

New Trade-Offs for Fully Dynamic Matching via Hierarchical EDCS

Soheil Behnezhad
Stanford University
`beh@stanford.edu`

Sanjeev Khanna*
University of Pennsylvania
`sanjeev@cis.upenn.edu`

Abstract

We study the maximum matching problem in *fully dynamic* graphs: a graph is undergoing both edge insertions and deletions, and the goal is to efficiently maintain a large matching after each edge update. This problem has received considerable attention in recent years. The known algorithms naturally exhibit a trade-off between the quality of the matching maintained (i.e., the approximation ratio) and the time needed per update. While several interesting results have been obtained, the optimal behavior of this trade-off remains largely unclear. Our main contribution is a new approach to designing fully dynamic approximate matching algorithms that in a *unified manner* not only (essentially) recovers all previously known trade-offs that were achieved via very different techniques, but reveals some new ones as well.

Specifically, we introduce a generalization of the *edge-degree constrained subgraph* (EDCS) of Bernstein and Stein (2015) that we call the *hierarchical EDCS* (HEDCS). We also present a randomized algorithm for efficiently maintaining an HEDCS. In an m -edge graph with maximum degree Δ , for any integer $k \geq 0$ that is essentially the number of levels of the hierarchy in HEDCS, our algorithm takes $\tilde{O}(\min\{\Delta^{1/(k+1)}, m^{1/(2k+2)}\})$ worst-case update-time and maintains an (almost) $\alpha(k)$ -approximate matching where we show:

- $\alpha(0) = 1$, $\alpha(1) = \frac{2}{3}$, $\alpha(\frac{1}{\delta}) \geq (\frac{1}{2} + \Omega_{\delta}(1))$ for any $\delta > 0$, and $\alpha(\log \Delta) \geq \frac{1}{2}$.

These bounds recover all previous trade-offs known for dynamic matching in the literature up to logarithmic factors in the update-time.

- $\alpha(2) > .612$ for bipartite graphs, and $\alpha(2) > .609$ for general graphs.

Note that these approximations are obtained in $\tilde{O}(\min\{\Delta^{1/3}, m^{1/6}\})$ update-time.

- $\alpha(3) > .563$ for bipartite graphs, and $\alpha(3) > .532$ for general graphs.

Note that these approximations are obtained in $\tilde{O}(\min\{\Delta^{1/4}, m^{1/8}\})$ update-time.

*Supported in part by NSF awards CCF-1763514, CCF-1934876, and CCF-2008305.

Contents

1	Introduction	3
1.1	Overview of Techniques	4
1.2	Organization	6
2	Preliminaries	6
3	Hierarchical Edge-Degree Constrained Subgraphs (HEDCS)	8
3.1	Basic Properties of HEDCS	8
3.2	Approximation Ratio of HEDCS: Basic Definitions	8
3.3	Approximation Ratio of HEDCS: A Factor Revealing LP	10
4	Maintaining a Hierarchical EDCS in Fully Dynamic Graphs	12
4.1	High-Level Overview of the Algorithm	13
4.2	The Formal Algorithm	14
4.3	Basic Properties of the Algorithm of Section 4.2	15
4.4	The Approximation Ratio	17
4.5	Update Time of the Algorithm of Section 4.2	20
4.6	Getting a Worst-Case Update-time Bound	26
4.7	Bounding Maximum Degree by $O(\sqrt{m})$	26
5	Conclusion & Open Problems	27
A	Approximation Ratio of HEDCS: An Analytical Lower Bound	32
B	Proof of Proposition 3.6	34
C	Proof of Claim 4.7	37

1 Introduction

The maximum matching problem in graphs plays a central role in combinatorial optimization, and hence has been extensively studied in the classical setting where we are given a static graph, and the goal is to compute a maximum matching of the graph. However, in many applications of the matching problem, the input graph may be dynamically changing via edge insertions and deletions. A natural question is if it is possible to efficiently maintain a near-optimal matching of a graph as it undergoes changes. In this paper, we study design of efficient *fully dynamic* algorithms for the *maximum matching* problem. Specifically, given a graph G that undergoes both edge insertions and deletions, the goal is to maintain a matching of G that

- (i) is approximately as large as the maximum matching of G after each update, and
- (ii) the time-complexity needed to address each update is small.

Throughout the paper, we will denote by n the number of vertices in G , by m the maximum number of edges in G at any time, and by Δ the maximum degree in G at any time. We will say that a matching is an α -*approximate* for some $\alpha \in [0, 1]$ if its size is at least an α -fraction of the maximum matching size.

When the goal is to maintain an *exact* maximum matching, then there are conditional lower bounds [AW14, HKNS15, Dah16] ruling out any $O(n^{1-\varepsilon})$ update-time algorithm (see also [San07, vdBNS19] for some progress on the algorithmic side). As a result, much of the focus in the literature has been on approximate solutions (see e.g. [BGS11, BGS18, NS13, GP13, BHI18, BHN17, BHN16, Sol16, CS18, ACC⁺18, BFH19, BDH⁺19, BLM20, Waj20, BDL21, BK21, RSW21] and the references therein). These works indeed show that settling for an approximate matching does translate into much improved update times. We highlight some of these results below.

Baswana, Gupta, and Sen [BGS11, BGS18] showed that a maximal matching, and hence a $1/2$ -approximate matching, can be maintained in $O(\log n)$ amortized update time (see also the follow-up work by Solomon [Sol16]). At a high-level, their algorithm is based on the insight that if a vertex of degree d is matched to a random neighbor, then in expectation, $\Omega(d)$ updates need to occur before this edge is affected. Gupta and Peng [GP13] showed that for any fixed $\varepsilon > 0$, a $(1-\varepsilon)$ -approximate matching can be maintained with a worst-case update time of $O(\min\{\Delta, m^{1/2}\})$. At a high-level, their approach is based on recomputing a $(1 - \varepsilon)$ -approximate matching once a sufficiently large number of updates have occurred. This idea directly gives the desired result when the matching size is large, and the authors then show that whenever the matching size is small, the underlying graph can be appropriately reduced in size. Bernstein and Stein [BS15a, BS16] introduced a powerful data structure called the *edge-degree constrained subgraph* (EDCS), a sparse subgraph of the original graph guaranteed to contain an (almost) $2/3$ -approximate matching. The authors showed that this data structure can be maintained with $O(\min\{\Delta^{1/2}, m^{1/4}\})$ amortized update time, yielding a much faster algorithm for maintaining a $2/3$ -approximate matching. A different approach, based on augmenting a half-approximation using short augmenting paths, was subsequently used by [BLM20] (see also [BHN16, Waj20]) to show that a $(1/2 + \Omega_\varepsilon(1))$ -approximate matching can be maintained in $\tilde{O}(\Delta^\varepsilon)$ update time for any $\varepsilon > 0$.

The results above clearly highlight a trade-off between the approximation ratio of the maintained matching and the update-time. While these results capture many interesting trade-offs, two salient features of the current state of the art are (i) there are many interesting regions where the trade-off between approximation ratio and update time is not understood, and (ii) very different techniques are used in obtaining results at different parts of this trade-off spectrum.

Approximation	Update-Time	k	Note
~ 1	$\tilde{O}(\min\{\Delta, m^{1/2}\})$	$k = 0$	This matches [GP13].
$\sim 2/3$	$\tilde{O}(\min\{\Delta^{1/2}, m^{1/4}\})$	$k = 1$	This matches [BS15a, BS16].
.609 (.612 bipartite)	$\tilde{O}(\min\{\Delta^{1/3}, m^{1/6}\})$	$k = 2$	This is a new trade-off.
.532 (.563 bipartite)	$\tilde{O}(\min\{\Delta^{1/4}, m^{1/8}\})$	$k = 3$	This is a new trade-off.
\vdots			
$1/2 + \Omega_\delta(1)$	$\tilde{O}_\delta(\min\{\Delta^\delta, m^{\delta/2}\})$	$k = 1/\delta - 1$	This matches [BLM20].
$\sim 1/2$	$\tilde{O}(1)$	$k = \Theta(\log \Delta)$	This matches [BGS11, BGS18, So16].

Table 1: Approximation/update-time trade-offs of our algorithm for different values of parameter k . The algorithm is randomized and the bound on the update-time is worst-case. By an approximation factor of $\sim \alpha$ we mean the algorithm can get $(1 - \varepsilon)\alpha$ -approximation for any fixed $\varepsilon > 0$.

The main contribution of this work is a new approach to designing fully dynamic algorithms for approximate maximum matching that in a *unified manner* not only (essentially) recovers all previously known trade-offs (up to logarithmic factors in the update-time) but reveals some new ones as well. Specifically, we prove the following theorem:

Theorem 1.1. *For any integer $k \geq 0$ and any $\varepsilon > 0$, there is a randomized algorithm that maintains an $(\alpha(k) - \varepsilon)$ -approximate maximum matching of a fully dynamic graph against an oblivious adversary with worst-case update time $\min\{\Delta^{1/(k+1)}, m^{1/(2k+2)}\} \cdot \text{poly}(k, 1/\varepsilon, \log n)$ where*

$$\alpha(0) = 1, \alpha(1) = 2/3, \alpha(2) \geq .609, \alpha(3) \geq .532, \dots, \alpha(\log \Delta) \geq 1/2.$$

If the graph is bipartite, then we show $\alpha(2) \geq .612$ and $\alpha(3) \geq .563$.

Therefore, our algorithm takes as input an integer parameter $k \geq 0$ such that as k goes from 0 to $\log \Delta$, the update time improves from $\tilde{O}(\min\{\Delta, m^{1/2}\})$ to $\tilde{O}(1)$. The approximation ratio, on the other hand, goes from almost 1 to almost 1/2 as k goes from 0 to $\log \Delta$.

Additionally, we prove $\alpha(1/\delta) \geq \frac{1}{2} + \frac{1}{2^{2\tilde{O}(1/\delta)}}$ for any $\delta > 0$. Thus our algorithm can also beat half-approximation with any arbitrarily small polynomial update-time.

Table 1 summarizes these trade-offs and compares them with prior works.

1.1 Overview of Techniques

As our main tool, we introduce a generalization of the *edge-degree constrained subgraph* (EDCS) of Bernstein and Stein [BS15b, BS16] that we call the *hierarchical EDCS* (HEDCS). Before formalizing our generalization HEDCS of EDCS, let us recall the notion of EDCS.

For any edge $e = (u, v)$ in a graph G we use the notation $\deg_G(e) := \deg_G(u) + \deg_G(v)$ to denote the degree of e in H , which is simply sum of the degrees of its endpoints. For an integer $\beta > 1$, a subgraph H of a graph G is called a β -EDCS of G if:

- (i) For any $e \in H$, $\deg_H(e) \leq \beta$.
- (ii) For any $e \in G \setminus H$, $\deg_H(e) \geq \beta - 1$.

Interestingly, these two local constraints suffice to guarantee that subgraph H of G includes a $(2/3 - O(\varepsilon))$ -approximate maximum matching of G if $\beta \geq 1/\varepsilon$ (see [Beh21, AB19]).

The HEDCS is a hierarchical generalization of EDCS, where in addition to β , we have a parameter k which is the number of levels in the hierarchy. Formally, the HEDCS is defined as:

Definition 1.2 (Hierarchical Edge-Degree Constrained Subgraphs (HEDCS)).

Let $\beta \geq 2$ and $k \geq 1$ be integers. We say that a graph H is a β -HEDCS $_k$ of G iff there is a hierarchical decomposition $\emptyset = H_0 \subseteq H_1 \subseteq H_2 \subseteq \dots \subseteq H_k = H$ satisfying the following:

- (i) For every $1 \leq i \leq k$ and any edge $e \in H_i \setminus H_{i-1}$, $\deg_{H_i}(e) \leq \beta$.
- (ii) For any edge $e \in G \setminus H$, $\deg_H(e) \geq \beta - 1$.

Note that a β -HEDCS $_k$ for $k = 1$ is equivalent to a β -EDCS. However, as k becomes larger than 1, the edge-degree constraints in a β -HEDCS $_k$ H of G become more relaxed, and for some edges e , we may now have $\deg_H(e) > \beta$ (this can, e.g., happen for any edge in H_{k-1}). This relaxation of edge-degree constraints gradually weakens the $2/3$ -approximation guarantee of an EDCS as k becomes larger but in return, we show that it becomes easier to maintain a β -HEDCS $_k$ in dynamic graphs as we increase the number of levels k .

Maintaining an HEDCS: We give an algorithm that for any β and k can (lazily) maintain a β -HEDCS $_k$ in a fully dynamic graph with update-time $\tilde{O}(\min\{\Delta^{1/(k+1)}, m^{1/(2k+2)}\})$. Here, and for the sake of intuition, we will only overview the key insights behind maintaining a β -HEDCS $_k$ in amortized update-time $\tilde{O}(\Delta^{1/(k+1)})$.

Our definition of β -HEDCS $_k$ allows for a k -step greedy way of constructing it: construct H_1 , then construct H_2 by adding some edges to H_1 , then construct H_3 by adding edges to H_2 , and so on so forth. The crucial observation here is that each edge $e \in H_i$ is constrained by property (i) of HEDCS to have edge-degree $\leq \beta$ only in subgraph H_i , regardless of which edges belong to H_{i+1}, \dots, H_k . Hence, it is safe to increase the edge-degree of $e \in H_i$ in the higher levels to beyond β . As a result of this greedy construction, right after constructing H_i , any edge e with $\deg_{H_i}(e) \geq \beta - 1$ will for sure satisfy the constraint $\deg_H(e) \geq \beta - 1$ of property (ii) of HEDCS. Thus, this edge e can be safely ignored in constructing the higher levels.

To make use of the greedy construction above, we first random sample the edges of G into subgraphs $G_1 \subseteq G_2 \subseteq \dots \subseteq G_{k+1} = G$, where each G_i includes each edge of G with probability $p_i \approx \Delta^{\frac{i}{k+1}-1}$. We construct H_1 only using the edges of G_1 , then construct H_2 by adding some of the edges of $G_2 \setminus G_1$ to H_1 , then construct H_3 by adding some of the edges of $G_3 \setminus G_2$ to H_2 , etc. However, instead of considering all the edges in $G_i \setminus G_{i-1}$ in constructing H_i , we ignore those edges in $G_i \setminus G_{i-1}$ that are already covered by H_{i-1} . To make sure that this helps with pruning the set of edges that we consider in each level, we specifically construct each H_i in a way that guarantees a *sparsification property*. That is, the set of edges left uncovered by H_i in the remaining graph, will be in the order $\tilde{O}(\mu_i/p_i)$ where μ_i is the size of the largest matching in G_i .

