Deterministic Dynamic Maximal Matching in Sublinear Update Time

Aaron Bernstein*
bernstei@gmail.com
New York University
New York, NY, USA

Sayan Bhattacharya
S.Bhattacharya@warwick.ac.uk
University of Warwick
Coventry, United Kingdom

Peter Kiss†
peter.kiss@univie.ac.at
University of Vienna
Vienna, Austria

Thatchaphol Saranurak‡
thsa@umich.edu
University of Michigan
Michigan, Michigan, USA

Abstract

We give a fully dynamic deterministic algorithm for maintaining a maximal matching of an n-vertex graph in $\tilde{O}(n^{8/9})$ amortized update time. This breaks the long-standing $\Omega(n)$-update-time barrier on dense graphs, achievable by trivially scanning all incident vertices of the updated edge, and affirmatively answers a major open question repeatedly asked in the literature Baswana, Gupta and Sen [FOCS 2011], Bhattacharya,Chakrabarty, Henzinger and Nanongkai [SODA 2018], Solomon [Dagstuhl].

We also present a faster randomized algorithm against an adaptive adversary with $\tilde{O}(n^{3/4})$ amortized update time.

Our approach employs the *edge degree constrained subgraph* (EDCS), a central object for optimizing approximation ratio, in a completely novel way; we instead use it for maintaining a matching that matches *all* high degree vertices in sublinear update time so that it remains to handle low degree vertices rather straightforwardly. To optimize this approach, we employ tools never used in the dynamic matching literature prior to our work, including sublinear-time algorithms for matching high degree vertices, random walks on directed expanders, and the monotone Even-Shiloach tree for dynamic shortest paths.

CCS Concepts

• **Theory of computation → Dynamic graph algorithms**.

Keywords

Dynamic Algorithm, Graph Algorithm, Maximum Matching, Maximal Matching

*Supported by Sloan Fellowship, Google Research Fellowship, NSF Grant 1942010, and Charles S. Baylis endowment at NYU.

†This research was funded in whole or in part by the Austrian Science Fund (FWF) 10.55776/ESP6088024, This work was partially done while at the University of Warwick

‡Supported by NSF Grant CCF-2238138. Part of this work was done while at INSAIT, Sofia University "St. Kliment Ohridski", Bulgaria. This work was partially funded from the Ministry of Education and Science of Bulgaria (support for INSAIT, part of the Bulgarian National Roadmap for Research Infrastructure).

STOC '25, Prague, Czechia

ACM ISBN 979-8-4007-1510-5/25/06
https://doi.org/10.1145/3717823.3718153

ACM Reference Format:
Aaron Bernstein, Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. 2025. Deterministic Dynamic Maximal Matching in Sublinear Update Time. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC '25), June 23–27, 2025, Prague, Czechia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3717823.3718153

1 Introduction

A *matching* M of a graph $G = (V, E)$ is a set of vertex-disjoint edges. We say that M is *maximal* if every free edge $(u, v) \in E \setminus M$ is not vertex-disjoint from M. That is, a maximal matching is precisely a set of edges that are disjoint yet intersect all other edges.

Although computing a maximal matching admits a simple linear-time greedy algorithm, it presents interesting challenges in various computational models and has been extensively studied since the 80s, including in the PRAM model [3, 12, 69, 86, 87, 90, 96], distributed models [15, 16, 66, 81, 82], and, more recently, the MPC model [7, 23, 62, 68, 71, 94]. In this paper, we study this problem in the dynamic setting.

Dynamic Maximal Matching. In the *fully dynamic maximal matching* problem, we must maintain a maximal matching M of an n-vertex m-edge graph G undergoing both edge insertions and deletions. The goal is to minimize the *update time* for updating M, given an edge update. Since the 90s, Ivković and Lloyd [88] showed the first non-trivial algorithm that has $O((n+m)^{0.7072})$ amortized update time and is deterministic.

A key milestone was made by Baswana, Gupta, and Sen [17, 19] who showed a near-optimal randomized algorithm with $O(\log n)$ amortized update time. This was further improved to an optimal $O(1)$ update time by Solomon [102]. Different poly$(\log n)$-update-time algorithms with additional properties were given in [22, 33, 70]. Unfortunately, all these algorithms share a common drawback; they are all randomized and assume an *oblivious adversary.*[1]

The state of the art of deterministic algorithms is much worse. No previous algorithms could even strictly beat the following trivial solution: When (u, v) is inserted, match u and v if both u and v were unmatched. When (u, v) is deleted, check if u or v can be matched by scanning through $O(n)$ neighbors of u and v. This straightforwardly

[1]Dynamic algorithms assume an *oblivious adversary* if they assume that the whole update sequence is fixed from the beginning and is independent of the answers maintained by the algorithm. Without this assumption, we say that the algorithms work against an *adaptive* adversary.

takes $O(n)$ update time. While there are algorithms with $\tilde{O}(\alpha)$ worst-case update time where α is the arboricity [57, 99], they still require $\Omega(n)$ update time in dense graphs. Overcoming the $\Omega(n)$ deterministic barrier remains a long-standing open problem.

Success in Dynamic Approximate Maximum Matching. The influential result of [17, 19] also implies a fully dynamic algorithm for maintaining 2-approximate maximum matching. Since then, the community has shifted its focus to fully dynamic *approximate maximum* matching in various approximation regimes, including the approximation factors of $(2+\epsilon)$ [4, 33, 39, 41–43, 49, 53, 92], $(2-\epsilon)$ [14, 20, 24, 42, 47, 100], $(1.5+\epsilon)$ [35, 37, 74, 92], $(1.5-\epsilon)$ [21], and $(1+\epsilon)$ [8, 76, 95]. Numerous papers dove into specialized topics, including partially dynamic algorithms [10, 34, 45, 51, 52, 54, 73, 77, 89], dynamic approximation-preserving reductions from weighted to unweighted graphs [31, 32, 76, 105], and dynamic rounding algorithms [4, 44, 48, 64, 107].

Within the last decade, these 40+ papers have made the dynamic matching problem one of the most actively studied dynamic graph problems. A significant part of this body of work [39, 41–43, 49, 75, 92] successfully developed deterministic algorithms whose update times match those of their randomized counterparts.

The Barrier Remains. But despite the successes mentioned above, the $\Omega(n)$ deterministic barrier for dynamic maximal matching remains unbroken. A high-level explanation is that a maximal matching is far more fragile than an approximate matching. For approximate matching, via standard reductions [11, 92], it requires $\Omega(\epsilon n)$ edge updates before the maximum matching size changes by a $(1+\epsilon)$ factor. In contrast, a single edge deletion can destroy maximality completely.

This challenge has become a major open problem in the field and has been repeatedly posed as an open question [18, 40, 104]. The problem remains unresolved even for randomized algorithms against an *adaptive* adversary.

In this paper, we break this long-standing $\Omega(n)$ deterministic barrier.

Theorem 1.1. *There is a* deterministic *fully dynamic maximal matching algorithm with $\tilde{O}(n^{8/9})$ amortized update time. Also, there is a randomized fully dynamic maximal matching algorithm with $\tilde{O}(n^{3/4})$ amortized update time that works with high probability against an* adaptive adversary.[2]

Thus, we give the first deterministic maximal matching algorithm with sublinear update time. Using randomization but *without* an oblivious adversary, we can speed up the algorithm even further.

In general, significant effort has been made towards closing the gap between oblivious and adaptive adversaries in dynamic graph problems, which include dynamic connectivity [58, 72, 85, 97, 98, 108], sparsifiers [28, 50, 55, 72, 80, 93, 106], and shortest paths [26, 29, 30, 59–61, 78–80, 93]. These algorithms are crucial building blocks in the modern development of fast static graph algorithms [1, 2, 52, 56]. We take the first step in this direction for dynamic maximal matching.

[2]The $\tilde{O}$ notation hides poly $\log n$ factors.

Previous Techniques. The key question in designing dynamic maximal matching algorithms is how to handle deletions of a matched edge, as it is relatively easy to handle edge insertions or deletions of an unmatched edge. A trivial solution is that, whenever a vertex v is unmatched, we scan through v's neighbors and try to match v. This takes $O(\deg(v)) = O(n)$ update time.

The previous randomized algorithms [19, 22, 70, 103] speed this up by trying to ensure that the matched edges are not deleted too often. For intuition, consider the following simplistic scenario when the graph is a star with root u and leaves $v_1, \ldots, v_n$, and the adversary only deletes edges. Suppose we sample a random edge (u, v_i) and include it in the matching. If *the adversary is oblivious to our random choices*, then it will take $\Omega(n)$ edge deletions in expectation before the adversary deletes our matched edge (u, v_i). Once this happens, even if we spend $O(n)$ time, we can charge this cost to the $\Omega(n)$ deletions, leading to $O(1)$ amortized update time. This is *the* basic idea that all previous randomized algorithms exploited and successfully carried out beyond this simplistic scenario. Unfortunately, it completely fails without an oblivious adversary, and, in particular, for deterministic algorithms.

A Bird's-Eye View of Our Algorithm. Our deterministic algorithm is very different. We maintain a vertex set $V^\star$ in $o(n)$ update time such that

(1) the maximum degree in $G[V \setminus V^\star]$ is $o(n)$[3], and
(2) we can further maintain a $V^\star$-*perfect* matching M_{base} that matches *all* vertices in $V^\star$ in $o(n)$ update time.

Given this, we can maintain a maximal matching $M \supseteq M_{\mathsf{base}}$ in $o(n)$ update time. Indeed, if vertex v becomes unmatched, we only need to scan through v's neighbors in $V \setminus V^\star$, which has $o(n)$ vertices, as all vertices in $V^\star$ are already matched by M_{base}. Observe that this argument crucially requires that *all* vertices of $V^\star$ are matched. Maintaining a perfect matching is normally very difficult in the dynamic setting and there are strong conditional lower bound for this task [63, 84]. So we will need to pick a $V^\star$ that has additional structure.

New Applications of EDCS.. Surprisingly, we can obtain $V^\star$ from an *edge-degree constrained subgraph* (EDCS), a well-known subgraph $H \subseteq G$ that is useful for $(1.5+\epsilon)$-approximate maximum matching algorithms in the dynamic setting and beyond [6, 9, 13, 20, 27, 36, 38, 46, 75]. See Definition 2.1 for definition. The outline below presents a completely novel application of this central object in the literature.

