon) = O(1)$ changes to each of these auxiliary matchings, by Proposition 2.6, and so $A := \bigcup_i (M^{(i)}_{SH} \cup M^{(i)}_H)$ changes by $O(1/\epsilon)$ edges per update to G. We conclude that the graph $AK := G[A \cup E_K]$ is a graph of maximum degree $d + O(1) = O(d)$, which changes by $O(1)$ edges per update: $O(1)$ changes to the kernel per update, by Section 4.1 and $O(1)$ changes in the matchings, by the preceding discussion. Therefore, we can maintain a $(1 + \delta/8)$-approximate matching in AK in $O(d/\delta^2) = O(d)$ time per update to AK, by Proposition 2.2, which is $O(d) \cdot O(1) = O(d)$ time per update to G. We conclude that this algorithm's (worst-case) update time is $O(d + \Delta \cdot \sqrt[4]{m}/d)$, which can decrease to $O(d + m^{3/4}/d)$ for sparse graphs by using dynamic sparsification (see the full version). Choosing $d = m^{3/8}$, yielding the claimed time bound. ◀

References

1 Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 79:1–79:16, 2018.

2 Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $O(\log n)$ update time. *SIAM Journal on Computing (SICOMP)*, 44(1):88–113, 2015.

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

4 Soheil Behnezhad, Jakub Łącki, and Vahab Mirrokni. Fully dynamic matching: Beating 2-approximation in Δ^ϵ update time. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2492–2508, 2020.

5 Aaron Bernstein, Aditi Dudeja, and Zachary Langley. A framework for dynamic matching in weighted graphs. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing (STOC)*, 2021.

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

7 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 167–179, 2015.

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

[5] All following bounds are worst-case, so we omit the words worst-case for brevity.

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

10 Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. Online edge coloring algorithms via the nibble method. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2830–2842. SIAM, 2021.

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

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

13 Sayan Bhattacharya and Peter Kiss. Deterministic rounding of dynamic fractional matchings. In *Proceedings of the 48th International Colloquium on Automata, Languages and Programming (ICALP)*, 2021.

14 Moses Charikar and Shay Solomon. 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)*, pages 33:1–33:14, 2018.

15 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms.* MIT press, 2009.

16 Søren Dahlgaard. 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)*, pages 48:1–48:14, 2016.

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

18 Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*, pages 26–37, 2019.

19 Fabrizio Grandoni, Chris Schwiegelshohn, Shay Solomon, and Amitai Uzrad. Maintaining an edcs in general graphs: Simpler, density-sensitive and with worst-case time bounds. *To appear in 5th Symposium on Simplicity in Algorithms (SOSA)*, 2022.

20 Anupam Gupta and Roie Levin. Fully-dynamic submodular cover with bounded recourse. In *Proceedings of the 61st Symposium on Foundations of Computer Science (FOCS)*, pages 1147–1157, 2020.

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

22 Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms. *arXiv preprint arXiv:2102.11169*, 2021.

23 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 21–30, 2015.

24 Peter Kiss. Improving update times of dynamic matching algorithms from amortized to worst case. *To appear in 13th Innovations in Theoretical Computer Science Conference (ITCS)*, 2022.

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

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

27 Silvio Micali and Vijay V Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algoithm for finding maximum matching in general graphs. In *Proceedings of the 21st Symposium on Foundations of Computer Science (FOCS)*, pages 17–27, 1980.

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

29 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Transactions on Algorithms (TALG)*, 12(1):7, 2016.

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

31 David Peleg and Shay Solomon. Dynamic $(1 + \epsilon)$-approximate matchings: a density-sensitive approach. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 712–729, 2016.

32 Noam Solomon and Shay Solomon. A generalized matching reconfiguration problem. In *Proceedings of the 12th Innovations in Theoretical Computer Science Conference (ITCS)*, 2021.

33 Shay Solomon. Fully dynamic maximal matching in constant update time. In *Proceedings of the 57th Symposium on Foundations of Computer Science (FOCS)*, pages 325–334, 2016.

34 Shay Solomon. Local algorithms for bounded degree sparsifiers in sparse graphs. In *Proceedings of the 9th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 52:1–52:19, 2018.

35 David Wajc. Rounding dynamic matchings against an adaptive adversary. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 194–207, 2020.