One main challenge in maintaining this HEDCS H in a fully dynamic graph is that the edges that are removed from each H_i may result in uncovered edges in the remaining graph, invalidating property (ii) of HEDCS. The crucial observation is that each edge removal of the adversary belongs to H_i (and thus G_i) with probability at most p_i . Hence, the adversary needs to issue $\approx \varepsilon\mu_i/p_i$ updates to remove $\varepsilon\mu_i$ edges of H_i . We can thus take a lazy approach in maintaining our solution. For every $i \in [k]$, we can roughly speaking “pretend” for $\varepsilon\mu_i/p_i$ updates that no edge of H_i is

removed (i.e., we assume those removed still exist in the graph) and thus all edges covered by H_i remain covered. After $\varepsilon\mu_i/p_i$ updates, we recompute all of H_i, \dots, H_k from scratch, and amortize the cost over these updates. Since H_{i-1} , as discussed, only leaves $\tilde{O}(\mu_{i-1}/p_{i-1})$ edges uncovered, we are able to construct all of H_i, \dots, H_k in time $\tilde{O}(\mu_{i-1}/p_{i-1})$. Amortizing this cost over $\varepsilon\mu_i/p_i$ updates leads to a bound of $\approx \frac{\mu_{i-1}/p_{i-1}}{\varepsilon\mu_i/p_i} \lesssim \frac{p_i}{p_{i-1}} = \Delta^{1/(k+1)}$ update-time for each level $i \in [k]$.

It is worth noting that our algorithm for maintaining a β -HEDCS $_k$ is very different from the algorithms of Bernstein and Stein [BS16] for maintaining an EDCS. In particular, [BS16] maintain a β -EDCS for $\beta \approx \sqrt{\Delta}$ deterministically and their update-time bound is amortized. In our construction, however, β is much smaller and a constant value often suffices (for $k = 1$, particularly, where we recover the (almost) 2/3-approximation of [BS16], β is a constant). Additionally, we use randomization in a crucial way but achieve a worst-case update-time in return.

Approximation ratio of HEDCS: To understand the approximation ratio of β -HEDCS $_k$, we study a function $\alpha(k)$ which essentially tracks how the native 2/3-approximation guarantee of EDCS gradually weakens with increasing k , when β is sufficiently large. The precise analysis of the function $\alpha(k)$ becomes challenging even for small values of k . However, for any β , the computation of $\alpha(k)$ can be expressed as a linear program (LP) (formalized in Section 3.3). As a result, for $k = 2, 3$, we compute the value of $\alpha(k)$ by solving this LP for a sufficiently large value of β which we then feed into our dynamic algorithm.

For larger values of k and β sufficiently large with respect to k , we analytically compute the value $\alpha(k)$ and show that it is at least $1/2 + \Omega(1/2^{2^k})$ for any $k \geq 1$. This, in particular, means that for any fixed $\varepsilon > 0$, the update time can be driven down to $\tilde{O}_\varepsilon(\min\{\Delta^\varepsilon, m^{\varepsilon/2}\})$ while still obtaining an approximation ratio that is strictly better than 1/2, namely, $1/2 + \Omega_\varepsilon(1)$.

1.2 Organization

We start by presenting some notation and relevant results from previous works in Section 2. We then present the hierarchical EDCS (HEDCS) data structure and its properties in Section 3. We also present here an LP-based approach for analyzing the approximation ratio achieved by the HEDCS data structure, and show the performance implied by it when the number of hierarchy levels k is small. We defer the analysis of approximation achieved by HEDCS data structure when k is allowed to asymptotically grow to Appendix A. Finally, in Section 4 we present our randomized algorithm for maintaining the HEDCS data structure, and analyze its performance.

2 Preliminaries

Notation: We denote the input graph by $G = (V, E)$. The vertex-set V includes n vertices that are fixed. However, the edge-set E is dynamic. Particularly, edges can be both inserted and deleted from E . We use Δ as a fixed upper bound on the graph's maximum degree at all times.

All graphs that we define in this work are on the same vertex-set V as the input graph G . As such, when it is clear from the context, we may treat these (sub)graphs as essentially sets of edges. Particularly, for a subgraph H we may use $|H|$ to denote the number of edges in H , or may use $H \setminus H'$ for two graphs H and H' on vertex set V to denote a subgraph on the same vertex-set, including edges of H that do not belong to H' .

For any graph H , we use $\mu(H)$ to denote the size of a maximum matching in H and use $\tilde{\mu}(H)$

to denote the size of a maximal matching of H . (We particularly use $\tilde{\mu}(H)$ when we want the value to be computable in linear time.) For any edge $e = (u, v)$ we define $\deg_H(e) := \deg_H(u) + \deg_H(v)$ to be the *edge-degree* of e in graph H . We note that so long as the endpoints of e belong to the vertex-set of H , $\deg_H(e)$ is well-defined and e does not need to belong to the edge-set of H . For any edge $e = (u, v)$, we say e is (H, β) -*underfull* if $\deg_H(e) < \beta - 1$ and (H, β) -*overfull* if $\deg_H(e) > \beta$.

Throughout the paper, the $\tilde{O}(f)$ notation hides $\text{poly}(\log n)$ factors, i.e., $\tilde{O}(f) = f \cdot \text{poly}(\log n)$.

Oblivious Adversary and Worst-Case Update-Time: Our dynamic algorithms are based on the standard *oblivious adversary assumption*. The sequence of updates in this model are provided by a computationally unbounded adversary that knows the algorithm to be used. However, the updates should not depend on the coin flips of the dynamic algorithm. Equivalently, one may assume that the sequence of updates are fixed before the dynamic algorithm starts to operate.

As standard, we say a fully dynamic algorithm has “worst-case update-time” T if every update is w.h.p. addressed in T time by the algorithm.

Tools: We will use the following algorithms from prior work.

Proposition 2.1 ([MV80, Vaz12]). *Given any m -edge graph $G = (V, E)$ and any parameter $\varepsilon > 0$, there is a static algorithm to find a $(1 - \varepsilon)$ -approximate maximum matching of G in $O(m/\varepsilon)$ time.*

In our algorithm, we will need a subroutine that maintains a c -approximation to the size of maximum matching in $\text{poly}(\log n)$ time, where c can be any arbitrarily large constant. Since there are already highly efficient 2-approximate algorithms, we will use them for this purpose but emphasize that we can instead use any other $O(1)$ -approximate algorithm.

Proposition 2.2 (See [BFH19] or [BDH⁺19]). *There is a randomized algorithm that maintains a maximal matching of an n -vertex fully dynamic graph against an oblivious adversary in $\text{poly}(\log n)$ worst-case update-time.*

We also use the following algorithm to argue that if the maximum matching of G becomes small at any point during the updates, then there is already an algorithm that can efficiently maintain a $(1 - \varepsilon)$ -approximation during those updates. We use this algorithm to assume that at all times $\mu(G)$ is larger than, say $\log n$, which is useful for our probabilistic events. See [Remark 4.2](#).

Proposition 2.3 ([GP13]). *Let $\mu' = \Omega(1)$ and $0 < \varepsilon < 1$ be any parameters. There is a deterministic algorithm that maintains a matching M of a fully dynamic graph G with worst case update-time $O(\mu'/\varepsilon^2 + \log^3 n)$ satisfying the following: at any time during the updates where $\mu(G) \leq \mu'$ it also holds that $|M| \geq (1 - \varepsilon)\mu(G)$.*

Proof sketch. The idea is due to [GP13]. Consider a graph G and suppose that C is a vertex cover of size $O(\mu(G))$ of G . Consider a *core subgraph* of G that includes all the edges of G with both endpoints in C and also includes $|C| + 1$ arbitrary edges of each vertex in C . It is not hard to see that a core graph includes a maximum matching of G — see [GP13, Section 3] for the proof.

Suppose that we maintain a 3-approximate vertex cover C at all times. This can be done deterministically in $O(\log^3 n)$ worst-case update time using the algorithm of [BHN17]. If at any point during the updates $|C| > 12\mu'$ then we know $\mu(G) \geq |C|/6 > 2\mu'$ so even returning the empty matching satisfies the proposition. Otherwise, we can construct the core graph in $O(|C^2|) = O(\mu'^2)$ time, find a maximum matching of it in $O(\mu'^2/\varepsilon)$ time using [Proposition 2.1](#) and amortize the cost

over the next $\varepsilon\mu'$ updates where the maximum matching size can only change by a small amount. As a result, we get a $(1 - \varepsilon)$ -approximation with $O(\mu'/\varepsilon^2 + \log^3 n)$ amortized update-time. The update time can also be made worst-case by standard techniques. See [GP13] for more details. \square

The following algorithm also follows from [GP13] which is helpful when Δ is small.

Proposition 2.4 ([GP13]). *Let Δ be an upper bound on the maximum degree of a fully dynamic graph G at all times. For any $\varepsilon > 0$, one can maintain a $(1 - \varepsilon)$ -approximate maximum matching of G in worst-case update-time $O(\Delta/\varepsilon^2)$.*

3 Hierarchical Edge-Degree Constrained Subgraphs (HEDCS)

In this section, we focus on HEDCS, give a few useful definitions for it, and prove some of its key properties. The dynamic algorithm for maintaining an HEDCS is then presented in [Section 4](#).

3.1 Basic Properties of HEDCS

One useful property of any β -HEDCS $_k$ is that its maximum degree is $\leq \beta - 1$, regardless of the value of k . This sparsity of HEDCS is particularly useful for maintaining it.

Observation 3.1. *Every β -HEDCS $_k$ H has maximum degree at most $\beta - 1$.*

Proof. Fix a hierarchical decomposition $H_1 \subseteq \dots \subseteq H_k$ of H . We define the level of any edge $e \in H$ to be the unique value of $i \in [k]$ such that $e \in H_i \setminus H_{i-1}$.

Towards contradiction suppose $\deg_H(v) \geq \beta$ for some vertex v . Take an arbitrary edge e of v with the highest level. Suppose that the level of e is i , i.e., $e \in H_i \setminus H_{i-1}$. It holds that $\deg_{H_i}(e) \geq \deg_{H_i}(v) + 1 = \deg_H(v) + 1 \geq \beta + 1$ contradicting property (i) of HEDCS. \square

While we do not use the next two simple observations in our proofs, it might be instructive to state them here regardless.

Observation 3.2. *For any integers $k' \geq k$ and β , any β -HEDCS $_k$ H is also a β -HEDCS $_{k'}$.*

Proof. Let H_1, \dots, H_k be a hierarchical decomposition of H and let $H_{k+1} = \emptyset, \dots, H_{k'} = \emptyset$. It is easy to see that $H_1, \dots, H_{k'}$ satisfies properties of β -HEDCS $_{k'}$, thus H is also a β -HEDCS $_{k'}$. \square

Observation 3.3. *For any parameters $k \geq 1$ and $\beta \geq 2$, any graph G has a β -HEDCS $_k$.*

Proof. Any graph G is known to have a β -EDCS for any $\beta \geq 2$ [BS15b]. Since a β -EDCS is equivalent to a β -HEDCS $_1$, the statement follows from [Observation 3.2](#). \square

3.2 Approximation Ratio of HEDCS: Basic Definitions

We now turn to measuring the approximation ratio guaranteed by an HEDCS, and give a few definitions for this purpose.

For any integers $\beta > \beta^- \geq 1$, and $k \geq 1$ we define a number $f(k, \beta, \beta^-)$ that plays a crucial role in bounding the approximation ratio achieved by a β -HEDCS $_k$ (for now think of β^- as a number that is very close to β but is smaller, we will specify its value soon). In the definition below, by a *bipartite* HEDCS we mean an HEDCS defined on a bipartite graph.

Definition 3.4 (Function f). For integers $k \geq 1$ and $\beta > \beta^- \geq 1$, let $f(k, \beta, \beta^-)$ be the largest number such that in every bipartite β -HEDCS $_k$ with vertex parts P and Q and $\geq \beta^-|P|/2$ edges,

$$|Q| \geq f(k, \beta, \beta^-)|P|.$$

Based on f , we define a function α that is more convenient to use for our approximations:

Definition 3.5 (Function α). For any integers $k \geq 1$ and $\beta > \beta^- \geq 1$ we define

$$\alpha(k, \beta, \beta^-) = \frac{2f(k, \beta, \beta^-)}{2f(k, \beta, \beta^-) + 1}.$$

Let us now relate HEDCS to the value of function α defined above.

Proposition 3.6 (Approximation guarantee of HEDCS). Let H and U be subgraphs of a graph G , let $\beta \geq 2$ be any integer, and suppose that H is a β -HEDCS $_k$ of $G \setminus U$. Then:

- If G is bipartite, $\mu(H \cup U) \geq \alpha(k, \beta, \beta - 1)\mu(G)$.
- If G is not necessarily bipartite, and $\beta \geq c(\beta'k)^2 \log(\beta'k)$ for some integer β' and a sufficiently large constant $c \geq 1$, then $\mu(H \cup U) \geq \alpha(k, \beta' + 2k - 1, \beta')\mu(G)$.

Subgraph U in [Proposition 3.6](#) will be important for our particular application. But it would be instructive to let $U = \emptyset$. Doing so, note that we get $\mu(H) \geq \alpha(k, \beta, \beta - 1)\mu(G)$ if H is a β -HEDCS $_k$ of a bipartite graph G . Hence, H is guaranteed to include an $\alpha(k, \beta, \beta - 1)$ -approximate matching of its base graph G in this case. The same can be done for general graphs, albeit with a slightly different dependence on the parameter β .

The proof of [Proposition 3.6](#) is very similar to the proof of [\[AB19\]](#) that an EDCS obtains a near 2/3-approximation. We provide the details of the needed modifications in [Appendix B](#).

So it only remains to lower bound the value of function $\alpha(k, \beta, \beta^-)$ for various k , β , and β^- to understand the approximation ratio achieved via HEDCS. Let us start with a trivial bound.

Observation 3.7. For any $k \geq 1$ and any $\beta > \beta^- \geq 1$, $\alpha(k, \beta, \beta^-) \geq \frac{\beta^-/(\beta-1)}{\beta^-/(\beta-1)+1}$.