The set $V^\star$ is simply the set of vertices with "high degree" in the EDCS H, which can be explicitly maintained in $o(n)$ time using existing results [74]. By the structure of EDCS, Item 1 follows directly. To see why Item 2 should hold, we observe that the graph $H_{\texttt{hilo}} := E_H(V^\star, V \setminus V^\star)$ has a *degree gap*. More precisely, for some number X and $\gamma > 0$, every vertex in $V^\star$ has degree at least X in $H_{\texttt{hilo}}$, while every vertex in $V \setminus V^\star$ has degree at most $(1-\gamma)X$ in $H_{\texttt{hilo}}$. Consequently, for every set $S \subseteq V^\star$, we have $|N_{H_{\texttt{hilo}}}(S)| \geq (1+\Omega(\gamma))|S|$, i.e., Hall's condition holds *with a slack*. This strong expansion implies the existence of short $O(\log(n)/\gamma)$-length augmenting paths and allows us to maintain a $V^\star$-perfect matching in $o(n)$ update time.

[3]Technically, we actually show that $G[V \setminus V^\star]$ has arboricity $o(n)$.

New Toolkit for Dynamic Matching. To carry out the above approach, we employ tools never used in the dynamic matching literature prior to our work. For example, to maintain the $V^\star$-perfect matching deterministically, we use the monotone Even-Shiloach tree for dynamic shortest paths [83, 91, 101] and, for our faster randomized algorithm, we apply random walks on directed expanders. Although not necessary for breaking the $\Omega(n)$ barrier, we also apply a a sublinear-time algorithm for matching high-degree vertices [5] to further optimize our update time.

Concurrent Work. Computing maximal matching is among the fundamental symmetry breaking problems, that also include computing maximal independent set, vertex coloring and edge coloring. Very recently, Behnezhad, Rajaraman and Wasim [25] showed the first randomized fully dynamic algorithm for maintaining a $(\Delta+1)$-vertex coloring against an adaptive adversary in sub-linear update time.

Organization. We first present the preliminaries in Section 2 and the subroutine for efficiently matching most vertices with almost maximum degree in Section 3. They are needed for our detailed overview in Section 4. Then, we present a complete algorithm for maintaining perfect matching of high-degree vertices in a graph with degree gap in Section 5. Due to page limitations we defer the precise description of our algorithm to the full version of the paper.

2 Preliminaries

Basic Notations. Consider any graph $G = (V, E)$. We let $N_G(v) \subseteq E$ denote the set of edges in G that are incident on a node $v \in V$, and we define $\deg_G(v) := |N_G(v)|$ to be the degree of v in G.

Matching. A matching $M \subseteq E$ in a graph $G = (V, E)$ is a subset of edges that do not share any common endpoint. We let $V(M)$ denote the set of nodes matched under M. Furthermore, for any set $S \subseteq V$, we let $M(S) := \{(u, v) \in M : \{u, v\} \cap S \neq \emptyset\}$ denote the set of matched edges under M that are incident on some node in S.

Let M be a matching in G. We say that a vertex v is M-free if $v \notin V(M)$, otherwise it is M-matched. An *augmenting path* P in G with respect to M is a path in G such that the endpoints are M-free and its edges alternate between edges in M and edges not in M. Let $M \oplus P$ stand for $(M \setminus P) \cup (P \setminus M)$, the matching obtained by extending M with the augmenting path P. We have $M \oplus P$ is a matching of size $|M| + 1$. Every M-matched vertex is also matched in $M \oplus P$.

EDCS.. For any graph H and $e = (u, v)$ where e is not necessary an edge of H, the degree of e in H is defined as $\deg_H(e) \stackrel{\text{def}}{=} \deg_H(u) + \deg_H(v)$. The basis of our algorithm is an edge degree constrained subgraph (EDCS).

Definition 2.1. *An (B, B^-)-EDCS H of G is a subgraph of G such that*

- *For each edge $e \in H$, $\deg_H(e) \leq B$, and*
- *For each edge $e \in G \setminus H$, $\deg_H(e) \geq B^-$.*

Theorem 2.2 ([74]). *For any parameters $B \leq n$ and $\epsilon > 0$, there is a deterministic algorithm that, given a graph G with n vertices undergoing both edge insertions and deletions, explicitly maintains a $(B, B(1-\epsilon))$-EDCS H of G using $O(\frac{n}{B\epsilon})$ worst-case update time. Furthermore, there are at most two vertices $u, v \in V$ such that their degree $\deg_H(v), \deg_H(u)$ in the EDCS changes due to the handling of an update.*

Graph Access. We say that an algorithm has *adjacency matrix query* access to graph G if it can answer queries with inputs $(u, v) \in V \times V$ returning true iff $(u, v) \in E$ in $O(1)$ time. We say that an algorithm has *adjacency list query* access to graph G if it can find the degree of any vertex and answer queries with input $(v, i) \in V \times [\deg_G(v)]$ returning the i-th neighbour of vertex v in G according to some arbitrary ordering in $O(\log n)$ time.

3 Match Most Vertices with Almost Maximum Degree

Next, we give a useful observation that there always exists a matching that matches most all vertices with almost maximum degree. Our deterministic and randomized implementations of the algorithm of Lemma 3.1 rely on a black box application of [65] and a white box application of [5]. As the lemma can mostly be concluded from existing results in literature we defer its proof to the complete version of the paper.

Lemma 3.1. *Let $G = (V, E)$ be a graph with maximum degree at most Δ which can be accessed through both adjacency list and matrix queries. Let V_κ denote the set of all nodes in G with degree at least $(1-\kappa)\Delta$. Then, for any $\kappa > 0$ there exists a matching $M \subseteq E$ that matches all but $2\kappa n$ many nodes in V_κ.*

- *If $\Delta \geq \frac{4}{\kappa}$, then M can be computed in $O(n\Delta\kappa \log^2 n)$ time with high probability.*
- *M can be deterministically computed in $O(m\frac{\log n}{\kappa}) = O(n\Delta \cdot \frac{\log n}{\kappa})$ time.*

Remark 3.2. *While the specific bounds of Lemma 3.1 rely on recent advances in static coloring algorithms, any bound of the form $O(m\text{poly}(\Delta/\kappa))$ would translate to a sublinear update time for our main result of fully dynamic maximal matching. For example, one could instead use the classical coloring algorithm of Gabow et al. [67], or one could start with a trivial fractional matching and use existing tools to round it to an integral matching one (see e.g. [48]).*

Remark 3.3. *The randomized algorithm of Lemma 3.1 can be improved to have a running time of $\tilde{O}(n)$ using a white-box modification of [5]. As this running time difference doesn't ultimately end up affecting the update time of our final algorithm, we opted for the slower subroutine of Lemma 3.1, because it can be obtained from [5] in a black-box way.*

4 Overview: Our Dynamic Maximal Matching Algorithm

For ease of exposition, we first present the algorithm in a decremental setting, where the input graph only undergoes edge deletions. We will show how to handle fully dynamic updates without increasing update time at the end in Section 4.8.

The Framework. We fix two parameters $B \in [n]$ and $\epsilon \in (0, 1)$ whose values will be determined later on, and define

$$\delta := 100\epsilon. \tag{1}$$

The reader should think of $\epsilon = 1/n^{\beta}$ and $B = 1/\epsilon^2$ for some absolute constant $0 < \beta < 1$.

Let $G = (V, E)$ be the input graph with n nodes, undergoing edge deletions. We maintain a $(B, (1-\epsilon)B)$-EDCS $H := (V, E(H))$ of G at all times, as per Theorem 2.2. This incurs an update time of $O\left(\frac{n}{B\epsilon}\right)$. The EDCS H is maintained explicitly, and we can make adjacency-list and adjacency-matrix queries to H whenever we want. We maintain the adjacency-lists of all vertices in H as binary search trees. Hence, we may answer an adjacency-list and adjacency-matrix query to H in $O(\log n)$ and $O(1)$ time respectively.

Our dynamic algorithm works in **phases**, where each phase consists of δn consecutive updates in G. Let $G_{\texttt{init}} = (V, E_{\texttt{init}})$ and $H_{\texttt{init}} = (V, E(H_{\texttt{init}}))$ respectively denote the status of the input graph G and the EDCS H at the start of a given phase. Throughout the duration of the phase, we let $H_{\texttt{core}}$ denote the subgraph of $H_{\texttt{init}}$ restricted to the edges in G. Specifically, we define $H_{\texttt{core}} := (V, E(H_{\texttt{core}}))$, where $E(H_{\texttt{core}}) := E(H_{\texttt{init}}) \cap E$. Note that $H_{\texttt{core}}$ is a decremental graph within a phase, i.e., the only updates that can occur to $H_{\texttt{core}}$ are edge deletions.

Organization of the Overview. In Section 4.1, we present a classification of nodes based on their degrees in the EDCS H at the start of a phase, under one simplifying assumption (see Assumption 4.1). To highlight the main ideas, we first describe and analyze our algorithm under this assumption. Section 4.2 shows how to maintain a *base matching* $M_{\texttt{base}}$, which matches *all* the nodes of a certain type (that have large EDCS degrees). Next, Section 4.3 shows how to maintain a maximal matching in the subgraph of G induced by the nodes that are free under $M_{\texttt{base}}$. In Section 4.4, we put together the algorithms from Sections 4.2 and 4.3, and derive Theorem 1.1. Next, in Section 4.5, we explain how to deal with the scenario when the simplifying assumption we made earlier (see Assumption 4.1) does not hold. Section 4.6 sketches the proof outline of a key lemma, which allows us to maintain the matching $M_{\texttt{base}}$ efficiently in Section 4.2. Finally, Section 4.8 shows how to extend our algorithm so that it can deal with fully dynamic updates.

4.1 Classification of Nodes and a Simplifying Assumption

We will outline how to maintain a maximal matching $M_{\texttt{final}}$ in G during a phase. We start by making one simplifying assumption stated below. Towards the end of this section, we explain how to adapt our algorithm when this assumption does not hold (see Assumption 4.5).

Assumption 4.1.
For every node $v \in V$, either $\deg_{H_{\texttt{init}}}(v) > \left(\frac{1}{2}+\delta\right)B$ *or* $\deg_{H_{\texttt{init}}}(v) < \left(\frac{1}{2}-\delta\right)B$.

In other words, this establishes a small "degree-gap" within the EDCS H at the start of the phase, by asserting that there does *not* exist any node $v \in V$ with $\left(\frac{1}{2}-\delta\right)B \le \deg_{H_{\texttt{init}}}(v) \le \left(\frac{1}{2}+\delta\right)B$. We say that a node $v \in V$ is **very high** if $\deg_{H_{\texttt{init}}}(v) > \left(\frac{1}{2}+\delta\right)B$, and **low** if $\deg_{H_{\texttt{init}}}(v) < \left(\frac{1}{2}-\delta\right)B$. We let $V_{\texttt{v-hi}}$ and $V_{\texttt{lo}}$ respectively denote the sets of very high and low nodes. By Assumption 4.1, the node-set V is partitioned into $V_{\texttt{v-hi}} \subseteq V$ and $V_{\texttt{lo}} = V \setminus V_{\texttt{v-hi}}$. Our algorithm will crucially use two properties that arise out of this classification of nodes, as described below.