Proof. Let H be a β -HEDCS $_k$ with vertex parts P and Q and at least $|H| \geq \beta^-|P|/2$ edges. Since the maximum degree in H is at most $\beta - 1$ by [Observation 3.1](#), we have $|Q|(\beta - 1) \geq |H| \geq \beta^-|P|/2$. Rearranging the terms, we get $|Q| \geq \frac{\beta^-}{2(\beta-1)}|P|$ and thus $f(k, \beta, \beta^-) \geq \frac{\beta^-}{2(\beta-1)}$. As such, we get

$$\alpha(k, \beta, \beta^-) = \frac{2f(k, \beta, \beta^-)}{2f(k, \beta, \beta^-) + 1} \geq \frac{2 \cdot \frac{\beta^-}{2(\beta-1)}}{2 \cdot \frac{\beta^-}{2(\beta-1)} + 1} \geq \frac{\beta^-/(\beta-1)}{\beta^-/(\beta-1)+1}. \quad \square$$

[Observation 3.7](#) implies that $\alpha(k, \beta, \beta - 1) \geq 1/2$ for any $\beta > \beta^- \geq 1$ and $k \geq 1$. From [Proposition 3.6](#), we thus get that a β -HEDCS $_k$ for every choice of $k \geq 1$ and $\beta \geq 2$ includes an at least 1/2-approximation for bipartite graphs. For general graphs too, [Observation 3.7](#) and [Proposition 3.6](#) together imply that by increasing β , the approximation ratio of a β -HEDCS $_k$ can get arbitrarily close to at least 1/2-approximation.

As we will see, however, much better lower bounds can be proven for $\alpha(k, \beta, \beta^-)$ when k is moderately small. In particular, by adapting techniques from [\[AB19\]](#) one can show $f(1, \beta, \beta^-)$

gets arbitrarily close to 1 if β is large and β^- is close to β . This implies that $\alpha(1, \beta, \beta^-)$ can get arbitrarily close to $2/3$, recovering the $2/3$ -approximation guarantee of EDCS.

The analysis of function $\alpha(k, \beta, \beta^-)$, however, is more challenging for $k > 1$. In [Section 3.3](#) we give an LP-based approach that can lower bound $\alpha(k, \beta, \beta^-)$ for moderately small values of k and β . We use the bounds achieved by this approach for our claimed approximations for $k \in \{2, 3\}$. Later in [Appendix A](#), we present a different approach for bounding α , implying that for $k = O(1/\varepsilon)$ and an appropriate β , the approximation guarantee is $\frac{1}{2} + \Omega_\varepsilon(1)$, i.e., strictly better than half.

3.3 Approximation Ratio of HEDCS: A Factor Revealing LP

In this section, we show how to lower bound the value of $f(k, \beta, \beta^-)$ (and thus $\alpha(k, \beta, \beta^-)$) by running a linear program (LP). This LP is particularly useful when the values of β and k are not too large. In particular, we use this LP to reveal the approximation factor of our algorithm for $k = 2$ and $k = 3$.

The LP is written based on three parameters β , β^- , and k . Let us start with a number of definitions. Define sets $\mathcal{P} = \{0, \dots, \beta\}^k$ and $\mathcal{Q} = \{0, \dots, \beta\}^k$. For any $p \in \mathcal{P}$ (resp. $q \in \mathcal{Q}$) and any $i \in [k]$, we use p_i (resp. q_i) to denote the i -th entry of p (resp. q). We define E_{LP} to denote all triplets $(p, q, j) \in \mathcal{P} \times \mathcal{Q} \times [k]$ such that $\sum_{i=1}^j p_i + q_i \leq \beta$.

The LP has four types of variables. First, for any $(p, q, j) \in E_{LP}$ we have a variable $x(p, q, j)$. Second, for any $p \in \mathcal{P}$ we have a variable $n_P(p)$. Third, for any $q \in \mathcal{Q}$ we have a variable $n_Q(q)$. Fourth, we have a single variable r that is also the objective value to be minimized.

The LP can now be formalized as follows; we use $LP(k, \beta, \beta^-)$ to denote its optimal value.

$$\begin{aligned} & \text{minimize} && r \\ & \text{subject to} && n_P(p) \cdot p_j = \sum_{q:(p,q,j) \in E_{LP}} x(p, q, j) \quad \text{for all } p \in \mathcal{P} \text{ and } j \in [k] \\ & && n_Q(q) \cdot q_j = \sum_{p:(p,q,j) \in E_{LP}} x(p, q, j) \quad \text{for all } q \in \mathcal{Q} \text{ and } j \in [k] \\ & && \sum_{q \in \mathcal{Q}} n_Q(q) = r \\ & && \sum_{p \in \mathcal{P}} n_P(p) = 1 \\ & && \sum_{(p,q,j) \in E_{LP}} x(p, q, j) \geq \beta^-/2 \\ & && x(p, q, j) \geq 0 \quad \text{for all } (p, q, j) \in E_{LP} \\ & && n_Q(q) \geq 0, n_P(p) \geq 0 \quad \text{for all } p \in \mathcal{P} \text{ and } q \in \mathcal{Q}. \end{aligned}$$

In the next lemma, we show that $LP(k, \beta, \beta^-)$ lower bounds the value of $f(k, \beta, \beta^-)$.

Lemma 3.8. *For any $k \geq 1$, β , and β^- , we have $f(k, \beta, \beta^-) \geq LP(k, \beta, \beta^-)$.*

Proof. Suppose that $f(k, \beta, \beta^-) = \rho$. From the definition of $f(k, \beta, \beta^-)$, we get that there exists a bipartite β -HEDCS $_k$ H with vertex parts P and Q and at least $\beta^-|P|/2$ edges, such that $|Q| = \rho|P|$. Based on this graph H , we construct a feasible solution to the LP, and show that its objective value is ρ . This clearly suffices to prove $f(k, \beta, \beta^-) \geq LP(k, \beta, \beta^-)$.

Let (H_1, \dots, H_k) be a hierarchical decomposition of H satisfying β -HEDCS $_k$ constraints. Let $s = (s_1, \dots, s_k)$ be a vector with each $s_i \in \{0, \dots, \beta\}$. We say a vertex v in H has *degree-profile* s if for any $i \in [k]$, $\deg_{H_i \setminus H_{i-1}}(v) = s_i$. For any $p \in \mathcal{P}$, we use $P(p)$ to denote the subset of vertices in part P of H that have degree-profile p . Similarly, for any $q \in \mathcal{Q}$, we use $Q(q)$ to denote the subset of vertices in part Q of H with degree-profile q . For any $p \in \mathcal{P}$, $q \in \mathcal{Q}$, and $j \in [k]$, we use $H(p, q, j)$ to denote the subset of edges of H that belong to $H_j \setminus H_{j-1}$, with the P -endpoint having degree-profile p and the Q -endpoint having degree profile q .

Consider the following values for the variables of the LP:

- For any $p \in \mathcal{P}$, we set $n_P(p) = |P(p)|/|P|$.
- For any $q \in \mathcal{Q}$, we set $n_Q(q) = |Q(q)|/|P|$. (Note that the denominator is $|P|$ and not $|Q|$.)
- For any $(p, q, j) \in E_{LP}$ we set $x(p, q, j)$ to be $|H(p, q, j)|/|P|$.
- We set $r = |Q|/|P| = \rho$.

Let us now verify that this is a feasible solution for the LP.

For the LHS of the first constraint, we have $n_P(p) \cdot p_j = \frac{|P(p)|p_j}{|P|}$ and for the RHS, we have $\sum_{q:(p,q,j) \in E_{LP}} |H(p, q, j)|/|P|$. The $|P|$ factors cancel out from both sides, and we just need to show

$$|P(p)|p_j = \sum_{q:(p,q,j) \in E_{LP}} |H(p, q, j)|.$$

The LHS is the number of vertices in P with degree-profile p times p_j . Since every vertex with degree-profile p by definition has exactly p_j edges in $H_j \setminus H_{j-1}$, the LHS counts the number of edges of $H_j \setminus H_{j-1}$ connected to vertices of P with degree-profile p . Note from definition of $H(p, q, j)$ that the RHS counts exactly the same quantity as we sum over all possible degree-profiles in the Q -side and thus count all the $H_j \setminus H_{j-1}$ edges with the P -endpoint having degree-profile p .

The second constraint can be verified to be satisfied in exactly the same way as the first.

For the LHS of the third constraint, we have $\sum_{q \in \mathcal{Q}} n_Q(q) = \sum_{q \in \mathcal{Q}} |Q(q)|/|P| = \frac{1}{|P|} \sum_{q \in \mathcal{Q}} |Q(q)|$. Since every vertex in Q has a unique degree-profile, the sum equals $|Q|$. Hence, the LHS of the third constraint equals $|Q|/|P| = \rho$. Since we set $r = \rho$, the third constraint is also satisfied.

For the LHS of the fourth constraint, we have $\sum_{p \in \mathcal{P}} n_P(p) = \sum_{p \in \mathcal{P}} |P(p)|/|P| = \frac{1}{|P|} \sum_{p \in \mathcal{P}} |P(p)|$. The sum counts the number of vertices in P , thus this indeed equals one as required by the LP.

For the fifth constraint, observe that the LHS equals $\frac{1}{|P|} \sum_{(p,q,j) \in E_{LP}} |H(p, q, j)|$. We claim that $\frac{1}{|P|} \sum_{(p,q,j) \in E_{LP}} |H(p, q, j)| \geq |H|/|P|$. Combined with our discussion of the first paragraph of the proof that $|H| \geq \beta^-|P|/2$, this suffices to prove that the fifth constraint holds. To prove this claim, take an edge $e \in H$, and let (p, q, j) be such that $e \in H(p, q, j)$. We show that $(p, q, j) \in E_{LP}$ which means the sum $\sum_{(p,q,j) \in E_{LP}} |H(p, q, j)|$ counts each edge of H at least once, proving the claim. From the definition of degree-profiles, it can be confirmed that $\deg_{H_j}(e) = \sum_{i=1}^j p_i + q_i$; now since $e \in H_j \setminus H_{j-1}$, from property (i) of k -HEDCS $_\beta$, we get that $\deg_{H_j}(e) \leq \beta$. Hence, $\sum_{i=1}^j p_i + q_i \leq \beta$ and thus $(p, q, j) \in E_{LP}$ by definition of E_{LP} .

Finally, the non-negativity constraints can be easily verified to hold since $P(q), Q(q), H(p, q, j)$, and P are all sets and hence have non-negative size. \square

k	β	β^-	$f(k, \beta, \beta^-)$	$\alpha(k, \beta, \beta^-)$	Note
2	220	217	$\geq .780$	$\geq .609$	Used for $k = 2$ and general graphs in Theorem 1.1 .
2	142	141	$\geq .789$	$\geq .612$	Used for $k = 2$ and bipartite graphs in Theorem 1.1 .
3	47	42	$\geq .569$	$\geq .532$	Used for $k = 3$ and general graphs in Theorem 1.1 .
3	35	34	$\geq .645$	$\geq .563$	Used for $k = 3$ and bipartite graphs in Theorem 1.1 .

Table 2: Lower bounds on the values of $f(k, \beta, \beta^-)$ and $\alpha(k, \beta, \beta^-)$ obtained via $LP(k, \beta, \beta^-)$.

From [Lemma 3.8](#), we get that it suffices to run $LP(k, \beta, \beta^-)$ to lower bound the value of $f(k, \beta, \beta^-)$, and thus $\alpha(k, \beta, \beta^-)$, which determines the approximation ratio of HEDCS for different parameters. [Table 2](#) gives some of these results that we use in our approximation guarantees.¹

4 Maintaining a Hierarchical EDCS in Fully Dynamic Graphs

In this section, we describe an algorithm that maintains a β -HEDCS $_k$ in fully dynamic graphs efficiently (the maintained structure actually deviates slightly from a β -HEDCS $_k$ since the updates are handled lazily, but the matching maintained is approximately as large as that of a β -HEDCS $_k$).

The formal guarantee of the algorithm is as follows.

Theorem 4.1. *Let $k \geq 0$ and $\beta \geq 2$ be integers, and let $\varepsilon \in (0, 1/12)$. Let G be an n -vertex fully dynamic graph with Δ and m being fixed upper bounds on the maximum degree and the number of edges of G . Provided that there are at least $\Omega(m)$ edge updates, there is an algorithm that maintains a matching M of G such that:*

- **Update-time:** *Each update takes $\min\{\Delta^{\frac{1}{k+1}}, m^{\frac{1}{2(k+1)}}\} \cdot \text{poly}(\varepsilon^{-1}\beta k \log n)$ time w.h.p.*
- **Approximation:** *Suppose $k \geq 1$ and let α be as in [Definition 3.5](#). If graph G is bipartite, then at any point, w.h.p., it holds that*

$$|M| \geq (\alpha(k, \beta, \beta - 1) - O(\varepsilon)) \cdot \mu(G).$$

If G is not necessarily bipartite and β is such that $\beta \geq c(\beta'k)^2 \log(\beta'k)$ for some integer $\beta' \geq 1$ and a large enough constant $c \geq 1$, then at any point, w.h.p.,

$$|M| \geq (\alpha(k, \beta' + 2k - 1, \beta') - O(\varepsilon)) \cdot \mu(G).$$

If $k = 0$, then at any point, w.h.p., $|M| \geq (1 - O(\varepsilon)) \cdot \mu(G)$.

For ease of exposition, we have decided not to optimize the $\text{poly}(\varepsilon^{-1}\beta k \log n)$ factor in the update-time guarantee of [Theorem 4.1](#). Note, however, that for our final claimed bounds in [Table 1](#) we will only need $\varepsilon^{-1}\beta k \log n = \text{poly}(\log n)$ and so the update time is $\tilde{O}(\min\{\Delta^{\frac{1}{k+1}}, m^{\frac{1}{2(k+1)}}\})$.

In [Section 4.2](#) we present an algorithm that we show obtains the same guarantees as those stated in [Theorem 4.1](#), except that its *amortized* update-time bound is $\tilde{O}(\Delta^{\frac{1}{k+1}})$. We then show in [Section 4.6](#) how the algorithm can be slightly modified to turn this amortized bound to worst-case

¹The code is available upon request.

using standard techniques. Finally, in Section 4.7 we show how to ignore some of the edges of the graph as they are inserted, such that the maximum degree remains $O(\sqrt{m}/\varepsilon)$ without changing the maximum matching of the graph by much. From this, we get the claimed $\tilde{O}(\min\{\Delta^{\frac{1}{k+1}}, m^{\frac{1}{2(k+1)}}\})$ worst-case update-time guarantee of Theorem 4.1.