Proposition 4.2. *The following properties hold.*
(1) *Consider any edge $(u, v) \in E(H_{\texttt{init}})$ where u is very-high. Then v must be low.*
(2) *Consider any edge $e \in E_{\texttt{init}}$ whose both endpoints are low. Then e must appear in $H_{\texttt{init}}$.*

Proof. First, we prove Item 1. Note that since $(u, v) \in E(H_{\texttt{init}})$, we have $\deg_{H_{\texttt{init}}}(u) + \deg_{H_{\texttt{init}}}(v) \le B$ (see Definition 2.1). Furthermore, since u is very-high, we have $\deg_{H_{\texttt{init}}}(u) > \left(\frac{1}{2}+\delta\right)B$. Thus, it follows that $\deg_{H_{\texttt{init}}}(v) \le B - \deg_{H_{\texttt{init}}}(u) < \left(\frac{1}{2}-\delta\right)B$, and so v must be a low node.

Next, we prove Item 2. Let $e = (u, v)$, where both u, v are low nodes. Then $\deg_{H_{\texttt{init}}}(u) < \left(\frac{1}{2}-\delta\right)B$ and $\deg_{H_{\texttt{init}}}(v) < \left(\frac{1}{2}-\delta\right)B$, and hence $\deg_{H_{\texttt{init}}}(u) + \deg_{H_{\texttt{init}}}(v) < (1-2\delta)B < (1-\epsilon)B$. As $H_{\texttt{init}}$ is a $(B, (1-\epsilon)B)$-EDCS of $G_{\texttt{init}}$, it follows that $(u, v) \in E(H_{\texttt{init}})$ (see Definition 2.1). □

We next introduce one last category of nodes, as defined below.

Definition 4.3. *Consider a very high node $v \in V_{\texttt{v-hi}}$. At any point in time within the current phase, we say that v is* **damaged** *if* $\deg_{H_{\texttt{core}}}(v) < \left(\frac{1}{2}+\delta-2\epsilon\right)B$, *and* **safe** *otherwise. We let $V_{\texttt{dmg}} \subseteq V_{\texttt{v-hi}}$ and $V_{\texttt{sf}} := V_{\texttt{v-hi}} \setminus V_{\texttt{dmg}}$ respectively denote the sets of damaged and safe nodes.*

Recall that $\deg_{H_{\texttt{init}}}(v) > \left(\frac{1}{2}+\delta\right)B$ for all nodes $v \in V_{\texttt{v-hi}}$. Thus, for such a node to get damaged at least $2\epsilon B$ many of its incident edges must be deleted since the start of the phase. Since each phase lasts for δn updates in G and $\delta = 100\epsilon$ (see (1)), we get the following important corollary.

Corollary 4.4. *We always have $\left|V_{\texttt{dmg}}\right| \le \frac{2\delta n}{2\epsilon B} = O\left(\frac{n}{B}\right)$. Furthermore, at the start of a phase we have $V_{\texttt{dmg}} = \emptyset$. During the phase, the subset $V_{\texttt{dmg}}$ grows monotonically over time.*

4.2 Maintaining a Matching of All the Safe Nodes

One of our key technical insights is that the degree gap in $H_{\texttt{core}}$ allows us to efficiently maintain a matching $M_{\texttt{base}} \subseteq E(H_{\texttt{core}})$ that matches <u>all</u> safe nodes, as per Lemma 4.5 below. With appropriate setting of parameters, the update time of both of the algorithms (deterministic and randomized) in Lemma 4.5 are sublinear in n.

Lemma 4.5. *We can maintain a matching $M_{\texttt{base}} \subseteq E(H_{\texttt{core}})$ that satisfy the following properties.*
(1) *Every safe node is matched under $M_{\texttt{base}}$, i.e., $V_{\texttt{sf}} \subseteq V(M_{\texttt{base}})$.*
(2) *Every update in G that is internal to a phase (i.e., excluding those updates where one phase ends and the next phase begins) leads to at most $O(1)$ node insertions/deletions in $V(M_{\texttt{base}})$.*

The matching $M_{\texttt{base}}$ can be maintained either by

- *a deterministic algorithm with $\tilde{O}\left(B \cdot n^{1/2} \cdot \delta^{-3/2}\right)$ amortized update time; or*

- *a randomized algorithm with $\tilde{O}\left(B \cdot \delta^{-1} + \delta^{-3}\right)$ amortized update time. The algorithm is correct with high probability against an adaptive adversary.*

Intuition Aside: The only property of H_{core} that we need for Lemma Lemma 4.5 is the degree gap between V_{sf} and $V_{\mathtt{lo}}$; in fact, the whole reason we use an EDCS in the first place is to ensure this degree gap.

We defer the proof sketch of Lemma 4.5 to Section 4.6, but to provide an intuition for why the degree gap helps match vertices in V_{sf}, we show here that it is easy to deterministically maintain a *fractional matching* satisfying the concerned properties of Lemma 4.5, in $O(B/\delta)$ update time. We do not this result in the main body of our paper and include it only for intuition. We denote the fractional matching by $x_{\mathsf{base}} : E(H_{\mathsf{core}}) \to [0,1]$, which is constructed as follows.

- $\Gamma \leftarrow \left(\frac{1}{2} + \delta - 2\epsilon\right) B$ and $x_{\mathsf{base}}(e) \leftarrow 0$ for all $e \in E$
- For every safe node $v \in V_{\mathsf{sf}}$
  - Let $E^{\star}_v \subseteq E(H_{\mathsf{core}})$ be a set of Γ distinct edges incident to v. ($E^{\star}_v$ exists as $v \in V_{\mathsf{sf}}$).
  - $x_{\mathsf{base}}(e) \leftarrow 1/\Gamma$ for all $e \in E^{\star}_v$
- Return $x_{\mathsf{base}} : E(H_{\mathsf{core}}) \to [0,1]$.

Consider any two distinct nodes $v, v' \in V_{\mathsf{sf}} \subseteq V_{\mathtt{v\text{-}hi}}$. By Item 1 of Proposition 4.2, there cannot be an edge $(v,v') \in E(H_{\mathsf{core}}) \subseteq E(H_{\mathtt{init}})$, and hence $E^{\star}_v \cap E^{\star}_{v'} = \emptyset$. Using this observation, it is easy to verify that the function x_{base} is a valid fractional matching in H_{core}, and that every node $v \in V_{\mathsf{sf}}$ receives a weight $x_{\mathsf{base}}(v) = 1$ under this fractional matching.[4] Furthermore, since the EDCS $H_{\mathtt{init}}$ has maximum degree at most B (this follows from Definition 2.1) and each phase lasts for δn updates, we can deterministically maintain the fractional matching x_{base} in $O\left(\frac{nB}{\delta n}\right) = O(B/\delta)$ amortized update time. One can also verify that each update in G that is internal to a phase leads to at most $O(1)$ nodes $v \in V$ changing their weights $x_{\mathsf{base}}(v)$ under x_{base}.

4.3 Maintaining a Maximal Matching in the Adjunct Subgraph

Let $G_{\mathtt{adj}} := (V_{\mathtt{adj}}, E_{\mathtt{adj}})$ denote the subgraph of G induced by the set of nodes that are unmatched under M_{base}, that is, $V_{\mathtt{adj}} := V \setminus V(M_{\mathsf{base}})$ and $E_{\mathtt{adj}} := \{(u,v) \in E : u, v \in V_{\mathtt{adj}}\}$. We will refer to $G_{\mathtt{adj}}$ as the **adjunct subgraph** of G w.r.t. M_{base}. By the following observation, all that remains is to maintain a maximal matching in $G_{\mathtt{adj}}$

Observation 4.6. *If $M_{\mathtt{adj}}$ is a maximal matching of $G_{\mathtt{adj}}$, then $M_{\mathtt{final}} := M_{\mathsf{base}} \cup M_{\mathtt{adj}}$ is a maximal matching in $G = (V, E)$*

We now show how to efficiently maintain a maximal matching $M_{\mathtt{adj}}$ of $G_{\mathtt{adj}}$ (we refer to it as the **adjunct matching**).

[4]The weight $x_{\mathsf{base}}(v)$ is the sum of the values $x_{\mathsf{base}}(e)$, over all the edges in H_{core} that are incident on v.

Lemma 4.7. *Suppose that we maintain the matching M_{base} as per Lemma 4.5. Then with an additive overhead of $\tilde{O}\left(\frac{B}{\delta} + \frac{n}{\delta B}\right)$ amortized update time, we can deterministically and explicitly maintain a* maximal *matching $M_{\mathtt{adj}} \subseteq E_{\mathtt{adj}}$ in $G_{\mathtt{adj}}$.*

At this point, the reader might get alarmed because $G_{\mathtt{adj}}$ undergoes *node-updates* every time an edge gets added to/removed from M_{base}, which might indicate that it is impossible to efficiently maintain a maximal matching in $G_{\mathtt{adj}}$. To assuage this concern, we emphasize that: (i) the *arboricity* of $G_{\mathtt{adj}}$ is sublinear in n (see Claim 4.8), and (ii) an edge update in G that is internal to a phase leads to at most $O(1)$ node-updates in $G_{\mathtt{adj}}$ (see Claim 4.9). These two properties ensure that we can maintain the matching $M_{\mathtt{adj}}$ in sublinear in n update time.

Claim 4.8. *The subgraph $G_{\mathtt{adj}} = (V, E_{\mathtt{adj}})$ satisfies the following properties.*

(1) *$V_{\mathsf{sf}} \cap V_{\mathtt{adj}} = \emptyset$ and $\left|V_{\mathtt{dmg}} \cap V_{\mathtt{adj}}\right| = O\left(\frac{n}{B}\right)$. Furthermore, $V_{\mathtt{dmg}} \cap V_{\mathtt{adj}} = \emptyset$ at the start of a phase.*
(2) *$\deg_{G_{\mathtt{adj}}}(v) \leq B + \left|V_{\mathtt{dmg}} \cap V_{\mathtt{adj}}\right|$ for all nodes $v \in V_{\mathtt{lo}}$.*

Proof. Since $V_{\mathtt{adj}} := V \setminus V(M_{\mathsf{base}})$, Item 1 follows from Item 1 of Lemma 4.5, and Corollary 4.4.