4.1 High-Level Overview of the Algorithm

Our starting point is a pre-processing algorithm where we construct three sequences of subgraphs G_i, U_i, H_i of G satisfying certain structures. We maintain these subgraphs upon updates too, but in a lazy fashion. The following properties, in particular, continue to hold throughout the algorithm:

- (i) $\emptyset = G_0 \subseteq G_1 \subseteq \dots \subseteq G_{k+1} = G$,
- (ii) $G = U_1 \supseteq U_2 \supseteq \dots \supseteq U_{k+1}$, and
- (iii) $\emptyset = H_0 \subseteq H_1 \subseteq \dots \subseteq H_k$.

Subgraphs G_i are simply edge-sampled random subgraphs of G . Specifically, for each edge e we draw a real $\pi_e \sim \text{Unif}[0, 1]$ independently (upon arrival of the edge) and e appears in G_i iff $\pi_e \leq p_i$ for some parameters $p_1 \leq \dots \leq p_{k+1} = 1$ (defined in Algorithm 1).

Each subgraph H_i , at any time, will be a β -HEDCS $_i$ of graph $(G \setminus G_i) \setminus U_{i+1}$ with (H_1, \dots, H_i) being its hierarchical decomposition (we prove this in Lemma 4.9). Subgraph U_{i+1} , in particular, will include all (H_i, β) -underfull edges in $G \setminus G_i$ at any time. This immediately proves Property (ii) of β -HEDCS $_i$ for H_i since no (H_i, β) -underfull edge is in $(G \setminus G_i) \setminus U_{i+1}$ (as they are all in U_{i+1}).

The final matching M that we output, and lazily maintain, is an (almost) maximum matching of the edges of $H_k \cup U_{k+1}$ that are present in the graph. By our discussion above, H_k will be a β -HEDCS $_k$ of $(G \setminus G_k) \setminus U_{k+1}$, thus by the guarantee of HEDCS (Proposition 3.6), $\mu(H_k \cup U_{k+1})$ should well-approximate $\mu(G \setminus G_k) \approx \mu(G)$ where the last almost-equality comes from the fact that G_k includes $p_k = o(1)$ fraction of the edges of G randomly, hence their removal does not change the matching size by much. It would be useful to note here that during the updates, some of the edges of each subgraph H_i may get removed from the graph. We do not remove these edges from H_i immediately. Rather, we recompute H_i frequently enough to ensure that at any point only a small number of its edges have been removed from the graph. We note that the output matching M will not use the edges of H_k that are deleted and will always be a proper matching of G .

Remark 4.2 (Assumptions). *We make the following assumption throughout the rest of Section 4 which all hold without loss of generality (w.l.o.g.).*

We assume that $\mu(G) \geq \mu' = 10^3 \varepsilon^{-1} k \log n$ throughout the whole sequence of updates. This assumption comes w.l.o.g. since by plugging this value of μ' in Proposition 2.3, we get an algorithm with worst-case update time $O(\log^3 n + k \log n / \varepsilon^3)$ that already maintains a $(1 - \varepsilon)$ -approximate matching of G whenever $\mu(G) < \mu'$.

We assume $\Delta \geq 15 \log n / \varepsilon$ as otherwise Proposition 2.4 already gives a $(1 - \varepsilon)$ -approximation with $O(\log n / \varepsilon)$ worst-case update time.

We assume k is small enough that $\Delta^{\frac{1}{k+1}} \geq 15 \log n / \varepsilon$. If not, we can pick a smaller k that satisfies it and additionally $\Delta^{\frac{1}{k+1}} = O(\log n / \varepsilon)$. Note that by picking a smaller k the approximation improves and the update-time of Theorem 4.1 would be $\text{poly}(\varepsilon^{-1} \beta k \log n)$ when $\Delta^{\frac{1}{k+1}} = O(\log n / \varepsilon)$.

4.2 The Formal Algorithm

We start with the pre-processing algorithm described as [Algorithm 1](#).

Algorithm 1: PREPROCESSING(G)

- 1 **Input:** A graph $G = (V, E)$ with maximum degree bounded by Δ .
 - 2 **Parameters:** Integers $\beta \geq 1$ and $k \geq 1$ and real $\varepsilon \in (0, 1)$.
 - 3 For any $i \in [k]$ let $p_i := \varepsilon \cdot \Delta^{\frac{i}{k+1} - 1}$ and let $p_{k+1} = 1$. ▷ See [Claim 4.3](#).
 - 4 Draw a random ranking π , i.e., for each edge $e \in E$ draw $\pi_e \sim \text{Unif}[0, 1]$ independently.
 - 5 For any $i \in [k + 1]$ let G_i be the subgraph of G including all edges e with $\pi_e \leq p_i$.
 - 6 Let $U_1 \leftarrow G$, let $G_0 \leftarrow \emptyset$, and let $H_0 \leftarrow \emptyset$.
 - 7 Run COMPUTELAYERS(1) (formalized as [Algorithm 2](#)) to generate subgraphs H_1, \dots, H_{k+1} , U_2, \dots, U_{k+1} , integers μ_1, \dots, μ_{k+1} , and the output matching M .
-

Subroutine COMPUTELAYERS(j) formalized below in [Algorithm 2](#) is called both in pre-processing [Algorithm 1](#) (for $j = 1$) and during the update time (for various $j \in [k + 1]$).

Algorithm 2: COMPUTELAYERS(j).

- 1 **for** i in j, \dots, k **do**
 - 2 $\mu_i \leftarrow \tilde{\mu}(G_i)$ ▷ See paragraph “Maintaining $\tilde{\mu}(G_i)$ ” below.
 - 3 $H_i \leftarrow \text{ADDAYER}(U_i \cap G_i, H_{i-1}, \mu_i)$. ▷ Formalized as [Algorithm 3](#).
 - 4 Let U_{i+1} be the graph including every edge in $U_i \setminus G_i$ that is (H_i, β) -underfull.
 - 5 $\mu_{k+1} \leftarrow \tilde{\mu}(G_{k+1})$ ▷ See paragraph “Maintaining $\tilde{\mu}(G_i)$ ” below.
 - 6 $M \leftarrow$ a $(1 - \varepsilon)$ -approximate max matching of $(H_k \cup U_{k+1}) \cap G$ computed via [Proposition 2.1](#).
-

The next subroutine ADDAYER(Γ, H_{i-1}, μ_i), which is called only from [Algorithm 2](#), starts with $H_i \leftarrow H_{i-1}$, then iterates over the edges of Γ in the increasing order of ranks π (the same rank function as in [Algorithm 1](#)), adds each encountered underfull edge to H_i (and removes their incident overfull edges that belong to $H_i \setminus H_{i-1}$, if any). After the algorithm iterates over sufficiently many edges without detecting any underfull edges, it returns H_i .

Algorithm 3: ADDAYER(Γ, H_{i-1}, μ_i). Input Γ will be $U_i \cap G_i$ when called.

- 1 Let $H_i \leftarrow H_{i-1}$, $\eta \leftarrow 0$.
 - 2 Iterate over the edges of Γ in the increasing order of π . Upon visiting an edge $e = (u, v)$:
 - 3 $\eta \leftarrow \eta + 1$.
 - 4 **if** e is (H_i, β) -underfull **then**
 - 5 Add e to H_i .
 - 6 If there exists any (H_i, β) -overfull edge $(u, w) \in H_i \setminus H_{i-1}$ remove one arbitrarily.
 - 7 If there exists any (H_i, β) -overfull edge $(v, w) \in H_i \setminus H_{i-1}$ remove one arbitrarily.
 - 8 $\eta \leftarrow 0$.
 - 9 **if** $\eta > \lfloor |\Gamma| / (4\mu_i \beta^2 + 1) \rfloor$ **then**
 - 10 **return** H_i . ▷ [Corollary 4.8](#) guarantees that we reach this line eventually.
-

Handling edge updates: We would like to maintain the same output as that of [Algorithm 1](#). However, to optimize the update-time we handle most edge updates in a lazy fashion. Only the following trivial updates are done immediately upon insertion/deletion:

- Upon insertion of an edge e to G , we add e to any graph U_{i+1} where e is (H_i, β) -underfull and $e \notin G_i$. Moreover, we immediately draw the rank $\pi_e \sim \text{Unif}[0, 1]$ for e and for any $i \in [k+1]$ with $\pi_e \leq p_i$, we add e to graph G_i .
- Upon deletion of an edge e from G , we immediately remove e from any of G_1, \dots, G_{k+1} , U_1, \dots, U_{k+1}, M that includes e . Note that we do *not* remove e from H_1, \dots, H_k .

By storing all the graphs in the adjacency-list format and storing each adjacency-list as a balanced binary search tree, these operations can easily be implemented in $O(k \log n)$ time per update.

As discussed, the more time-consuming updates are done lazily. Particularly, for any $i \in [k+1]$ we keep a counter c_i that is initially zero after the pre-processing step. Then upon every update we set $c_i \leftarrow c_i + 1$ for all $i \in [k+1]$. For any $j \in [k+1]$ immediately after condition $c_j \geq \frac{\varepsilon}{k} \cdot \frac{\mu_j + 1}{p_j}$ holds, we reset $c_j, c_{j+1}, \dots, c_{k+1}$ to zero and call subroutine COMPUTELAYERS(j) of [Algorithm 2](#) to recompute $H_j, \dots, H_k, U_{j+1}, \dots, U_{k+1}$, and M_j, \dots, M_{k+1} . (If for multiple j the condition $c_j \geq \frac{\varepsilon}{k} \cdot \frac{\mu_j + 1}{p_j}$ holds at once, we apply the procedure on the minimum such j .)

Maintaining $\tilde{\mu}(G_i)$: In [Algorithm 2](#) we need to compute the size of a maximal matching $\tilde{\mu}(G_i)$ for any $i \in \{j, \dots, k+1\}$. This can of course be done in time $|G_i|$ for each i by iterating over the edges of G_i and constructing a maximal matching greedily. However, we need a more efficient algorithm. To do this, for each $i \in [k+1]$ we maintain a maximal matching of G_i after each edge update along with its size. This can be done in $\text{poly}(\log n)$ worst-case update-time for each G_i using [Proposition 2.2](#). Hence, whenever we call [Algorithm 2](#) we have $\tilde{\mu}(G_i)$ readily computed. That is, it can be accessed in $O(1)$ time, without the need to go over all edges of G_i to construct the maximal matching. The cost is only an additive worst-case $k \cdot \text{poly}(\log n)$ factor to the final update-time. It is worth noting that instead of $\tilde{\mu}(G_i)$, any $O(1)$ -approximation of $\mu(G_i)$ would suffice for our purpose and the maximality is not important. We decided to use maximal matching algorithms since they are already fast enough.

In [Section 4.3](#) we state some basic properties of the algorithm described above. In [Section 4.5](#) we analyze the update-time of this algorithm. Then in [Section 4.4](#) we analyze its approximation.

4.3 Basic Properties of the Algorithm of [Section 4.2](#)

In this section we prove a number of basic properties of the algorithm that we later use to analyze its update-time and approximation.

Claim 4.3. *Values of p_1, \dots, p_{k+1} are set in [Algorithm 1](#) such that they satisfy the following:*

- (i) $p_1 \leq p_2 \leq \dots \leq p_{k+1} = 1$.
- (ii) $p_1 \geq 15 \frac{\log n}{\Delta}$.
- (iii) $p_i/p_{i-1} = O(\Delta^{\frac{1}{k+1}}/\varepsilon)$ for all $i \geq 2$.
- (iv) $p_k \leq \varepsilon$.
- (v) $\frac{p_i - p_{i-1}}{1 - p_{i-1}} \geq p_i/2$ for all $i \geq 2$.

Proof. Note from [Algorithm 1](#) that $p_i := \varepsilon \cdot \Delta^{\frac{i}{k+1}-1}$ for $i \in [k]$ and $p_{k+1} = 1$. From this definition, we immediately get $p_1 \leq \dots \leq p_k \leq \varepsilon < p_{k+1} = 1$, hence proving part (i).

Property (ii) follows since $p_1 = \varepsilon \cdot \Delta^{\frac{1}{k+1}-1}$ and we assumed in Remark 4.2 that $\Delta^{\frac{1}{k+1}} \geq 15 \log n / \varepsilon$.

For Property (iii), note that $p_i/p_{i-1} \leq \Delta^{\frac{i}{k+1}-1}/(\varepsilon \Delta^{\frac{i-1}{k+1}-1}) = O(\Delta^{\frac{1}{k+1}}/\varepsilon)$.

Property (iv) trivially holds since $p_k = \varepsilon \Delta^{\frac{k}{k+1}-1} \leq \varepsilon$.

Property (v) holds since $\frac{p_i - p_{i-1}}{1 - p_{i-1}} \geq p_i - p_{i-1} \geq p_i - p_i / \Delta^{\frac{1}{k+1}} \geq p_i / 2$. (The latter holds so long as $\Delta^{\frac{1}{k+1}} \geq 2$ and recall that in Remark 4.2 we assume it is indeed much larger.) \square

Claim 4.4. *It holds at all times that:*

- (i) $G_1 \subseteq G_2 \subseteq \dots \subseteq G_{k+1} = G$.
- (ii) $\emptyset = H_0 \subseteq H_1 \subseteq H_2 \subseteq \dots \subseteq H_k$.
- (iii) $G = U_1 \supseteq U_2 \supseteq \dots \supseteq U_{k+1}$.
- (iv) For any $i \in [k]$, U_{i+1} is the set of all (H_i, β) -underfull edges in $G \setminus G_i$.
- (v) $H_i \subseteq G_i$ for all $i \in \{0, \dots, k\}$.

Proof. Property (i) follows from Claim 4.3 part (i) since G_i simply includes an edge e iff $\pi_e \leq p_i$, at all times.

For Property (ii), observe that H_i 's are only modified in Algorithm 2. Particularly H_i is obtained by calling ADDLAYER($U_i \cap G_i, H_{i-1}, \mu_i$) which only adds some of the edges of $U_i \cap G_i$ to H_{i-1} . Additionally, any time that H_i is recomputed, all of H_i, \dots, H_k are also recomputed. Hence, the property continues to hold at all times.