To prove Item 2, consider any node $v \in V_{\mathtt{lo}}$. Let $u \in V_{\mathtt{lo}}$ be any low neighbor of v in $E_{\mathtt{adj}} \subseteq E \subseteq E_{\mathtt{init}}$. Then, by Item 2 of Proposition 4.2, we have $(u,v) \in H_{\mathtt{init}}$. Accordingly, the number of low neighbors of v in $G_{\mathtt{adj}}$ is at most $\deg_{H_{\mathtt{init}}}(v) \leq B$, where the last inequality holds because $H_{\mathtt{init}}$ is a $(B, (1-\epsilon)B)$-EDCS of $G_{\mathtt{init}}$ (see Definition 2.1). As the node-set V is partitioned into $V_{\mathtt{lo}}, V_{\mathtt{v\text{-}hi}}$ (see Assumption 4.1) and $V_{\mathtt{v\text{-}hi}}$ is further partitioned into $V_{\mathsf{sf}}, V_{\mathtt{dmg}}$, we get:

$$\deg_{G_{\mathtt{adj}}}(v) \leq B + \left|V_{\mathsf{sf}} \cap V_{\mathtt{adj}}\right| + \left|V_{\mathtt{dmg}} \cap V_{\mathtt{adj}}\right| = B + \left|V_{\mathtt{dmg}} \cap V_{\mathtt{adj}}\right|,$$

where the last equality follows from Item 1. □

Claim 4.9. *An edge-update in G internal to a phase leads to at most $O(1)$ node-updates in $G_{\mathtt{adj}}$.*

Proof. Since $V_{\mathtt{adj}} := V \setminus V(M_{\mathsf{base}})$, the claim follows from Item 2 of Lemma 4.5. □

Corollary 4.10. *At the start of a phase, we have $|E_{\mathtt{adj}}| = O(Bn)$.*

Proof. By Proposition 4.2(2) we have that $E_{\mathtt{adj}} \subseteq H_{\mathtt{init}}$. As $H_{\mathtt{init}}$ is a $(B, B(1-\epsilon))$-EDCS and the maximum vertex degree of a $(B, B(1-\epsilon))$-EDCS is B we must have that $|E_{\mathtt{adj}}| \leq |H_{\mathtt{init}}| = O(nB)$.

□

Proof of Lemma 4.7. At the start of a phase, we initialize $M_{\mathtt{adj}}$ to be any arbitrary maximal matching in $G_{\mathtt{adj}}$. By Corollary 4.10, this takes $O(Bn)$ time. Since the phase lasts for δn updates in G, this step incurs an amortized update time of:

$$O\left(\frac{Bn}{\delta n}\right) = O\left(\frac{B}{\delta}\right). \tag{2}$$

Subsequently, to handle the updates during a phase, we use the following auxiliary data structures: Each node $v \in V_{\mathtt{dmg}}$ maintains the set $\mathcal{F}_{\mathtt{adj}}(v) := \left\{u \in V_{\mathtt{adj}} \setminus V(M_{\mathtt{adj}}) : (u,v) \in E_{\mathtt{adj}}\right\}$ of its free

neighbors (in G_{adj}) under the matching M_{adj}.[5] Whenever a node v moves from V_{sf} to V_{dmg}, we spend $\tilde{O}(n)$ time to initialize the set $\mathcal{F}_{\mathsf{adj}}(v)$ as a balanced search tree. Within a phase, a node can move from V_{sf} to V_{dmg} at most once (see Definition 4.3 and Corollary 4.4). Thus, by Item 1 of Claim 4.8, these initializations take $\tilde{O}\left(\frac{n}{B} \cdot n\right) = \tilde{O}\left(\frac{n^2}{B}\right)$ total time during a phase. As each phase lasts for δn updates in G, this incurs an amortized update time of $\tilde{O}\left(\frac{n^2}{B \cdot \delta n}\right) = \tilde{O}\left(\frac{n}{\delta B}\right)$.

Claim 4.9 guarantees that each edge update in G leads to $O(1)$ node-updates in G_{adj}. We now show that to maintain the maximal matching M_{adj}, a node-update to G_{adj} can be handled in $\tilde{O}\left(B + \frac{n}{B}\right)$ worst-case time. If a new vertex v enters G_{adj} (because v became unmatched in M_{base}), then to maintain the maximality of M_{adj}, the algorithm needs to find a free neighbor of v (if one exists). The same needs to be done if x leaves G_{adj}, and x was previously matched to v in M_{adj}. So to handle a node-update in G_{adj} we need a subroutine that takes in a vertex v and either finds a M_{adj}-free neighbor of v in G_{adj} or certifies that none exists.

The above subroutine can be performed in $\tilde{O}(1)$ time if $v \in V_{\mathsf{dmg}}$ (using the set $\mathcal{F}_{\mathsf{adj}}(v)$); if $v \in V_{\mathsf{adj}} \setminus V_{\mathsf{dmg}}$ then by Item 1 of Claim 4.8 $v \in V_{\mathtt{lo}}$, so by Item 2 of the same claim we can perform the subroutine in time $\deg_{G_{\mathsf{adj}}}(v) = O\left(B + \frac{n}{B}\right)$. Further, once we decide to match a node $u \in V_{\mathsf{adj}}$ to one of its free neighbors, we can update all the relevant sets $\mathcal{F}_{\mathsf{adj}}(v)$ in $\tilde{O}\left(\left|V_{\mathsf{dmg}}\right|\right) = \tilde{O}\left(\frac{n}{B}\right)$ time (see Claim 4.8).

To summarize, the amortized update of maintaining the matching M_{adj} under adversarial updates in G is at most:

$$\tilde{O}\left(\frac{n}{\delta B} + B + \frac{n}{B}\right) = \tilde{O}\left(B + \frac{n}{\delta B}\right). \tag{3}$$

Lemma 4.7 follows from (2) and (3).

4.4 Putting Everything Together: How to Derive Theorem 1.1

Since $E(H_{\mathsf{core}}) \subseteq E$, it follows that $M_{\mathsf{base}} \subseteq E(H_{\mathsf{core}})$ is a valid matching in G (see Lemma 4.5). Furthermore, Lemma 4.7 guarantees that M_{adj} is a *maximal* matching in G_{adj} Thus, by Observation 4.6, the matching $M_{\mathtt{final}} := M_{\mathsf{base}} \cup M_{\mathsf{adj}}$ is a maximal matching in G. It now remains to analyze the overall amortized update time of our algorithm.

Recall that maintaining the EDCS H incurs an update time of $O\left(\frac{n}{\epsilon B}\right) = O\left(\frac{n}{\delta B}\right)$, according to (1). Now, for the deterministic algorithm, Lemma 4.5 incurs an amortized update time of $\tilde{O}\left(B \cdot n^{1/2} \cdot \delta^{-3/2}\right)$. Thus, by Lemma 4.7, the overall amortized update time becomes

$$O\left(\frac{n}{\delta B}\right) + \tilde{O}\left(B \cdot n^{1/2} \cdot \delta^{-3/2}\right) + \tilde{O}\left(\frac{B}{\delta} + \frac{n}{\delta B}\right) = \tilde{O}(n^{8/9})$$

$$\text{for } \delta = \frac{1}{n^{1/9}} \text{ and } B = n^{2/9}.$$

For the randomized algorithm against an adaptive adversary, Lemma 4.5 incurs an amortized update time of $\tilde{O}(B\delta^{-1} + \delta^{-3})$. Thus, by Lemma 4.7, the overall amortized update time becomes

$$O\left(\frac{n}{\delta B}\right) + \tilde{O}\left(\frac{B}{\delta} + \frac{1}{\delta^3}\right) + \tilde{O}\left(\frac{B}{\delta} + \frac{n}{\delta B}\right) = \tilde{O}\left(n^{3/4}\right)$$

$$\text{for } \delta = \frac{1}{n^{1/4}} \text{ and } B = n^{1/2}.$$

This leads us to Theorem 1.1 in the decremental setting.

4.5 Dealing with Medium Nodes: Getting Rid of Assumption 4.1

We now provide a high-level outline of our algorithm in the general case, when Assumption 4.1 does not hold. Recall that under Assumption 4.1 there cannot be any node v with $\left(\frac{1}{2} - \delta\right) B \le \deg_{H_{\mathtt{init}}}(v) \le \left(\frac{1}{2} + \delta\right) B$. Conceptually, the most significant challenge here is to deal with the set of **medium nodes**, given by:

$$V_{\mathsf{med}} := \left\{ v \in V : \left(\frac{1}{2} - \delta\right) B \le \deg_{H_{\mathtt{init}}}(v) < \left(\frac{1}{2} + \delta - \epsilon\right) B \right\}.$$

For the rest of this overview, we replace Assumption 4.1 with the following weaker assumption

Assumption 4.11 (Replaces Assumption 4.1). *All vertices are in $V_{\mathtt{lo}}$, V_{med}, or $V_{\mathtt{v\text{-}hi}}$*

In the main body of the paper we also have to handle the nodes that are ruled out by the above assumption, namely the ones whose degrees in $H_{\mathtt{init}}$ lie in the range $\left[\left(\frac{1}{2} + \delta - \epsilon\right) B, \left(\frac{1}{2} + \delta\right) B\right]$. These remaining nodes lie in an extremely narrow range, so while they require some technical massaging, they do not pose any additional conceptual difficulties.

We now show how to adapt our algorithm to also handle the medium nodes. Lemma 4.12, stated below, generalizes Lemma 4.5, and serves as a key building block of our algorithm. The only difference between these two lemmas is that in Item 1 of Lemma 4.12, we allow for $O(\delta n)$ medium nodes that are free under M_{base}. Note that sets V_{sf} and V_{dmg} are defined the same as before (Definition 4.3), and are in particular both subsets of $V_{\mathtt{v\text{-}hi}}$.

Lemma 4.12. *We can maintain a matching $M_{\mathsf{base}} \subseteq E(H_{\mathsf{core}})$ that satisfy the following properties.*

1. *Every safe node is matched under M_{base}, i.e., $V_{\mathsf{sf}} \subseteq V(M_{\mathsf{base}})$. Furthermore, at most $O(\delta n)$ medium nodes are free under M_{base}, i.e., $|V_{\mathsf{med}} \setminus V(M_{\mathsf{base}})| = O(\delta n)$.*
2. *Every update in G that is internal to a phase (i.e., excluding those updates where one phase ends and the next phase begins) leads to at most $O(1)$ node insertions/deletions in $V(M_{\mathsf{base}})$.*

The matching M_{base} can be maintained

- *either by a deterministic algorithm with $\tilde{O}\left(B \cdot n^{1/2} \cdot \delta^{-3/2}\right)$ amortized update time; or*
- *by a randomized algorithm with $\tilde{O}\left(B \cdot \delta^{-1} + \delta^{-3}\right)$ amortized update time. The algorithm is correct with high probability against an adaptive adversary.*

Before outlining the proof sketch of Lemma 4.12, we explain how to adapt the algorithm for maintaining a maximal matching $M_{\mathsf{adj}} \subseteq E_{\mathsf{adj}}$ in the adjunct graph $G_{\mathsf{adj}} := (V_{\mathsf{adj}}, E_{\mathsf{adj}})$, which is

[5] Maximality of M_{adj} implies that $\mathcal{F}_{\mathsf{adj}}(v) = \emptyset$ for all $v \in V_{\mathsf{dmg}} \setminus V(M_{\mathsf{adj}})$.

the subgraph of G induced by the node-set $V_{\mathtt{adj}} := V \setminus V(M_{\mathtt{base}})$ (see Section 4.3). Lemma 4.13, stated below, generalizes Lemma 4.7 in the presence of medium nodes.