We prove Property (iv) by induction. The base case $i = 0$ holds trivially since U_1 is always equal to G . Now observe that we set U_{i+1} in Algorithm 2 to be the graph including (H_i, β) -underfull edges in $U_i \setminus G_i$. By induction hypothesis, U_i is the subgraph of (H_i, β) -underfull edges in $G \setminus G_{i-1}$. Hence, U_{i+1} includes every (H_i, β) underfull edge in $(G \setminus G_{i-1}) \setminus G_i = G \setminus G_i$ (by Property (i)). During the update time, we maintain the invariant that U_{i+1} is the set of all (H_i, β) -underfull edges not in G_i for every update. Hence, the property holds at all times.

For Property (iii), note that if $e \in U_{i+1}$, then it must be (H_i, β) -underfull and in $G \setminus G_i$ by Property (iv). Since $H_{i-1} \subseteq H_i$ by Property (ii), then e is also (H_{i-1}, β) -underfull and definitely in $G \setminus G_{i-1}$ (since $G_{i-1} \subseteq G_i$ by Property (i)). This means e should also belong to U_i .

We prove Property (v) by induction on i . The base case $i = 0$ trivially holds since $H_0 = \emptyset$. Now observe that subgraph H_i is obtained in Algorithm 3 by adding some edges of $U_i \cap G_i$ to H_{i-1} . This means $H_i \subseteq (G_i \cap U_i) \cup H_{i-1} \subseteq G_i \cup H_{i-1} \subseteq G_i \cup G_{i-1} = G_i$ where the last two follow from the induction hypothesis and Property (i) respectively. \square

The next observation will play an important role later in Lemma 4.9 where we argue that H_k is a β -HEDCS_k of $(G \setminus G_k) \setminus G_{k+1}$.

Observation 4.5. *By the end of every iteration of the loop in Line 2 of Algorithm 3 (and thus by the end of the whole algorithm also), the subgraph $H_i \setminus H_{i-1}$ includes no (H_i, β) -overfull edge.*

Proof. We prove by induction on the number of iterations. Initially $H_i \setminus H_{i-1}$ is empty and so the claim holds. Once we add an edge e to H_i , e must be (H_i, β) -underfull meaning that $\deg_{H_i}(e) < \beta - 1$. Hence, after inserting e to H_i , we have $\deg_{H_i}(e) < \beta + 1$ which means e is not (H_i, β) -overfull. Note, however, that adding e to H_i increases its endpoints' degrees in H_i by one. This may lead to (H_i, β) -overfull edges in $H_i \setminus H_{i-1}$ that are incident to e , but removing any one such edge from each endpoint of e ensures that $H_i \setminus H_{i-1}$ remains to have no (H_i, β) -overfull edges. \square

We will also use the following upper bound on the maximum degree of any H_i several times.

Observation 4.6. *For any $i \in \{0, \dots, k\}$, the maximum degree of H_i is at most β at all times.*

Proof. We prove by induction on i . For the base case $H_0 = \emptyset$ and so the claim clearly holds. Suppose, now, that the claim holds for H_{i-1} , we prove it for H_i . Consider a call to [Algorithm 2](#) where we set $H_i \leftarrow \text{ADDLAYER}(U_i \cap G_i, H_{i-1}, \mu_i)$. It suffices to prove that at this point, H_i has maximum degree β since until H_i gets re-computed, we may only remove edges from it.

Suppose toward contradiction that $\deg_{H_i}(v) > \beta$ for some vertex v . Since $\deg_{H_{i-1}}(v) \leq \beta$ by the induction hypothesis and $H_{i-1} \subseteq H_i$ by [Claim 4.4](#), v must have an edge $e = (v, u) \in H_i \setminus H_{i-1}$. Moreover, $\deg_{H_i}(e) = \deg_{H_i}(u) + \deg_{H_i}(v) \geq \deg_{H_i}(v) \geq \beta + 1$ and so e must be (H_i, β) -overfull. This, however, contradicts [Observation 4.5](#). Hence $\deg_{H_i}(v) \leq \beta$ for all v . \square

The following claim bounds the number of times that we encounter an (H_i, β) -underfull edge in [Algorithm 3](#) by $4\mu_i\beta^2$. The proof is based on a potential function used previously for EDCS by [BS16, ABB⁺19, AB19, Ber20] with a simple additional idea that bounds the number of edges of H_i by $2\mu_i\beta$. For completeness, we provide the full proof in [Appendix C](#).

Claim 4.7. *Algorithm 3 reaches Line 5 at most $4\mu_i\beta^2$ times.*

As an immediate corollary of [Claim 4.7](#) we get that [Algorithm 3](#) always terminates:

Corollary 4.8 (of [Claim 4.7](#)). *Algorithm 3 reaches Line 10 with probability one.*

Proof. Suppose for the sake of contradiction that the algorithm does not reach [Line 10](#). Note from [Algorithm 3](#) that this means $\eta \leq \tau$ at all times where $\tau = \lfloor |\Gamma|/(4\mu_i\beta^2 + 1) \rfloor$. From the definition of counter η in [Algorithm 3](#), we get that the algorithm encounters at least one (H_i, β) -underfull edge within every τ consecutive edges of Γ (processed in the order of π). But this means, we must encounter at least $|\Gamma|/\tau \geq 4\mu_i\beta^2 + 1$ edges that are (H_i, β) -underfull, contradicting [Claim 4.7](#). \square

4.4 The Approximation Ratio

We now analyze the size of M and prove the approximation guarantee of [Theorem 4.1](#).

Fix an arbitrary sequence of updates and suppose that we run the algorithm of [Section 4.2](#) on them. Unless otherwise stated, when we refer to a data structure of the algorithm throughout [Section 4.4](#) (such as matching M , subgraph H_i , integer μ_i , graph G , etc.) we refer to the value stored in this data structure after the whole sequence of updates has been applied.

We prove that, w.h.p., the size of M is as claimed in [Theorem 4.1](#) at the end of applying this sequence of updates. Note that since this holds for any arbitrary sequence, it also holds for any update throughout the sequence.

Our starting point is the following lemma.

Lemma 4.9. *H_k is a β -HEDCS _{k} of $(G \setminus G_k) \setminus U_{k+1}$ with (H_1, \dots, H_k) being its hierarchical decomposition.*

Proof. Let us first confirm property (i) of β -HEDCS _{k} . Observe that H_i is computed in [Algorithm 2](#) and is the output of $\text{ADDLAYER}(U_i \cap G_i, H_{i-1}, \mu_i)$. As such, by [Observation 4.5](#), $H_i \setminus H_{i-1}$ includes no (H_i, β) -overfull edges. That is, for any edge $e \in H_i$, we have $\deg_{H_i}(e) \leq \beta$ right after recomputation of H_i . Now note that any time that some subgraph H_j for $j \leq i$ is recomputed, H_i gets

recomputed too in [Algorithm 2](#). Moreover, other than these recomputations, the subgraphs H_i do not change (even if their edges are removed from the graph). Hence, property (i) of β -HEDCS $_k$ continues to hold throughout the sequence of updates.

For property (ii) of β -HEDCS $_k$, recall from [Claim 4.4](#) part (iv) that all (H_k, β) -underfull edges of $G \setminus G_k$ belong to U_{k+1} at all times. Hence, $(G \setminus G_k) \setminus U_{k+1}$ includes no (H_k, β) -underfull edges. That is, for any $e \in (G \setminus G_k) \setminus U_{k+1}$, we have $\deg_{H_k}(e) \geq \beta - 1$. \square

The only remaining problem is that some of the edges of H_k may have been deleted from the graph. Therefore, although H_k is a valid β -HEDCS $_k$ of $G \setminus U_{k+1}$, not all the edges in $H_k \cup U_{k+1}$ actually exist in the graph. The rest of this section is essentially devoted to upper bounding the number of such deleted edges.

A few definitions are in order. We use \hat{G}_i to denote subgraph G_i right after the last recomputation of μ_i and H_i , which must have been exactly c_i updates ago. We also use F_i to refer to any edge inserted or removed at least once from G_i during the last c_i updates.

A key claim to bounding the number of deleted edges of H_k , say, is the following:

Lemma 4.10. *With probability $1 - 1/n^4$, $|F_1| + \dots + |F_{k+1}| \leq 13\epsilon\mu(G)$.*

Let us first see why [Lemma 4.10](#) proves the approximation guarantee of [Theorem 4.1](#).

Proof of approximation guarantee of Theorem 4.1. Consider the last time that we recomputed M ; this must have been c_{k+1} updates ago. Let us use H'_i , U'_i , and G' to denote H_i , U_i , and G right before recomputation of M . Observe that we set M to be a $(1 - \epsilon)$ -approximate maximum matching of $(H'_k \cup U'_{k+1}) \cap G'$. We show that

$$\begin{aligned} |M| &\geq (1 - \epsilon)\mu((H'_k \cup U'_{k+1}) \cap G') - |G' \setminus G| \\ &\geq (1 - \epsilon)\mu((H_k \cup U_{k+1}) \cap G) - |G' \setminus G| - |G \setminus G'| - |U_{k+1} \setminus U'_{k+1}| - |H_k \setminus H'_k|. \end{aligned} \quad (1)$$

The first inequality holds since M at the end of the sequence is its last value computed, which is a $(1 - \epsilon)$ -approximate maximum matching of $(H'_k \cup U'_{k+1}) \cap G'$, excluding its edges that have been removed from the graph during the last c_{k+1} updates. The second bound holds since by replacing G' with G , H'_k with H_k , and U'_{k+1} with U_{k+1} , we may only add $(G \setminus G') \cup (H_k \setminus H'_k) \cup (U_{k+1} \setminus U'_{k+1})$ edges to the matching, which is then canceled out with the subtracted terms.

Now note that $H'_k = H_k$ where recall we use H_k to denote the final value of H_k at the end of the update sequence. This is correct because M gets recomputed any time that COMPUTELAYERS(j) is called for *any* value of j . Hence, since the last recomputation of M , we have not called COMPUTELAYERS(j) for any j and so H_k must have remained unchanged. Note also that any edge in $U_{k+1} \setminus U'_{k+1}$, $G \setminus G'$, or $G' \setminus G$ must have been updated during the last c_{k+1} updates. Since F_{k+1} by definition includes any edge of $G_{k+1} = G$ updated during the last c_{k+1} updates, (1) gives

$$|M| \geq (1 - \epsilon)\mu((H_k \cup U_{k+1}) \cap G) - 3|F_{k+1}|. \quad (2)$$

Next, observe that $U_{k+1} \subseteq G$ by [Claim 4.4](#). Moreover, if an edge $e \in H_i \setminus H_{i-1}$ does not belong to G , then e must have been updated during the last c_i updates and must belong to G_i which together imply $e \in F_i$. Combined with $H_1 \subseteq \dots \subseteq H_k$ of [Claim 4.4](#), we get $|H_k \setminus G| \leq |F_1| + \dots + |F_k|$. Now combined with (2) this implies

$$|M| \geq (1 - \epsilon)\mu(H_k \cup U_{k+1}) - 3|F_{k+1}| - (|F_1| + \dots + |F_k|)$$

$$\begin{aligned} &\geq (1 - \varepsilon)\mu(H_k \cup U_{k+1}) - 3(|F_1| + \dots + |F_{k+1}|) \\ &\geq (1 - \varepsilon)\mu(H_k \cup U_{k+1}) - 39\varepsilon\mu(G). \end{aligned} \quad (\text{Holds w.h.p. by Lemma 4.10.}) \quad (3)$$

This immediately proves the approximation guarantee of [Theorem 4.1](#) for $k = 0$ since at all times $U_{k+1} = U_1 = G$. So let us now focus on $k \geq 1$.

Observe that by [Lemma 4.9](#) H_k is a β -HEDCS $_k$ of $(G \setminus G_k) \setminus U_{k+1}$ and so applying [Proposition 3.6](#) and noting from the statement of [Theorem 4.1](#) that $\beta = c(\beta')^2 \log(\beta')$, we get

$$\mu(H_k \cup U_{k+1}) \geq \alpha(k, \beta, \beta - 1) \cdot \mu(G \setminus G_k) \quad \text{for bipartite } G, \quad (4)$$

$$\mu(H_k \cup U_{k+1}) \geq \alpha(k, \beta', \beta' - k) \cdot \mu(G \setminus G_k) \quad \text{for general } G. \quad (5)$$

To complete the proof, note that G_k includes each edge of G independently with probability $p_k \leq \varepsilon$ by [Claim 4.3](#) part (v). This means that fixing a maximum matching of G , only ε fraction of its edges appear in G_k in expectation. This bound also holds with high probability by a Chernoff bound noting that $\mu(G) \geq 10^3 \varepsilon^{-1} k \log n$ from [Remark 4.2](#). As such, we get that with probability, say, $1 - 1/n^4$, $\mu(G \setminus G_k) \geq (1 - \varepsilon)\mu(G)$. Plugging this into (4) for bipartite graphs and (5) for general graphs, and then applying (3), we get the claimed lower bound of [Theorem 4.1](#) on $|M|$. \square

Toward proving [Lemma 4.10](#), we prove two auxiliary claims first.

Claim 4.11. *It holds that $\mu(G) \geq \frac{2}{3(k+1)}(\mu_1 + \dots + \mu_{k+1}) - \sum_{i=1}^{k+1} |F_i|$ with probability 1.*

Proof. Recall that μ_i is the size of a maximal matching of \hat{G}_i . Let us denote this maximal matching by M_i . Let us also use $M_{1,k+1}$ to denote $M_1 \cup \dots \cup M_{k+1}$. Observe that if an edge of M_i does not belong to G , then it must be in F_i . This means $\mu(G) \geq \mu(M_{1,k+1}) - \sum_{i=1}^{k+1} |F_i|$.

Let us now lower bound $\mu(M_{1,k+1})$. To any edge $e \in M_{1,k+1}$ we assign fractional value $x_e := h_e/(k+1)$ where h_e is the number of matchings M_1, \dots, M_{k+1} that include e . It can be confirmed that x is a valid fractional matching of $M_{1,k+1}$. On the other hand, this fractional matching has size exactly $\frac{|M_1| + \dots + |M_{k+1}|}{k+1} = \frac{\mu_1 + \dots + \mu_{k+1}}{k+1}$. Any general graph has an integral matching of size at least $2/3$ times the size of any of its fractional matchings. Hence, $\mu(M_{1,k+1}) \geq \frac{2}{3(k+1)}(\mu_1 + \dots + \mu_{k+1})$. Combined with the bound of the previous paragraph, we thus get

$$\mu(G) \geq \mu(M_{1,k+1}) - \sum_{i=1}^{k+1} |F_i| \geq \frac{2(\mu_1 + \dots + \mu_{k+1})}{3(k+1)} - \sum_{i=1}^{k+1} |F_i|. \quad \square$$

Next, we prove the following high probability upper bound on $|F_i|$.

Claim 4.12. *With probability $1 - 1/n^4$, it holds that $|F_i| < \frac{2\varepsilon}{k}\mu_i + 200 \log n + 4$ for all $i \in [k+1]$.*

Proof. Fix some integer $t \geq 1$. Since the update sequence is oblivious to the randomization of the algorithm, we expect exactly $p_i \cdot t$ edges of the last t updates to have rank $\leq p_i$, i.e., belong to G_i . Applying Chernoff and union bounds, we get that with probability $1 - 1/n^4$, for any $t \leq n^2$ and any $i \in [k]$ (recall that $k \leq \log n$), we have at most $p_i \cdot t + \sqrt{100(p_i \cdot t) \log n}$ updates to G_i among the last t updates in the sequence (note that this holds for all values of $p_i \cdot t$). We assume this high probability event holds and proceed to prove the claim.

By definition of F_i , any edge in F_i must have been updated in the last c_i updates, and that each of these edges must have belonged to G_i . By the discussion above, at most $p_i \cdot c_i + \sqrt{100(p_i \cdot c_i) \log n}$

edges of G_i are updated in the last c_i updates. Now observe that $c_i \leq \frac{\varepsilon}{k} \cdot \frac{\mu_i+1}{p_i} + 1$ since any time $c_i \geq \frac{\varepsilon}{k} \cdot \frac{\mu_i+1}{p_i}$, we immediately reset c_i to zero in the update algorithm. Therefore, since clearly $c_i \leq n\Delta \leq n^2$, under the high probability event of the previous paragraph,

$$\begin{aligned} |F_i| &\leq p_i \cdot \left(\frac{\varepsilon}{k} \left(\frac{\mu_i+1}{p_i}\right) + 1\right) + \sqrt{100 \left(p_i \cdot \left(\frac{\varepsilon}{k} \left(\frac{\mu_i+1}{p_i}\right) + 1\right)\right) \log n} \\ &= \frac{\varepsilon}{k} \mu_i + \frac{\varepsilon}{k} + p_i + 10 \sqrt{\left(\frac{\varepsilon}{k} \mu_i + \frac{\varepsilon}{k} + p_i\right) \log n} \\ &\leq \frac{\varepsilon}{k} \mu_i + 2 + 10 \sqrt{\left(\frac{\varepsilon}{k} \mu_i + 2\right) \log n} \quad (\text{Since } p_i \leq 1 \text{ and } \varepsilon/k \leq 1.) \\ &\leq 2\left(\frac{\varepsilon}{k} \mu_i + 2\right) + 2 \cdot 100 \log n. \end{aligned}$$

(If $\left(\frac{\varepsilon}{k} \mu_i + 2\right) \geq 10 \sqrt{\left(\frac{\varepsilon}{k} \mu_i + 2\right) \log n}$ then the first term ensures inequality, otherwise the second.)

The proof is thus complete. \square

We are now ready to prove [Lemma 4.10](#).

Proof of Lemma 4.10. From [Claim 4.12](#) we get that with probability $1 - 1/n^4$,

$$\sum_{i=1}^{k+1} |F_i| \leq \sum_{i=1}^{k+1} (2\frac{\varepsilon}{k} \mu_i + 200 \log n + 4) \leq 204(k+1) \log n + 2\frac{\varepsilon}{k} \sum_{i=1}^{k+1} \mu_i.$$

Combined with inequality $\sum_{i=1}^{k+1} \mu_i \leq \frac{3(k+1)}{2}(\mu(G) + \sum_{i=1}^{k+1} |F_i|)$ implied by [Claim 4.11](#), we get

$$\sum_{i=1}^{k+1} |F_i| \leq 204(k+1) \log n + \frac{2\varepsilon}{k} \cdot \frac{3(k+1)}{2} \left(\mu(G) + \sum_{i=1}^{k+1} |F_i| \right).$$

Noting that $k \geq 1$, we can simplify and re-arrange the terms, obtaining that

$$\sum_{i=1}^{k+1} |F_i| \leq \frac{204(k+1) \log n + 6\varepsilon\mu(G)}{1 - 6\varepsilon} \stackrel{(\varepsilon \leq 1/12)}{\leq} 408(k+1) \log n + 12\varepsilon\mu(G) < 13\varepsilon\mu(G),$$

where the last inequality follows from our assumption of [Remark 4.2](#) that $\mu(G) \geq 10^3 \varepsilon^{-1} k \log n$. \square

4.5 Update Time of the Algorithm of [Section 4.2](#)

In this section we prove the update-time bound of [Theorem 4.1](#), except that instead of a worst-case update-time, we here prove an amortized update-time bound. We then show in [Section 4.6](#) how with a small modification this can be turned into a worst-case bound.

The cost of maintaining the maximal matchings of G_1, \dots, G_{k+1} as stated before is $k \text{ poly}(\log n)$ for every update w.h.p. The “easy updates” such as removing a deleted edge from any of U_i ’s or G_i ’s can also be done in $O(k \log n)$ time as previously discussed. It only remains to analyze the cost of the “heavy updates”. That is, the calls to `COMPUTELAYERS(j)`.

Let us fix some $j \in [k+1]$ and analyze the (amortized) cost of a call to `COMPUTELAYERS(j)`. For the rest of this section, unless otherwise stated explicitly, when we refer to a data structure of the algorithm (such as H_i , μ_i , G , etc.) we refer to the value of this data structure right after the call to `COMPUTELAYERS(j)`. Note that it takes at least another $\frac{\varepsilon}{k} \cdot \frac{\mu_j+1}{p_j}$ updates until we call

$\text{COMPUTELAYERS}(j)$ for this specific value of j again. As such, we can amortize the cost of a call to $\text{COMPUTELAYERS}(j)$ over at least $\frac{\varepsilon}{k} \cdot \frac{\mu_j+1}{p_j}$ updates.² The resulting amortized update-time summed up for all $j \in [k+1]$ gives the amortized update-time of the algorithm.

Let us start with an upper bound on the running time of $\text{COMPUTELAYERS}(j)$.

Claim 4.13. *The time spent in subroutine $\text{COMPUTELAYERS}(j)$ is $\tilde{O}\left((\varepsilon^{-1} + \beta k)|U_j| + \varepsilon^{-1}|H_k|\right)$.*

Proof. We start by analyzing the calls $H_i \leftarrow \text{ADDAYER}(U_i \cap G_i, H_{i-1}, \mu_i)$ for $i \in \{j, \dots, k\}$. Note that each graph $U_i \cap G_i$ can be constructed in $\tilde{O}(|U_i|)$ time by iterating over the edges e of U_i and considering the rank π_e which determines if $e \in G_i$. Algorithm $\text{ADDAYER}(U_i \cap G_i, H_{i-1}, \mu_i)$ iterates over the edges in $U_i \cap G_i$ and for each edge that is added to H_i , we have to find out if there are any (H_i, β) -overfull connected to its endpoints. By trivially scanning all the at most β neighbors (by [Observation 4.6](#)) this can be done in $O(|U_i \cap G_i| \beta) = O(|U_i| \beta)$ time. The overall time-complexity of these calls is therefore bounded by

$$\tilde{O}(|U_j| \beta + \dots + |U_k| \beta) = \tilde{O}(|U_j| \beta k) \quad (\text{Since } U_j \supseteq \dots \supseteq U_k \text{ by } \text{Claim 4.4}.)$$

Construction of each U_{i+1} from U_i can also be done in $O(|U_i|)$ time by simply iterating over the edges and checking the edge-degree of each edge in $O(1)$ time. Since this is run for $i \in \{j, \dots, k\}$, the total time is $O(|U_j| + \dots + |U_k|) = O(|U_j|k)$.

As discussed, each μ_i takes $O(1)$ time to compute, hence μ_j, \dots, μ_{k+1} take $O(k)$ time to compute.

The final step is to run [Proposition 2.1](#) to find a $(1 - \varepsilon)$ -approximation of $(H_k \cup U_{k+1}) \cap G$. We first construct graph $(H_k \cup U_{k+1})$ in $O(|H_k| + |U_{k+1}|)$ time, then iterate over its edges and remove any edge that does not belong to G . This can be done easily in $\tilde{O}(|H_k| + |U_{k+1}|)$ time and $\tilde{O}(m)$ space by storing the adjacency lists of G in a BST so that each pair can be checked to be neighbors in $\tilde{O}(1)$ time. Then running [Proposition 2.1](#) on the resulting graph requires $O(\varepsilon^{-1}(|H_k| + |U_{k+1}|))$ time. Noting that $U_{k+1} \subseteq U_j$ by [Claim 4.4](#), the overall time of this step is $\tilde{O}(\varepsilon^{-1}(|H_k| + |U_j|))$.

Summing up all the mentioned bounds proves the bound of the claim. \square

Recall that the time-complexity of $\text{COMPUTELAYERS}(j)$ is amortized over $\frac{\varepsilon}{k} \cdot \frac{\mu_j+1}{p_j}$ updates. Using the upper bound of [Claim 4.13](#), we amortized cost of $\text{COMPUTELAYERS}(j)$ is thus at most:

$$O\left(\frac{(\varepsilon^{-1} + \beta k)|U_j| + \varepsilon^{-1}|H_k|}{\frac{\varepsilon}{k} \cdot \frac{\mu_j+1}{p_j}}\right) = \left(\frac{p_j|U_j|}{\mu_j+1} + \frac{p_j|H_k|}{\mu_j+1}\right) \text{poly}(\varepsilon^{-1} \beta k \log n). \quad (6)$$

[Claim 4.14](#) below can be used to bound the first term, and [Claim 4.15](#) can be used to bound the second term.

Claim 4.14. *For every $j \in [k+1]$, with probability $1 - 1/n^4$, $\frac{p_j|U_j|}{\mu_j+1} = \Delta^{\frac{1}{k+1}} \cdot \text{poly}(\varepsilon^{-1} \beta \log n)$.*

Claim 4.15. *For every $j \in [k+1]$, with probability $1 - 1/n^4$, $\frac{p_j|H_k|}{\mu_j+1} = O(\beta \log n)$.*

²Note that there is an edge case: If the number of remaining updates is not as large as $\frac{\varepsilon}{k} \cdot \frac{\mu_j+1}{p_j}$ then we cannot amortize the cost over the future updates. However, since this happens at most once for each j , and since $\text{COMPUTELAYERS}(j)$ clearly takes at most linear-time in the number of edges of the whole graph, we can amortize this cost over the whole sequence of updates which involves $\Omega(m)$ updates as assumed in [Theorem 4.1](#).

The proof of [Claim 4.14](#) is harder and is carried out in two sections. In [Section 4.5.1](#) we prove [Claim 4.14](#) for $j \geq 2$ using a sparsification guarantee on the size of U_j for $j \geq 2$. Then in [Section 4.5.2](#) we prove [Claim 4.14](#) for $j = 1$ using a lower bound on the size of μ_1 .

The proof of [Claim 4.15](#) is simple and we present it in [Theorem 4.1](#).

Before proving these claims, let us confirm that they do indeed imply the update-time bound of [Theorem 4.1](#) (emphasizing again that we are bounding the amortized update-time here which we turn into a worst-case bound in [Section 4.6](#)).

Proof of the update-time of Theorem 4.1. As discussed, all computations outside COMPUTELAYERS take $k \text{ poly}(\log n)$ worst-case time per update. On the other hand, replacing the bounds of [Claim 4.14](#) and [Claim 4.15](#) into [Equation \(6\)](#), we get that the amortized cost of COMPUTELAYERS(j) for every $j \in [k+1]$ is $\Delta^{\frac{1}{k+1}} \cdot \text{poly}(\varepsilon^{-1} \beta k \log n)$. Summing all of them up, this only multiplies this bound by a $(k+1)$ factor, which is still $\Delta^{\frac{1}{k+1}} \cdot \text{poly}(\varepsilon^{-1} \beta k \log n)$.

We show in [Section 4.6](#) how the algorithm can easily be de-amortized by spreading the cost of a call to COMPUTELAYERS over multiple edge-updates, obtaining the claimed update-time bound of [Theorem 4.1](#). \square

4.5.1 Proof of [Claim 4.14](#) for $j \geq 2$

For when $j \geq 2$, our main tool is the following sparsification lemma, which bounds the size of subgraph U_j . This sparsification property holds because of the special way we construct subgraph H_{j-1} in [Algorithm 3](#). Intuitively, we commit to H_{j-1} in [Algorithm 3](#) when many edges have arrived in the order of π and none of them are (H_{j-1}, β) -underfull. Since π is a random order of the edge-set of the graph, we can then conclude that w.h.p. there are not so many (H_{j-1}, β) -underfull edges left, which is precisely the size of U_j right after we recompute H_{j-1} .

Although the details and parameters are very different, we note that the proof of this sparsification property is inspired by a work of Bernstein [\[Ber20\]](#) in the random-order streaming model.

Lemma 4.16 (Sparsification Lemma). *For any $i \in [k]$, at any given time in the algorithm it holds with probability $1 - 1/n^5$ that*

$$|U_{i+1}| = O\left(\frac{\mu_i \beta^2 \log n}{p_i}\right).$$

Proof. Note that after H_i and U_{i+1} are computed in [Algorithm 2](#), it takes at most $\frac{\varepsilon}{k} \cdot \frac{\mu_i}{p_i}$ other updates to recompute them. During these updates, the size of U_{i+1} increases by at most $\frac{\varepsilon}{k} \cdot \frac{\mu_i}{p_i} \leq \frac{\mu_i}{p_i}$ since each edge update either adds at most one edge to U_{i+1} . As a result, it suffices to prove that w.h.p. $|U_{i+1}| = O\left(\frac{\mu_i \beta^2 \log n}{p_i}\right)$ right after a call to [Algorithm 2](#).