Lemma 4.13. *Suppose that we maintain the matching $M_{\mathtt{base}}$ as per Lemma 4.12. Then with an additive overhead of $\tilde{O}\left(\delta n + \frac{B}{\delta} + \frac{n}{\delta B}\right)$ amortized update time, we can deterministically and explicitly maintain a* maximal *matching $M_{\mathtt{adj}} \subseteq E_{\mathtt{adj}}$ in $G_{\mathtt{adj}}$.*

Proof Sketch. The only difference with the proof of Lemma 4.7 is that now the set $V_{\mathtt{adj}}$ can contain some medium nodes, although $|V_{\mathtt{adj}} \cap V_{\mathtt{med}}| = O(\delta n)$ as per Item 1 of Lemma 4.12.

We explicitly maintain the set $V_{\mathtt{adj}} \cap V_{\mathtt{med}}$ as a doubly linked list. Whenever a node $v \in V_{\mathtt{lo}} \cup V_{\mathtt{dmg}}$ is searching for a free neighbor (in the proof of Lemma 4.7), we ensure that it additionally scans the set $V_{\mathtt{adj}} \cap V_{\mathtt{med}}$. This leads to an additive overhead of $|V_{\mathtt{med}} \cap V_{\mathtt{adj}}| = O(\delta n)$ in the update time.

Next, recall that $H_{\mathtt{init}}$ is a $(B, (1-\epsilon)B)$-EDCS of $G_{\mathtt{init}}$. Consider any edge $(u, v) \in E_{\mathtt{adj}} \subseteq E \subseteq E_{\mathtt{init}}$ with $v \in V_{\mathtt{med}}$ and $u \in V_{\mathtt{lo}}$. Then, we have $\deg_{H_{\mathtt{init}}}(u) + \deg_{H_{\mathtt{init}}}(v) < \left(\frac{1}{2} - \delta\right)B + \left(\frac{1}{2} + \delta - \epsilon\right)B = (1-\epsilon)B$, and hence Definition 2.1 implies that $(u, v) \in E(H_{\mathtt{init}})$. In other words, all the edges in $G_{\mathtt{adj}}$ that connect a low node with a medium node belong to the EDCS $H_{\mathtt{init}}$, which in turn has maximum degree at most B (see Definition 2.1). Thus, every medium node has at most B low neighbors in $G_{\mathtt{adj}}$.

Now, suppose that we are searching for a free neighbor of a node $v \in V_{\mathtt{med}}$ in $G_{\mathtt{adj}}$. This step involves scanning through: (i) all the low neighbors of v in $G_{\mathtt{adj}}$ (and by the previous discussion there are at most B such neighbors), (ii) the set $V_{\mathtt{med}} \cap V_{\mathtt{adj}}$ of size $O(\delta n)$, and (iii) and at most $O\left(\frac{n}{B}\right)$ damaged nodes $V_{\mathtt{dmg}} \cap V_{\mathtt{adj}}$ (see Section 4.3). Overall, this scan takes $O\left(\delta n + B + \frac{n}{B}\right)$ time, and just as before, contributes an additive overhead of $O(\delta n)$ to the update time in Lemma 4.7.

Everything else remains the same as in the proof of Lemma 4.7. □

Comparing Lemmas 4.12 and 4.13 against Lemmas 4.5 and 4.7, we conclude that there is an additive overhead of $O(\delta n)$ update time while dealing with the medium nodes. Finally, from the analysis from Section 4.4, it is easy to verify that this overhead of $O(\delta n)$ does *not* degrade our overall asymptotic update time (the values of δ, B remain the same as in Section 4.4).

Proof Sketch of Lemma 4.12. Consider the scenario at the start of a new phase (just before the first update within that phase). At this point, we first compute a subgraph $H'_{\mathtt{init}} := (V, E(H'_{\mathtt{init}}))$ of $H_{\mathtt{init}}$, as follows. We initialize $H'_{\mathtt{init}} \leftarrow H_{\mathtt{init}}$. Subsequently, for every node $v \in V_{\mathtt{v\text{-}hi}}$, we keep deleting edges $(u, v) \in E(H'_{\mathtt{init}})$ incident on v from $H'_{\mathtt{init}}$, until $\deg_{H'_{\mathtt{init}}}(v)$ becomes equal to $\Delta := \left(\frac{1}{2} + \delta\right)B$. By Item 1 of Proposition 4.2, every edge (u, v) that gets deleted from $H'_{\mathtt{init}}$ has its other endpoint $u \in V_{\mathtt{lo}}$. Accordingly, this process does not reduce the degree of any medium node. When this for loop terminates, we have $\left(\frac{1}{2} - \delta\right)B \leq \deg_{H'_{\mathtt{init}}}(v) \leq \Delta$ for all $v \in V_{\mathtt{med}} \cup V_{\mathtt{v\text{-}hi}}$, and $\deg_{H'_{\mathtt{init}}}(v) < \left(\frac{1}{2} - \delta\right)B$ for all $v \in V_{\mathtt{lo}}$. In other words, the degree of every node $v \in V_{\mathtt{med}} \cup V_{\mathtt{v\text{-}hi}}$ in $H'_{\mathtt{init}}$ is at least $(1 - \Theta(\delta))$ times the maximum degree Δ. Now, we apply Lemma 3.1, with $\kappa = \Theta(\delta)$, to compute a matching $M \subseteq E(H'_{\mathtt{init}}) \subseteq E(H_{\mathtt{init}})$ that matches all but $O(\delta n)$ nodes in $V_{\mathtt{med}} \cup V_{\mathtt{v\text{-}hi}}$. Note that $\Delta = \Theta(B)$, and that a phase lasts for δn updates in G. Thus, at the start of a phase, computing the subgraph $H'_{\mathtt{init}}$ and the invocation of Lemma 3.1 incurs an amortized update time of

$$\tilde{O}\left(\frac{nB + n\Delta/\delta}{\delta n}\right) = \tilde{O}\left(\frac{B}{\delta^2}\right)$$

if we use the deterministic algorithm, and an amortized update time of

$$\tilde{O}\left(\frac{nB + n\Delta\delta}{\delta n}\right) = \tilde{O}\left(\frac{B}{\delta}\right)$$

if we use the randomized algorithm. It is easy to verify that both these terms get subsumed within the respective (deterministic or randomized) guarantees of Lemma 4.12.

We have thus shown to initialize the desired matching $M_{\mathtt{base}} = M$ at the beginning of the phase. We now give a sketch of how we maintain $M_{\mathtt{base}}$ dynamically throughout the phase. Define $H_{\mathtt{hilo}}$ to be the induced graph $H_{\mathtt{core}}[V_{\mathtt{v\text{-}hi}} \cup V_{\mathtt{lo}}]$. Recall that all edges in $E_{H_{\mathtt{core}}}(V_{\mathtt{v\text{-}hi}})$ have their other endpoint in $V_{\mathtt{lo}}$ (Item 1 of Proposition 4.2), so the set of edges incident to $V_{\mathtt{v\text{-}hi}}$ is exactly the same in $E_{H_{\mathtt{hilo}}}$ and $E_{H_{\mathtt{core}}}$. We are thus able to apply Lemma 4.5 exactly as before to maintain a matching $M_{\mathtt{hilo}}$ in $H_{\mathtt{hilo}}$ that matches all of $V_{\mathtt{sf}}$. We will maintain the invariant that $M_{\mathtt{hilo}} \subseteq M_{\mathtt{base}}$, thus guaranteeing that all nodes in $V_{\mathtt{sf}}$ remain matched throughout the phase. The only remaining concern is that the algorithm for maintaining $M_{\mathtt{hilo}}$ might end up unmatching medium vertices in $M_{\mathtt{base}}$: if vertex $u \in V_{\mathtt{hi}}$ becomes unmatched, then to maintain $M_{\mathtt{hilo}}$ the algorithm from Lemma 4.5 finds an augmenting path from u to some $v \in V_{\mathtt{lo}}$ that was free in $M_{\mathtt{hilo}}$; but although v was free in $M_{\mathtt{hilo}}$, it might have been matched to some medium node x in $M_{\mathtt{base}}$, in which case node x now becomes unmatched. Fortunately, it is not hard to show that every adversarial update creates only $O(1)$ free vertices, so since $M_{\mathtt{base}}$ has $O(\delta n)$ free medium vertices at the beginning of the phase, and since each phase lasts for $O(\delta n)$ updates, we maintain the desired property that there are always $O(\delta n)$ free medium vertices.

4.6 Matching All the Safe Nodes: Proof Outline for Lemma 4.5

The key to our proof is an algorithm for maintaining a left-perfect matching in a bipartite graph with a degree gap.

Lemma 4.14 (Simplified version of Lemma 5.1). *Let $G = (L \cup R, E)$ be a bipartite graph where, for some $X > 0$ and $0 < \gamma < 1$, every vertex $v \in L$ has $2X \geq \deg(v) \geq X$ and every vertex in R has degree $\leq X(1-\gamma)$. Suppose that the above degree constraints always hold as G is subject to $O(\delta n)$ decremental updates. Then, there exists an algorithm that maintains a left-perfect matching in G – i.e. a matching in which every vertex in L is matched – with the following update times:*

- *A randomized algorithm with worst-case update time $\tilde{O}\left(X + 1/\gamma^3\right)$.*
- *A deterministic algorithm with* total *update time $\tilde{O}\left(\frac{nX}{\gamma} + \frac{n^{1.5}X\delta}{\gamma^{1.5}}\right)$.*

Before proving the above Lemma, let us see why it implies Lemma 4.5

Proof sketch of Lemma 4.5. For this proof sketch, we assume that we always have $V_{\texttt{v-hi}} = V_{\texttt{sf}}$; in the full version, we are able to effectively ignore damaged nodes because Lemma 4.5 only requires us to match all the nodes of $V_{\texttt{sf}}$. Let $H_{\texttt{hilo}}$ be the subgraph of $H_{\texttt{core}}$ that contains only the edges incident to $V_{\texttt{v-hi}}$. Observe that under our assumption of $V_{\texttt{v-hi}} = V_{\texttt{sf}}$, $H_{\texttt{hilo}}$ satisfies the properties of Lemma 4.14 with $X = B/2$ and $\gamma \sim \delta$: note in particular that the first property of an EDCS (Definition 2.1) ensures that every vertex has degree $\leq B = 2X$, and that $H_{\texttt{hilo}}$ is bipartite because by Proposition 4.2, $H_{\texttt{init}}$ contains no edges between vertices in $V_{\texttt{v-hi}}$. We can thus apply Lemma 4.14 to maintain a matching that matches all of $V_{\texttt{sf}}$, as desired. The update-time bounds of Lemma 4.14 precisely imply those of Lemma 4.5

□

4.7 Maintaining Left-Perfect Matching: Proof of Lemma 4.14

Randomized Algorithm. The randomized algorithm is extremely simple. Say that we have a matching M under which some $v \in L$ is still unmatched. We then find an augmenting path from v to an M-free node in R by taking a *random matching-walk* from v. That is, we follow a random edge (v, x) to R; if x is unmatched then we have successfully found an augmenting path, while if x is matched to some $y \in L$ then we repeat the process from y.

We are able to show that because of the assumed degree gap of G, this random matching-walk terminates within $\tilde{O}(1/\gamma^3)$ steps with high probability. The intuitive reason is as follows: because of the degree gap, every set $S \subset L$ as at least $S(1+\gamma)$ neighbors in R. The graph is thus effectively a γ-expander, and in particular we are able to show a correspondence between random matching-walks in G and ordinary random walks in some Eulerian directed γ-expander G'; the bound of $\tilde{O}(1/\gamma^3)$ then follows from known facts about Eulerian expanders.

To initialize the a left perfect matching before the first decremental update occurs we apply the randomized algorithm of Lemma 3.1 with $\Delta = X, \kappa = \gamma$. This returns a matching covering all but $O(n\gamma)$ vertices of L in time $\tilde{O}(nX\gamma)$ adding an amortized update time component of $\tilde{O}(X)$ over the deletions. We may match each of the $O(n\gamma)$ unmatched vertices using the same random walk based technique as we use to handle deletions in time $\tilde{O}(n\gamma^{-2})$ adding an amortized update time component of $\tilde{O}(\gamma^{-3})$ over the $O(n\gamma)$ deletions.

Deterministic Algorithm. The deterministic algorithm is more involved than the randomized one, so we only give a high-level sketch. We start by maintaining a standard directed residual graph G_{res} corresponding to the matching M that we maintain (edges in $E \setminus M$ point from L to R in G_{res}, while edges in M point from R to L). We also create a dummy sink t in G_{res}, with an edge from all M-free vertices in R to t. It is not hard to check that there is a correspondence between paths from L to t in G_{res} and augmenting paths in G. Our algorithm maintains a shortest path tree T_{res} to sink t in G_{res}. Given any M-free free vertex $v \in L$, we can find an augmenting path from v by simply following the $v - t$ path in T_{res}.

The primary challenge is to maintain T_{res}. It is not hard to prove that for any M-free $v \in L$, there always *exists* an augmenting path of length $\tilde{O}(1/\gamma)$ from v to an M-free vertex in R; this follows from the same expander-based arguments as above, and can also be proved using more elementary techniques. This means that the depth of T_{res} is bounded by $\tilde{O}(1/\gamma)$, which makes it easier to maintain.

There exists a classic data structure known as an *Even and Shiloach tree* (denoted ES-tree) that can maintain a low-depth shortest path tree to a sink t in a decremental graph [101]. The ES-tree can also handle a special kind of edge insertion into G – namely, an edge insertion is a valid update if it does not decrease any distances to t. If G_{res} were a decremental graph, then we could us an ES-tree to maintain T_{res}, and we would be done. Unfortunately, even though the input graph G to Lemma 4.14 only undergoes edge deletions, the graph G_{res} can undergo edge insertions for two reasons.

Firstly, whenever we augment down a path P, every edge in P flips whether or not it is in M, which means we need to flip the directions of all these edges in G_{res}. We can model this flipping as an inserting of all edges on the reverse path $\overleftarrow{P}$, followed by a deletion of all edges on the original path P. Fortunately, it is not hard to show that because T_{res} is a shortest-path tree, inserting $\overleftarrow{P}$ does not decrease any distances (as long as we so before deleting P), and hence is a valid update to the ES-tree.

It is the second kind of edge insertion to G_{res} that poses a significant problem. Say that the adversary deletes some matching edge (u, v) in G. Then v becomes free, so the graph G_{res} should now contain an edge (v, t); but simply inserting this edge into G_{res} would not be a valid update to the ES-tree, as it would decrease $\text{dist}(v, t)$. To overcome this issue, we observe that before the deletion of edge (u, v) we had $\text{dist}_{G_{\text{res}}}(v, t) = \tilde{O}(1/\gamma)$, because we could follow the matching edge from v to u, and as discussed above we know that there exists an augmenting path from u to a free vertex of length $\tilde{O}(1/\gamma)$. So inserting an edge (v, t) of weight $\tilde{O}(1/\gamma)$ does not decrease any distances to t and is hence a valid operation for the ES-tree. Unfortunately, once we start adding weights to edges to of G_{res}, the distances in G_{res} will get longer and longer, so the next time we add an edge (v', t) it might need to have even larger weight for this to be a valid insertion into the ES-tree. (In particular, the presence of edge weights means that we can no longer guarantee that before the insertion of (v', t) we had $\text{dist}(v', t) = \tilde{O}(1/\gamma)$.) The distances in G_{res} can thus blow up, which increases the (weighted) depth of T_{res} and hence causes the ES-tree to become inefficient.

To overcome the above issue, we define parameter $q = \sqrt{n\gamma}$ and use structural properties of degree-gap graphs to show that as long as $\leq q$ insertions have occurred, the edge weights have not had a chance to get too large, and the *average* $\text{dist}(v, t)$ remains small, so the ES-tree remains efficient. Then, after q updates, we simply rebuild the entire ES tree from scratch. Since the total number of resets is only $O(n\delta/q)$, the total update time ends up being not too large.

To initialize a left perfect matching we will use the deterministic algorithm of Lemma 3.1 which returns a matching covering all but $O(n\gamma)$ vertices of L in time $\tilde{O}(nX/\gamma)$. We augment this imperfect matching at initialization to be a perfect matching using the same datastructure as we use to handle edge deletions. This results in a

running time proportional to handling $O(n\gamma)$ deletions matching the cost of handling the decremental updates.

4.8 Extension to the Fully Dynamic Setting

It is relatively straightforward to extend the algorithm from Section 4 to a setting where the input graph $G = (V, E)$ undergoes both edge insertions and edge deletions. As in Section 4, our fully dynamic algorithm works in phases, where each phase lasts for δn updates in G.

We maintain two subgraphs $G_{\mathsf{new}} := (V, E_{\mathsf{new}})$ and $G_{\mathsf{core}} := (V, E_{\mathsf{core}})$ of $G = (V, E)$, such that the edge-set E is partitioned into E_{new} and E_{core}. To be more specific, at the start of a phase, we set $E_{\mathsf{new}} := \emptyset$ and $E_{\mathsf{core}} := E$. Subsequently, during the phase, whenever an edge e gets deleted from G, we set $E_{\mathsf{core}} \leftarrow E_{\mathsf{core}} \setminus \{e\}$ and $E_{\mathsf{new}} \leftarrow E_{\mathsf{new}} \setminus \{e\}$. In contrast, whenever an edge e gets inserted into G, we leave the set E_{core} unchanged, and set $E_{\mathsf{new}} \leftarrow E_{\mathsf{new}} \cup \{e\}$.

It is easy to verify that within any given phase G_{core} is a *decremental* subgraph of G. Furthermore, since $E_{\mathsf{new}} = \emptyset$ at the start of a phase, each update adds at most one edge to E_{new} and a phase lasts for δn updates, it follows that $|E_{\mathsf{new}}| \leq \delta n$ at all times.

We maintain a maximal matching $M_{\mathsf{core}} \subseteq E_{\mathsf{core}}$ in G_{core}, using the decremental algorithm from Section 4. In a bit more detail, we handle an update in G as follows. We first let the decremental algorithm from Section 4 handle the resulting update in G_{core} and accordingly modify the matching M_{core}. Next, we scan through all the edges in G_{new} and greedily add as many of these edges as we can on top of M_{core}, subject to the condition that the resulting edge-set $M_{\mathsf{final}} \supseteq M_{\mathsf{core}}$ remains a valid matching in G. This scan takes $O(|E_{\mathsf{new}}|) = O(\delta n)$ time. It is easy to verify that the matching M_{final} we obtain at the end of our scan is a maximal matching in G.

To summarize, with an additive overhead of $O(\delta n)$ update time, we can convert the decremental maximal matching algorithm from Section 4 into a fully dynamic algorithm. Finally, looking back at the analysis from Section 4.4, it is easy to verify that this additive overhead of $O(\delta n)$ does *not* degrade our overall asymptotic update time (the values of δ, B remain the same as in Section 4.4).

5 Decremental Perfect Matching in Graphs with Degree Gap

We say that a bipartite graph $G = (L, R, E)$ has a *γ-degree-gap at X* if $\deg_G(v) \geq X$ for all $v \in L$ and $\deg_G(v) \leq X(1-\gamma)$ for all $v \in R$. Note that if $\gamma \geq 0$, then, by Hall's theorem, there exists a matching M that is *left-perfect*, i.e., M matches every vertex in L.

Lemma 5.1. *Let $G = (L, R, E)$ be a bipartite graph with initial γ-degree-gap at X. We can build a data structure LPM (stands for Left Perfect Matching) on G that maintains a matching M and supports the following operations:*

- Init(M_0) *where M_0 is a matching in G before any update: $M \leftarrow M_0$.*
- Delete(u, v) *where $(u, v) \in E(L, R)$: set $G \leftarrow G - (u, v)$ and $M \leftarrow M - (u, v)$. If $\deg_G(u) < X$ and $(u, v') \in M$, $M \leftarrow M - (u, v')$.*
- Augment(u) *where $u \in L$ is M-free and $\deg_G(u) \geq X$: augment M by an augmenting path so that u is M-matched.*

The data structure can be implemented by either:

(1) *a deterministic algorithm with $O(n\Delta\frac{\log(n)}{\gamma}) \cdot (1 + \frac{u}{\sqrt{n\gamma}})$ total update time where Δ is the maximum degree of G and u is the number of calls to* Delete$(\cdot)$*. In particular, the total update time is independent of the number of calls to* Augment$(\cdot)$*.*

(2) *a randomized algorithm that takes $O(|M_0|)$ time to* Init$(\cdot)$*, $O(X \log n)$ worst-case time to* Delete$(\cdot)$*, and $O(\log^3(n)/\gamma^3)$ time to* Augment$(\cdot)$*. The algorithm is correct with high probability against an adaptive adversary. The algorithm assumes that each vertex in G has access to a binary search tree of edges incident to it.*

Note that the LPM data structure allows one to maintain a matching that matches all nodes with L with $\deg_G(u) \geq X$: whenever such a node u becomes unmatched, call Augment(u).