The crux of the proof will be about analyzing the behavior of ADDAYER ([Algorithm 3](#)) which is called in [Algorithm 2](#) to construct subgraph H_i , which in turn, is used to define U_{i+1} .

As a thought experiment and only for the sake of the analysis, consider a modified version of [Algorithm 3](#), which we call MODIFIEDADDAYER, with two changes: (i) instead of Γ which will be $U_i \cap G_i$ when ADDAYER is called, we iterate over all edges of U_i in a *random* order; additionally (ii) MODIFIEDADDAYER takes a parameter τ as the input and returns H_i when $\eta \geq \tau$ as opposed

to the condition in **Line 9** of ADDLAYER. We will first analyze MODIFIEDADDLAYER and then show how it relates to the actual algorithm ADDLAYER.

Let us condition on the subgraph H_i after MODIFIEDADDLAYER processes some t edges of U_i and let y be the number of unprocessed edges of U_i that are (H_i, β) -underfull. The probability that the next edge that arrives is (H_i, β) -underfull is at least $y/|U_i|$. Moreover, assuming that this edge is not (H_i, β) -underfull, subgraph H_i does not change and so there are still at least y other (H_i, β) -underfull unprocessed edges. As a result, the probability that the algorithm processes at least $10(|U_i|/y) \log n$ more edges and none of them are (H_i, β) -underfull, is at most

$$(1 - y/|U_i|)^{10(|U_i|/y) \log n} \leq e^{-10 \log n} \leq n^{-10}.$$

Equivalently, by a union bound over at most n^2 choices of t , we get that if MODIFIEDADDLAYER successfully returns subgraph H_i , which recall that happens when it processes τ edges of U_i and does not encounter any (H_i, β) -underfull edges, then with probability $1 - n^{-8}$, the total number of (H_i, β) -underfull unprocessed edges in U_i is at most $10(|U_i|/\tau) \log n$. By another union bound over at most n^2 choices of τ , this holds for *every* input $\tau \in [n^2]$ with probability $1 - n^{-6}$.

Now let us go back to the actual algorithm ADDLAYER. We prove the claim even if we condition on subgraph G_{i-1} . That is, suppose that all the edges e with $\pi_e \leq p_{i-1}$ are revealed. Note that conditioned on this event, the rank of every edge in $G \setminus G_{i-1}$ is independent and uniformly picked from $(p_{i-1}, 1]$. Now, recall that in ADDLAYER we process the edges of $U_i \cap G_i$ in the increasing order of π . Since an edge belongs to G_i iff $\pi_e \leq p_i$ and $p_i < p_{i+1} < \dots < p_{k+1}$ and since U_i does not include any edge of G_{i-1} by definition, this is equivalent to iterating over the edges of U_i until the next edge e in the sequence has $\pi_e > p_i$, i.e., does not belong to G_i anymore. But if we reach this point in ADDLAYER, then it means that we have not already returned H_i , contradicting **Corollary 4.8** that the algorithm always terminates. This implies that the extra condition on the next edge not belonging to G_i is not, in fact, needed. This implies, in turn, that algorithm MODIFIEDADDLAYER is exactly equivalent to ADDLAYER where the parameter τ is simply set to $\lfloor |\Gamma|/(4\mu_i\beta^2 + 1) \rfloor = \lfloor |U_i \cap G_i|/(4\mu_i\beta^2 + 1) \rfloor$. From the discussion of the previous paragraph, therefore, we can infer that when ADDLAYER terminates, the number of unprocessed edges in U_i that are (H_i, β) -underfull (which also includes all edges in U_{i+1}) is with probability $1 - n^{-6}$ at most

$$\frac{10|U_i| \log n}{\lfloor |U_i \cap G_i|/(4\mu_i\beta^2 + 1) \rfloor}. \quad (7)$$

We finish the proof by considering the two cases (i) $p_i|U_i| \geq \mu_i\beta^2 \log n$ and (ii) $p_i|U_i| < \mu_i\beta^2 \log n$ separately. (Case (ii) happens to be trivial.)

Consider case (i) first. Since $U_i \subseteq G \setminus G_{i-1}$ and since each edge of $G \setminus G_{i-1}$ belongs to G_i independently with probability $\frac{p_i - p_{i-1}}{1 - p_{i-1}} \geq p_i/2$ (the inequality holds by **Claim 4.3** part (v)) conditioned on G_{i-1} , we get $\mathbf{E}[|U_i \cap G_i| \mid G_{i-1}] \geq \Omega(p_i|U_i|)$. Moreover, $p_i|U_i|$ is large enough in case (i) to apply the Chernoff bound and get $|U_i \cap G_i| = \Omega(p_i|U_i|)$ with probability, say, $1 - 1/n^{-6}$. Combined with **Equation (7)** we can bound the size of $|U_{i+1}|$, with probability $\geq 1 - n^{-5}$ by

$$|U_{i+1}| \stackrel{(7)}{\leq} \frac{10|U_i| \log n}{\lfloor |U_i \cap G_i|/(4\mu_i\beta^2 + 1) \rfloor} = O\left(\frac{|U_i| \log n}{p_i|U_i|/(\mu_i\beta^2)}\right) = O\left(\frac{\mu_i\beta^2 \log n}{p_i}\right).$$

(We note that we used the assumption of (i) one more time in the first equality above to get that the denominator does not become zero when taking the floor.)

For case (ii), note that $U_{i+1} \subseteq U_i$ from **Claim 4.4** and thus we simply get $|U_{i+1}| \leq |U_i| \leq \frac{\mu_i\beta^2 \log n}{p_i}$ where the last inequality uses the assumption of (ii). The proof is thus complete. \square

Armed with the sparsification lemma, we can now prove [Claim 4.14](#) for $j \geq 2$.

Proof of Claim 4.14 for $j \geq 2$. From [Lemma 4.16](#), we get that with probability $1 - 1/n^5$,

$$\frac{p_j|U_j|}{\mu_j + 1} = \frac{p_j}{\mu_j + 1} \cdot O\left(\frac{\mu_{j-1}\beta^2 \log n}{p_{j-1}}\right) = O\left((\Delta^{\frac{1}{k+1}}/\varepsilon) \cdot \frac{\mu_{j-1}\beta^2 \log n}{\mu_j + 1}\right), \quad (8)$$

where the last bound follows from bound $p_j/p_{j-1} = O(\Delta^{\frac{1}{k+1}}/\varepsilon)$ for all $j \geq 2$ of [Claim 4.3](#) part (iii).

Now note that μ_{j-1} is set to be the size of a maximal matching of G_{j-1} in COMPUTELAYERS and μ_j is the size of a maximal matching of G_j for the same reason. We would like to say that this means $\mu_{j-1} = O(\mu_j)$ since $G_{j-1} \subseteq G_j$. The only remaining challenge, however, is that we only set μ_{j-1} to be $\tilde{\mu}(G_{j-1})$ in COMPUTELAYERS and do not update μ_{j-1} until COMPUTELAYERS(i) is called again for some $i \leq j-1$. As a result, the edge updates since μ_{j-1} was last computed may cause $\tilde{\mu}(G_i)$ to get much smaller than μ_{j-1} . However, given that μ_{j-1} is recomputed after at most $\frac{\varepsilon}{k} \cdot \frac{\mu_{j-1}+1}{p_{j-1}}$ updates and only p_{j-1} fraction of these updates belong to G_{j-1} in expectation, the expected number of removed edges from G_{i-1} is at most $O(\frac{\varepsilon}{k}\mu_{j-1}) = O(\varepsilon\mu_{j-1})$ and so $\tilde{\mu}(G_{i-1})$ is in expectation still at least $\Omega(\mu_{j-1})$.

To go from expectation to high probability, let F_{j-1} be the set of edges added/removed from G_{j-1} during the last c_j updates (i.e., since the last time μ_{j-1} was computed). We showed in [Claim 4.12](#) that $|F_{j-1}| = O(\frac{\varepsilon}{k}\mu_{j-1} + \log n)$ w.h.p. As such, under this high probability event, we have $\mu_j = \tilde{\mu}(G_j) = \Omega(\tilde{\mu}(G_{j-1})) = \Omega(\mu_{j-1} - \frac{\varepsilon}{k}\mu_{j-1} - \log n) = \Omega(\mu_{j-1} - \log n)$. This means that, w.h.p., $\mu_{j-1}/(\mu_j + 1) = O(\log n)$. Plugging this to [Equation \(8\)](#) we get $\frac{p_j|U_j|}{\mu_j + 1} \ll O(\Delta^{\frac{1}{k+1}}\varepsilon^{-1}\beta^2 \log^3 n)$. \square

4.5.2 Proof of [Claim 4.14](#) for $j = 1$

For the case where $j = 1$, by definition $U_j = U_1 = G$. Therefore, U_1 includes all the edges of the graph. To bound the update-time in this case, we show that μ_1 is sufficiently large.

While it is well-known that any m -edge graph of maximum degree Δ has a matching of size $\Omega(m/\Delta)$, we prove in [Claim 4.17](#) below the somewhat surprising fact that essentially the same lower bound of $\Omega(m/\Delta)$ holds for the size of the maximum matching in a random edge-subgraph provided that the edge-sampling probability satisfies a rather mild constraint.

Claim 4.17. *Let $G = (V, E)$ be an arbitrary n -vertex graph, let Δ be an upper bound on G 's maximum degree, and let $G_p = (V, E_p)$ be a random subgraph of G including each edge independently with some probability p . If $p \geq \max\{15\frac{\ln n}{\Delta}, 32\frac{\ln n}{|E|}\}$, then $\Pr\left[\mu(G_p) \geq \frac{|E|}{8\Delta}\right] \geq 1 - 2/n^4$.*

Proof. Since $|E_p|$ is a sum of $|E|$ independent p -Bernoulli random variables, we have $\mathbf{E}|E_p| = p|E|$ and by applying the Chernoff bound we get

$$\Pr\left[|E_p| < \frac{p|E|}{2}\right] \leq \exp\left(-\frac{0.5^2 \mathbf{E}|E_p|}{2}\right) = \exp\left(-\frac{p|E|}{8}\right) \stackrel{p \geq \frac{32\ln n}{|E|}}{\leq} \exp(-4\ln n) = n^{-4}. \quad (9)$$

Next, note that for every vertex v , $\deg_{G_p}(v)$ is a sum of $\deg_G(v)$ independent p -Bernoulli random variables. This means $\mathbf{E}[\deg_{G_p}(v)] = p\deg_G(v) \leq p\Delta$. Applying Chernoff bound, we therefore get

$$\Pr[\deg_{G_p}(v) \geq 2p\Delta] \leq \exp\left(-\frac{p\Delta}{3}\right) \stackrel{p \geq 15\ln n/\Delta}{\leq} \exp\left(-\frac{15\ln n}{3}\right) = n^{-5}. \quad (10)$$

By a union bound, the maximum degree Δ_p of G_p is at most $2p\Delta$ with probability $1 - n^{-4}$.

Now take an arbitrary $2\Delta_p$ edge coloring of G_p and pick the color class with the largest number of colors. This is a matching of size at least $\frac{|E_p|}{2\Delta_p}$. Hence,

$$\mu(G_p) \geq \frac{|E_p|}{2\Delta_p} \stackrel{(9)}{\geq} \frac{p|E|/2}{2\Delta_p} \stackrel{(10)}{\geq} \frac{p|E|/2}{2(2p\Delta)} \geq \frac{|E|}{8\Delta},$$

with probability at least $1 - 2n^{-4}$. \square

Claim 4.17 is all we need to prove **Claim 4.14** for $j = 1$.

Proof of Claim 4.14 for $j = 1$. Recall from the statement of **Claim 4.14** that we need to prove $\frac{p_1|U_1|}{\mu_1+1} = O(\Delta^{\frac{1}{k+1}}\beta^2 \log^2 n)$. Since $U_1 = G$ at all times, we would like to apply **Claim 4.17** and obtain that μ_1 is w.h.p. at least $\Omega(|G|/\Delta) = \Omega(|U_1|/\Delta)$. From this, we would get that $\frac{p_1|U_1|}{\mu_1+1} = O(p_1\Delta)$. Given that $p_1 \leq 15\Delta^{\frac{1}{k+1}-1} \log n$, the RHS is $O(\Delta^{\frac{1}{k+1}} \log n)$ which is the desired bound.

To apply **Claim 4.17** and complete the proof, we only need to show that $p_1 \geq 15\frac{\ln n}{\Delta}$ and $p_1 \geq 32\frac{\ln n}{|G|}$. The first inequality is proved in **Claim 4.3** part (ii). If the second condition does not hold, i.e., if $p_1 < 32\frac{\ln n}{|G|}$, then we can prove the claim trivially. To see this, note that $\frac{p_1|U_1|}{\mu_1+1} \leq p_1|U_1| = p_1|G|$ and the latter is at most $O(\log n)$ if $p_1 < 32\frac{\ln n}{|G|}$. Hence, either **Claim 4.14** follows trivially or we can apply **Claim 4.17** and prove it as discussed above. \square

4.5.3 Proof of **Claim 4.15**

In this section we prove **Claim 4.15** that, w.h.p., $\frac{p_j|H_k|}{\mu_j+1} = O(\beta \log n)$.

We start with a simple observation to bound the size of H_k .

Observation 4.18. $|H_k| \leq 2\mu(G)\beta$.

Proof. Observe that G has a vertex cover W with size at most $2\mu(G)$ (pick the two endpoints of a maximum matching of G). Moreover, since $H_k \subseteq G$ by **Claim 4.4** part (v), W is also a vertex cover of H_k . Combined with **Observation 4.6** that bounds the maximum degree of H_k by β , we get that H_k has at most $|W|\beta = 2\mu(G)\beta$ edges. \square

Observe that if $p_j\mu(G) \leq 10\log n$ then we readily have the bound of **Claim 4.15** since, by **Observation 4.18**, $\frac{p_j|H_k|}{\mu_j+1} \leq \frac{p_j\mu(G)\beta}{\mu_j+1} \leq p_j\mu(G)\beta = O(\beta \log n)$. So let us assume $p_j\mu(G) \geq 10\log n$.