Due to page limitations, we defer the proof of Lemma 5.1 and the precise description of our dynamic algorithm to the full version of the paper.

6 Open Problems

Our novel approach for maintaining a maximal matching suggests several interesting questions.

The Right Answer. Fundamentally, how fast can we maintain a maximal matching against an adaptive adversary? Our approach inherently takes $\Omega(\min\{\sqrt{n}, \Delta\})$ update time where Δ is the maximum degree. Is there an algorithm against an adaptive adversary with $o(\min\{\sqrt{n}, \Delta\})$ update time? Otherwise, is there a conditional lower bound refuting this hope, or even refuting polylogarithmic update time? The answer in either direction would be very exciting.

Better Decremental Perfect Matching. The algorithm from Lemma 5.1 is the main bottleneck in our dynamic maximal matching algorithm. In the *static* setting, we can deterministically find a perfect matching in an m-edge graph with γ-degree-gap with $\tilde{O}(m/\gamma)$ time, for example, using Dinic's flow algorithm. Can we deterministically match this running time in the decremental setting? Such an algorithm would immediately imply a deterministic dynamic maximal matching algorithm with $\tilde{O}(n^{4/5})$ update time via our framework.

We can also hope to improve our randomized update time further to $\tilde{O}(n^{2/3})$. Currently, our amortized time for initializing each phase is $\tilde{O}(\frac{n}{B\delta} + \delta n + \frac{B}{\delta})$. But we can actually improve this to $\tilde{O}(\frac{n}{B\delta} + \delta n + B)$ via warm-starting. We omit this because it alone does not lead to any improvement and will complicate the algorithm further. But, suppose that, in our randomized algorithm, the time spent on Augment$(\cdot)$ can be improved from $\tilde{O}(1/\gamma^3)$ to $\tilde{O}(1/\gamma^2)$. Combining this with warm-starting would immediately improve the final update time from

$$\tilde{O}(\frac{n}{B\delta} + \delta n + \frac{B}{\delta} + \frac{1}{\delta^3}) = \tilde{O}(n^{3/4}) \text{ by setting } \delta = \frac{1}{n^{1/4}} \text{ and } B = n^{1/2}$$

to

$$\tilde{O}(\frac{n}{B\delta} + \delta n + B + \frac{1}{\delta^2}) = \tilde{O}(n^{2/3}) \text{ by setting } \delta = \frac{1}{n^{1/3}} \text{ and } B = n^{2/3},$$

where the improvement from $\frac{1}{\delta^3}$ to $\frac{1}{\delta^2}$ is from the assumption on Augment$(\cdot)$.

Worst-case Update Time. Interestingly, it is open if there exists a dynamic maximal matching algorithm with $o(n)$ worst-case update time, even against an *oblivious* adversary. Previous algorithms against an oblivious adversary inherently guarantee only amortized update time because they must spend a lot of time once the adversary "deletes the sampled edge".

Our algorithm might be a starting point to resolve this open problem. Indeed, our approach exploits amortization in a much more superficial manner. Within each phase, our update time can be made worst-case.Thus, it remains to spread the work for preprocessing at the beginning of each phase; the task has been easily handled in many other dynamic problems. The complication we observed is as follows. However, we do not see this as a fundamental barrier.

References

[1] Amir Abboud, Robert Krauthgamer, Jason Li, Debmalya Panigrahi, Thatchaphol Saranurak, and Ohad Trabelsi. 2022. Breaking the cubic barrier for all-pairs max-flow: Gomory-hu tree in nearly quadratic time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 884–895.

[2] Amir Abboud, Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. 2023. All-pairs max-flow is no harder than single-pair max-flow: Gomory-hu trees in almost-linear time. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2204–2212.

[3] Noga Alon, László Babai, and Alon Itai. 1986. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of algorithms* 7, 4 (1986), 567–583.

[4] Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. 2018. Dynamic Matching: Reducing Integral Algorithms to Approximately-Maximal Fractional Algorithms. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic (LIPIcs, Vol. 107)*, Ioannis Chatzigiannakis, Christos Kaklamanis, Daniel Marx, and Donald Sannella (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 7:1–7:16. doi:10.4230/LIPICS.ICALP.2018.7

[5] Sepehr Assadi. 2024. Faster Vizing and Near-Vizing Edge Coloring Algorithms. *arXiv preprint arXiv:2405.13371* (2024).

[6] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. 2019. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1616–1635.

[7] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. 2019. Coresets Meet EDCS: Algorithms for Matching and Vertex Cover on Massive Graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, Timothy M. Chan (Ed.). SIAM, 1616–1635. doi:10.1137/1.9781611975482.98

[8] 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 2023, Orlando, FL, USA, June 20-23, 2023*, Barna Saha and Rocco A. Servedio (Eds.). ACM, 131–144. doi:10.1145/3564246.3585110

[9] Sepehr Assadi and Aaron Bernstein. 2019. Towards a Unified Theory of Sparsification for Matching Problems. In *Proceedings of the 2nd Symposium on Simplicity in Algorithms (SOSA)*. 11:1–11:20.

[10] Sepehr Assadi, Aaron Bernstein, and Aditi Dudeja. 2022. Decremental Matching in General Graphs. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France (LIPIcs, Vol. 229)*, Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 11:1–11:19. doi:10.4230/LIPICS.ICALP.2022.11

[11] Sepehr Assadi, Sanjeev Khanna, and Yang Li. 2019. The Stochastic Matching Problem with (Very) Few Queries. *ACM Trans. Economics and Comput.* 7, 3 (2019), 16:1–16:19.

[12] Baruch Awerbuch, Andrew V Goldberg, Michael Luby, and Serge A Plotkin. 1989. Network decomposition and locality in distributed computation. In *FOCS*, Vol. 30. Citeseer, 364–369.

[13] Amir Azarmehr and Soheil Behnezhad. 2023. Robust Communication Complexity of Matching: EDCS Achieves 5/6 Approximation. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[14] Amir Azarmehr, Soheil Behnezhad, and Mohammad Roghani. 2024. Fully Dynamic Matching: $(2-\sqrt{2})$-Approximation in Polylog Update Time. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, David P. Woodruff (Ed.). SIAM, 3040–3061. doi:10.1137/1.9781611977912.109

[15] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. 2021. Lower bounds for maximal matchings and maximal independent sets. *Journal of the ACM (JACM)* 68, 5 (2021), 1–30.

[16] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. 2016. The locality of distributed symmetry breaking. *Journal of the ACM (JACM)* 63, 3 (2016), 1–45.

[17] Surender Baswana, Manoj Gupta, and Sandeep Sen. 2011. Fully dynamic maximal matching in $O(\log n)$ update time. In *Proceedings of the 52nd Symposium on Foundations of Computer Science (FOCS)*. 383–392.

[18] Surender Baswana, Manoj Gupta, and Sandeep Sen. 2015. Fully Dynamic Maximal Matching in $O(\log(n))$ Update Time. *SIAM J. Comput.* 44, 1 (2015), 88–113. doi:10.1137/130914140 Announced at FOCS'11..

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

[20] Soheil Behnezhad. 2023. Dynamic Algorithms for Maximum Matching Size. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. To appear in.

[21] Soheil Behnezhad. 2023. Dynamic Algorithms for Maximum Matching Size. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, Nikhil Bansal and Viswanath Nagarajan (Eds.). SIAM, 129–162. doi:10.1137/1.9781611977554.CH6

[22] 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.

[23] Soheil Behnezhad, Mohammad Taghi Hajiaghayi, and David G Harris. 2019. Exponentially faster massively parallel maximal matching. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1637–1649.

[24] 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.

[25] Soheil Behnezhad, Rajmohan Rajaraman, and Omer Wasim. 2025. Fully Dynamic $(\Delta+1)$ Coloring Against Adaptive Adversaries. In *Proceedings of the 2025 ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

[26] Aaron Bernstein. 2017. Deterministic partially dynamic single source shortest paths in weighted graphs. *arXiv preprint arXiv:1705.10097* (2017).

[27] Aaron Bernstein. 2024. Improved Bounds for Matching in Random-Order Streams. *Theory Comput. Syst.* 68, 4 (2024), 758–772. doi:10.1007/S00224-023-10155-7

[28] Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. 2020. Fully-dynamic graph sparsifiers against an adaptive adversary. *arXiv preprint arXiv:2004.08432* (2020).

[29] Aaron Bernstein and Shiri Chechik. 2016. Deterministic decremental single source shortest paths: beyond the o (mn) bound. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. 389–397.

[30] Aaron Bernstein and Shiri Chechik. 2017. Deterministic partially dynamic single source shortest paths for sparse graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 453–469.

[31] Aaron Bernstein, Jiale Chen, Aditi Dudeja, Zachary Langley, Aaron Sidford, and Ta-Wei Tu. 2024. Matching Composition and Efficient Weight Reduction in Dynamic Matching. *arXiv preprint arXiv:2410.18936* (2024). To appear at SODA'25.

[32] Aaron Bernstein, Aditi Dudeja, and Zachary Langley. 2021. A framework for dynamic matching in weighted graphs. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, Samir Khuller and Virginia Vassilevska Williams (Eds.). ACM, 668–681. doi:10.1145/3406325.3451113

[33] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. 2021. A Deamortization Approach for Dynamic Spanner and Dynamic Maximal Matching. *ACM Trans. Algorithms* 17, 4 (2021), 29:1–29:51. doi:10.1145/3469833

[34] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. 2020. Deterministic Decremental Reachability, SCC, and Shortest Paths via Directed Expanders and Congestion Balancing. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, Sandy Irani (Ed.). IEEE, 1123–1134. doi:10.1109/FOCS46700.2020.00108

[35] Aaron Bernstein and Cliff Stein. 2015. Fully Dynamic Matching in Bipartite Graphs. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9134)*, Magnus M. Halldorsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann (Eds.). Springer, 167–179. doi:10.1007/978-3-662-47672-7_14

[36] Aaron Bernstein and Cliff Stein. 2015. Fully dynamic matching in bipartite graphs. In *International Colloquium on Automata, Languages, and Programming*. Springer, 167–179.

[37] Aaron Bernstein and Cliff Stein. 2016. Faster Fully Dynamic Matchings with Small Approximation Ratios. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, Robert Krauthgamer (Ed.). SIAM, 692–711. doi:10.1137/1.9781611974331.CH50

[38] Aaron Bernstein and Cliff Stein. 2016. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 692–711.

[39] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. 2017. Deterministic Fully Dynamic Approximate Vertex Cover and Fractional Matching in O(1) Amortized Update Time. In *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10328)*, Friedrich Eisenbrand and Jochen Könemann (Eds.). Springer, 86–98. doi:10.1007/978-3-319-59250-3_8

[40] Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. 2018. Dynamic Algorithms for Graph Coloring. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, Artur Czumaj (Ed.). SIAM, 1–20. doi:10.1137/1.9781611975031.1

[41] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. 2018. Deterministic Fully Dynamic Data Structures for Vertex Cover and Matching. *SIAM J. Comput.* 47, 3 (2018), 859–887. doi:10.1137/140998925

[42] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 2016. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, Daniel Wichs and Yishay Mansour (Eds.). ACM, 398–411. doi:10.1145/2897518.2897568

[43] 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 Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, Philip N. Klein (Ed.). SIAM, 470–489. doi:10.1137/1.9781611974782.30

[44] Sayan Bhattacharya and Peter Kiss. 2021. Deterministic Rounding of Dynamic Fractional Matchings. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference) (LIPIcs, Vol. 198)*, Nikhil Bansal, Emanuela Merelli, and James Worrell (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 27:1–27:14. doi:10.4230/LIPICS.ICALP.2021.27

[45] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. 2023. Dynamic Algorithms for Packing-Covering LPs via Multiplicative Weight Updates. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 1–47.

[46] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. 2023. Sublinear Algorithms for $(1.5+\epsilon)$-Approximate Matching. In *STOC*.

[47] 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 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, Nikhil Bansal and Viswanath Nagarajan (Eds.). SIAM, 100–128. doi:10.1137/1.9781611977554.CH5

[48] 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 2024, Vancouver, BC, Canada, June 24-28, 2024*, Bojan Mohar, Igor Shinkar, and Ryan O'Donnell (Eds.). ACM, 59–70. doi:10.1145/3618260.3649648

[49] Sayan Bhattacharya and Janardhan Kulkarni. 2019. Deterministically Maintaining a $(2+\epsilon)$-Approximate Minimum Vertex Cover in $O(1/\epsilon)$ Amortized Update Time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, Timothy M. Chan (Ed.). SIAM, 1872–1885. doi:10.1137/1.9781611975482.113

[50] Sayan Bhattacharya, Thatchaphol Saranurak, and Pattara Sukprasert. 2022. Simple dynamic spanners with near-optimal recourse against an adaptive adversary. *arXiv preprint arXiv:2207.04954* (2022).

[51] Joakim Blikstad and Peter Kiss. 2023. Incremental $(1-\epsilon)$-approximate dynamic matching in $O(poly(1/\epsilon))$ update time. *arXiv preprint arXiv:2302.08432* (2023).

[52] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P Liu, Simon Meierhans, Maximilian Probst Gutenberg, and Sushant Sachdeva. 2024. Almost-Linear Time Algorithms for Decremental Graphs: Min-Cost Flow and More via Duality. *arXiv preprint arXiv:2407.10830* (2024).

[53] Moses Charikar and Shay Solomon. 2018. Fully Dynamic Almost-Maximal Matching: Breaking the Polynomial Worst-Case Time Barrier. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic (LIPIcs, Vol. 107)*, Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 33:1–33:14. doi:10.4230/LIPICS.ICALP.2018.33

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

[55] Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. 2020. Fast dynamic cuts, distances and effective resistances via vertex sparsifiers. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1135–1146.

[56] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. 2022. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 612–623.

[57] Aleksander BG Christiansen, Jacob Holm, Ivor van der Hoog, Eva Rotenberg, and Chris Schwiegelshohn. 2022. Adaptive out-orientations with applications. *arXiv preprint arXiv:2209.14087* (2022).

[58] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. 2020. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1158–1167.

[59] 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 SIGACT Symposium on Theory of Computing*. 389–400.

[60] Julia Chuzhoy and Thatchaphol Saranurak. 2021. Deterministic algorithms for decremental shortest paths via layered core decomposition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2478–2496.

[61] Julia Chuzhoy and Ruimin Zhang. 2023. A new deterministic algorithm for fully dynamic all-pairs shortest paths. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*. 1159–1172.

[62] Artur Czumaj, Jakub Łacki, Aleksander Madry, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. 2018. Round compression for parallel matching algorithms. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. 471–484.

[63] 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.

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

[65] Michael Elkin and Ariel Khuzman. 2024. Deterministic Simple $(1+\epsilon)\Delta$-Edge-Coloring in Near-Linear Time. *arXiv preprint arXiv:2401.10538* (2024).

[66] Manuela Fischer. 2020. Improved deterministic distributed matching via rounding. *Distributed Computing* 33, 3 (2020), 279–291.

[67] Harold N. Gabow, Takao Nishizeki, Oded Kariv, Daniel Leven, and Osamu Terada. 1985. *Algorithms for edge-coloring*. Technical Report 41/85. Tel Aviv University.

[68] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. 2018. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. 129–138.

[69] Mohsen Ghaffari and Bernhard Haeupler. 2021. A time-optimal randomized parallel algorithm for mis. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2892–2903.

[70] Mohsen Ghaffari and Anton Trygub. 2024. Parallel Dynamic Maximal Matching. In *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures*. 427–437.

[71] Mohsen Ghaffari and Jara Uitto. 2019. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1636–1653.

[72] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. 2021. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2212–2228.

[73] Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. 2019. $(1+\epsilon)$-Approximate Incremental Matching in Constant Deterministic Amortized Time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1886–1898.

[74] 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. In *5th Symposium on Simplicity in Algorithms, SOSA@SODA 2022, Virtual Conference, January 10-11, 2022*, Karl Bringmann and Timothy M. Chan (Eds.). SIAM, 12–23. doi:10.1137/1.9781611977066.2

[75] 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.

[76] Manoj Gupta and Richard Peng. 2013. Fully Dynamic $(1+\epsilon)$-Approximate Matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*. IEEE Computer Society, 548–557. doi:10.1109/FOCS.2013.65

[77] Manoj Gupta, Venkatesh Raman, and SP Suresh. 2014. Maintaining Approximate Maximum Matching in an Incremental Bipartite Graph in Polylogarithmic

Update Time. In *Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, Vol. 29. 227–239.

[78] Maximilian Probst Gutenberg and Christian Wulff-Nilsen. 2020. Decremental SSSP in weighted digraphs: Faster and against an adaptive adversary. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2542–2561.

[79] Maximilian Probst Gutenberg and Christian Wulff-Nilsen. 2020. Deterministic algorithms for decremental approximate shortest paths: Faster and simpler. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2522–2541.

[80] Bernhard Haeupler, Yaowei Long, and Thatchaphol Saranurak. 2024. Dynamic Deterministic Constant-Approximate Distance Oracles with n^{ϵ} Worst-Case Update Time. *arXiv preprint arXiv:2402.18541* (2024).

[81] Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. 1999. A faster distributed algorithm for computing maximal matchings deterministically. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*. 219–228.

[82] Michal Hanckowiak, Michal Karonski, and Alessandro Panconesi. 2001. On the distributed complexity of computing maximal matchings. *SIAM Journal on Discrete Mathematics* 15, 1 (2001), 41–57.

[83] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. 2016. Dynamic approximate all-pairs shortest paths: Breaking the o(mn) barrier and derandomization. *SIAM J. Comput.* 45, 3 (2016), 947–1006.

[84] 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 Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, Rocco A. Servedio and Ronitt Rubinfeld (Eds.). ACM, 21–30. doi:10.1145/2746539.2746609

[85] Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. 2001. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)* 48, 4 (2001), 723–760.

[86] Amos Israeli and Alon Itai. 1986. A fast and simple randomized parallel algorithm for maximal matching. *Inform. Process. Lett.* 22, 2 (1986), 77–80.

[87] Amos Israeli and Yossi Shiloach. 1986. An improved parallel algorithm for maximal matching. *Inform. Process. Lett.* 22, 2 (1986), 57–60.

[88] Zoran Ivkovic and Errol L Lloyd. 1993. Fully Dynamic Maintenance of Vertex Cover. In *Proceedings of the 19th International Workshop on Graph-Theoretic Concepts in Computer Science*. 99–111.

[89] Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. 2022. Regularized Box-Simplex Games and Dynamic Decremental Bipartite Matching. In *International Colloquium on Automata, Languages, and Programming (ICALP)*.

[90] Richard M Karp and Avi Wigderson. 1985. A fast parallel algorithm for the maximal independent set problem. *Journal of the ACM (JACM)* 32, 4 (1985), 762–773.

[91] V. King. 1999. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*. 81–89.

[92] Peter Kiss. 2022. Deterministic Dynamic Matching in Worst-Case Update Time. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA (LIPIcs, Vol. 215)*, Mark Braverman (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 94:1–94:21. doi:10.4230/LIPICS.ITCS.2022.94

[93] Rasmus Kyng, Simon Meierhans, and Maximilian Probst Gutenberg. 2024. A dynamic shortest paths toolbox: Low-congestion vertex sparsifiers and their applications. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*. 1174–1183.

[94] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. 2011. Filtering: a method for solving graph problems in mapreduce. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*. 85–94.

[95] Yang P. Liu. 2024. On Approximate Fully-Dynamic Matching and Online Matrix-Vector Multiplication. *CoRR* abs/2403.02582 (2024). doi:10.48550/ARXIV.2403.02582 arXiv:2403.02582

[96] Michael Luby. 1985. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. 1–10.

[97] 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.

[98] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. 2017. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 950–961.

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

[100] 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.

[101] Yossi Shiloach and Shimon Even. 1981. An On-Line Edge-Deletion Problem. *J. ACM* 28, 1 (jan 1981), 1–4.

[102] Shay Solomon. 2016. Fully Dynamic Maximal Matching in Constant Update Time. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, Irit Dinur (Ed.). IEEE Computer Society, 325–334. doi:10.1109/FOCS.2016.43

[103] 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.

[104] Shay Solomon. 2022. Open Problem. *Report from Dagstuhl Seminar 22461 Dynamic Graph Algorithms* (2022), 62. doi:10.4230/DagRep.12.11.45

[105] Daniel Stubbs and Virginia Vassilevska Williams. 2017. Metatheorems for Dynamic Weighted Matching. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA (LIPIcs, Vol. 67)*, Christos H. Papadimitriou (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 58:1–58:14. doi:10.4230/LIPICS.ITCS.2017.58

[106] Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P Liu, Richard Peng, and Aaron Sidford. 2022. Faster maxflow via improved dynamic spectral vertex sparsifiers. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. 543–556.

[107] David Wajc. 2020. Rounding dynamic matchings against an adaptive adversary. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy (Eds.). ACM, 194–207. doi:10.1145/3357713.3384258

[108] Christian Wulff-Nilsen. 2017. Fully-dynamic minimum spanning forest with improved worst-case update time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 1130–1143.

Received 2024-11-03; accepted 2025-02-01