Now fix a maximum matching of G and recall that each edge appears in G_j independently with probability p_i . As such, $\mathbf{E}[\mu(G_j)] \geq p_j\mu(G)$. With our earlier assumption of $p_j\mu(G) \geq 10\log n$, we can thus apply the Chernoff bound to get that with probability, say, $1 - 1/n^4$, $\mu(G_j)$ is at least $\Omega(p_j\mu(G))$. Combined with **Observation 4.18**, we thus get

$$\frac{p_j|H_k|}{\mu_j+1} \leq \frac{p_j \cdot \mu(G)\beta}{\mu_j+1} \leq \frac{p_j \cdot \mu(G)\beta}{\mu_j} = \frac{p_j \cdot \mu(G)\beta}{\Omega(p_j\mu(G))} = O(\beta).$$

This completes the proof of **Claim 4.15**.

4.6 Getting a Worst-Case Update-time Bound

The algorithm that we presented in [Section 4.2](#) was shown in [Section 4.5](#) to have the same update-time as claimed in [Theorem 4.1](#). However, we analyzed the *amortized* update-time in [Section 4.5](#) instead of the *worst-case* update-time. In this section, we show how with a simple trick of spreading the computation over multiple updates, we can get the same update-time but in the worst-case. We note that this idea is standard and has been used before in [\[GP13\]](#) and [\[BFH19\]](#). As such, we only give a high level discussion of how it works.

Observe that the only place in the analysis of update-time that we used amortization was in bounding the update-time caused by the calls to $\text{COMPUTELAYERS}(j)$ for various j . Indeed, with the way we defined the algorithm in [Section 4.2](#), this amortization is necessary since when we call $\text{COMPUTELAYERS}(j)$ the time-complexity is larger than the final update-time and this must be amortized. The trick to get a worst-case bound is to spread this computation over the updates. That is, suppose that $\text{COMPUTELAYERS}(j)$ takes T operations. Instead of performing all these T operations over one single edge update, we do it over multiple edge updates.

More formally, recall that in our algorithm, when we call $\text{COMPUTELAYERS}(j)$, we set c_j to be zero. Then upon each update we increase c_j by one and call $\text{COMPUTELAYERS}(j)$ again when c_j exceeds $\frac{\varepsilon}{k} \cdot \frac{\mu_j+1}{p_j}$. Now instead, when c_j exceeds half this threshold, we call $\text{COMPUTELAYERS}(j)$ but spread its computation over the next $0.5 \frac{\varepsilon}{k} \cdot \frac{\mu_j+1}{p_j}$ updates. Only when this computation is finished, we update the data structures. It is easy to see that the approximation ratio does not hurt since the total “wait time” until $\text{COMPUTELAYERS}(j)$ is called again remains the same. For the update-time, one can adapt essentially the same analysis of the amortized update-time of [Section 4.5](#) to show that this modified algorithm now has the same update-time but in the worst-case.

See [\[GP13\]](#) and [\[BFH19\]](#) for more discussions on this deamortization technique.

4.7 Bounding Maximum Degree by $O(\sqrt{m})$

Up to this point, we have given an algorithm satisfying the approximation guarantee of [Theorem 4.1](#) in $\Delta^{\frac{1}{k+1}} \cdot \text{poly}(\varepsilon^{-1} \beta k \log n) = \tilde{O}(\Delta^{\frac{1}{k+1}})$ update-time, whereas observe that we claimed a bound of $\min\{\Delta^{\frac{1}{k+1}}, m^{\frac{1}{2(k+1)}}\} \cdot \text{poly}(\varepsilon^{-1} \beta k \log n)$ in [Theorem 4.1](#). In this section, we show how this is possible.

We prove the following lemma that can be applied as a black-box to the algorithm we have, yielding the guarantee of [Theorem 4.1](#).

Lemma 4.19. *Consider a fully dynamic graph G and let Δ and m be fixed upper bounds on the maximum degree and the number of edges of G . Suppose that there is an algorithm \mathcal{A} that maintains an α -approximate maximum matching of G in $T(\Delta, n)$ update-time for some function $T(\Delta, n)$. Then there is an algorithm \mathcal{A}' that maintains a $(1 - \varepsilon)\alpha$ -approximate maximum matching of G in $O(\min\{T(\Delta, n), T(\sqrt{m}/\varepsilon, n)\} \log n)$ update-time. If the update-time of \mathcal{A} is worst-case, then so is that of \mathcal{A}' .*

Proof. Consider a process where each vertex v in G marks $\Delta' = O(\sqrt{m}/\varepsilon)$ of its edges arbitrarily and let \tilde{G} be the subgraph of G including each edge that is marked by *both* of its endpoints. Note that \tilde{G} clearly has maximum degree Δ' . We show that this subgraph can be maintained in a way that every edge update to G leads to at most three edge updates to \tilde{G} . Additionally, we show that \tilde{G} will always include a $(1 - O(\varepsilon))$ -approximate maximum matching of G .

Let us first show how \tilde{G} can be maintained in $O(\log n)$ worst-case time, by simply maintaining the marked and not-marked edges of each vertex in a balanced BST. Upon insertion of an edge e we check how many edges each of its endpoints are marked; each one of them that has marked less than Δ' edges adds e to the set of its marked edges and if both add it we insert e to \tilde{G} . Upon deletion of an edge e , if it belongs to \tilde{G} we remove it, we also remove it from the marked edges of its endpoints. If the number of marked edges of any endpoint of e goes below the threshold Δ' , it marks a new edge and adds it to \tilde{G} if it should. Overall, each edge update to G can be handled in $O(\log n)$ time and leads to at most 3 edge updates to \tilde{G} . Additionally, the construction of \tilde{G} is completely deterministic and so the sequence of updates to \tilde{G} gets fixed once those of G are fixed.

Now we prove that at any time $\mu(\tilde{G}) \geq (1 - O(\varepsilon))\mu(G)$. To show this, we note that the marking algorithm above was first introduced by Solomon [Sol18]. He showed that by setting $\Delta' = O(\alpha/\varepsilon)$ where α is the *arboricity* of the graph, \tilde{G} will include a $(1 - \varepsilon)$ -approximate maximum matching of G . This is all we need since it is a well-known fact that every m -edge graph has arboricity $O(\sqrt{m})$.

To conclude the proof, note that we can simply maintain \tilde{G} and feed it to algorithm \mathcal{A} . Since the maximum degree of \tilde{G} is always $O(\sqrt{m}/\varepsilon)$ and its matching is nearly as large as G , we get the claimed bound. \square

5 Conclusion & Open Problems

We introduced the hierarchical edge-degree subgraph (HEDCS). Using the HEDCS, we gave a unified framework that leads to several new update-time/approximation trade-offs for the fully dynamic matching problem, while also recovering previous trade-offs.

While we provided both a factor revealing LP (Section 3.3) and another analytical method (Appendix A) for analyzing the approximation ratio achieved via HEDCS, it remains an extremely interesting problem to analyze its precise approximation factor. Specifically:

- While the approximation ratio of HEDCS can still be tangibly above $1/2$ for say $k = 4, k = 5$, etc., we did not specify any lower bounds on this approximation ratio in Theorem 1.1 since our factor revealing LP of Section 3.3 becomes too inefficient to run for $k > 3$.
- In our bounds of Theorem 1.1 there are gaps between bipartite graphs and general graphs. We conjecture that this gap should not exist and the approximation ratio achieved via β -HEDCS $_k$, for any constant k , should converge to the same value for both bipartite and general graphs (by letting parameter β to be a large enough constant).
- It would be interesting to analyze how fast the approximation ratio of β -HEDCS $_k$ converges to $1/2$ by increasing k . While our Lemma A.1 shows that the approximation ratio of any β -HEDCS $_k$ is at least $\frac{1}{2} + \frac{1}{2^{2^{\mathcal{O}(k)}}}$ (for large enough β), we do not believe this double-exponential dependence on k is the right answer. A bound of, say, $\frac{1}{2} + \frac{1}{k^{\mathcal{O}(1)}}$ would be very interesting.

Next, we note that while the oblivious adversary assumption is well-received in the literature and holds in many natural applications of dynamic matching, it would be interesting to obtain the new trade-offs that we give also against adaptive adversaries (see [Waj20, BK21] for discussions about adaptive adversaries in the context of dynamic matching). One way to achieve this would be to give a deterministic dynamic algorithm for maintaining an HEDCS.

More broadly, the following intriguing questions about dynamic matching still remain open:

Open Problem 1. Does there exist a fully dynamic algorithm maintaining a $(\frac{1}{2} + \Omega(1))$ -approximate maximum matching in $n^{o(1)}$ update-time? In $\text{poly}(\log n)$ update-time?

Open Problem 2. Does there exist a fully dynamic algorithm maintaining a $(\frac{2}{3} + \Omega(1))$ -approximate maximum matching in $o(n)$ update-time?

Acknowledgements

We thank the anonymous SODA'22 reviewers for their thoughtful comments.

References

- [AB19] Sepehr Assadi and Aaron Bernstein. Towards a Unified Theory of Sparsification for Matching Problems. In *2nd Symposium on Simplicity in Algorithms, SOSA 2019, January 8-9, 2019, San Diego, CA, USA*, volume 69 of *OASIcs*, pages 11:1–11:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- [ABB⁺19] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. 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*, pages 1616–1635. SIAM, 2019.
- [ACC⁺18] Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. 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*, pages 7:1–7:16, 2018.
- [AW14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014.
- [BDH⁺19] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully Dynamic Maximal Independent Set with Polylogarithmic Update Time. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 382–405. IEEE Computer Society, 2019.
- [BDL21] Aaron Bernstein, Aditi Dudeja, and Zachary Langley. A Framework for Dynamic Matching in Weighted Graphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021, to appear*, 2021.
- [Beh21] Soheil Behnezhad. Improved analysis of edcs via gallai-edmonds decomposition. *arXiv preprint arXiv:2110.05746*, 2021.
- [Ber20] Aaron Bernstein. Improved Bounds for Matching in Random-Order Streams. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPICS*, pages 12:1–12:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- [BFH19] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A Deamortization Approach for Dynamic Spanner and Dynamic Maximal Matching. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1899–1918, 2019.
- [BGS11] Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 383–392. IEEE Computer Society, 2011.

- [BGS18] Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully Dynamic Maximal Matching in $O(\log n)$ Update Time (Corrected Version). *SIAM J. Comput.*, 47(3):617–650, 2018.
- [BHI18] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic Fully Dynamic Data Structures for Vertex Cover and Matching. *SIAM J. Comput.*, 47(3):859–887, 2018.
- [BHN16] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 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*, pages 398–411. ACM, 2016.
- [BHN17] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 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, 2017*, pages 470–489. SIAM, 2017.
- [BK21] Sayan Bhattacharya and Peter Kiss. 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)*, pages 27:1–27:14, 2021.
- [BLM20] Soheil Behnezhad, Jakub Lacki, and Vahab S. Mirrokni. Fully Dynamic Matching: Beating 2-Approximation in Δ^ε Update Time. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2492–2508. SIAM, 2020.
- [BS15a] Aaron Bernstein and Cliff Stein. 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*, volume 9134 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2015.
- [BS15b] Aaron Bernstein and Clifford Stein. Fully Dynamic Matching in Bipartite Graphs. *CoRR*, abs/1506.07076, 2015.
- [BS16] Aaron Bernstein and Cliff Stein. Faster Fully Dynamic Matchings with Small Approximation Ratios. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 692–711. SIAM, 2016.
- [CS18] Moses Charikar and Shay Solomon. Fully Dynamic Almost-Maximal Matching: Breaking the Polynomial Worst-Case Time Barrier. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 33:1–33:14, 2018.
- [Dah16] Søren Dahlgaard. On the hardness of partially dynamic graph problems and connections to diameter. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 48:1–48:14, 2016.
- [GKPS06] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *J. ACM*, 53(3):324–360, 2006.

- [GP13] Manoj Gupta and Richard Peng. Fully Dynamic $(1 + \varepsilon)$ -Approximate Matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 548–557. IEEE Computer Society, 2013.
- [HKNS15] 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 Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30, 2015.
- [MV80] Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V||E|})$ Algorithm for Finding Maximum Matching in General Graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27. IEEE Computer Society, 1980.
- [NS13] Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 745–754, 2013.
- [RSW21] Mohammad Roghani, Amin Saberi, and David Wajc. Beating the folklore algorithm for dynamic matching. *CoRR*, abs/2106.10321, 2021.
- [San07] Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 118–126. SIAM, 2007.
- [Sol16] Shay Solomon. 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*, pages 325–334. IEEE Computer Society, 2016.
- [Sol18] Shay Solomon. Local algorithms for bounded degree sparsifiers in sparse graphs. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPICS*, pages 52:1–52:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
- [Vaz12] Vijay V. Vazirani. An Improved Definition of Blossoms and a Simpler Proof of the MV Matching Algorithm. *CoRR*, abs/1210.4594, 2012.
- [vdBNS19] Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 456–480, 2019.
- [Waj20] David Wajc. 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*, pages 194–207. ACM, 2020.

A Approximation Ratio of HEDCS: An Analytical Lower Bound

In this section, we give a lower bound on the approximation ratio $\alpha(k, \beta, \beta^-)$ for large k . This analysis is particularly useful when the LP approach described in [Section 3.3](#) (which produces better lower bounds) becomes too inefficient to run in practice. We prove the following:

Lemma A.1. *Fix any integers $k \geq 1$ and $\beta > \beta^- \geq 1$ where $\beta^- = (1 - \delta)\beta$ for some $0 \leq \delta \leq 0.2$. Define $h(x) := 1/2^{2^x}$. If $h(k) - 4\delta \geq 0$ then*

$$\alpha(k, \beta, \beta^-) \geq \frac{1}{2} + \frac{h(k)}{6} - \frac{2}{3}\delta.$$

Note that by picking β large enough and β^- close enough to β , we can make δ desirably small (even dependent on k), satisfying $h(k) - 4\delta \geq h(k)/2 > 0$. Hence, [Lemma A.1](#) together with the approximation guarantee of HEDCS based on function $\alpha(\cdot)$ discussed in [Section 3](#) implies that setting $k = \frac{1}{\varepsilon} - 1$ and setting $\beta = O_\varepsilon(1)$ large enough, results in a β -HEDCS $_k$ which includes a strictly better than half approximate matching of ratio $\frac{1}{2} + \Omega_\varepsilon(1)$ and can be maintained in update-time $\tilde{O}(\min\{\Delta^\varepsilon, m^{\varepsilon/2}\})$, as claimed in [Table 1](#).

We prove the following auxiliary claim.

Claim A.2. *Let $h(x) := 1/2^{2^x}$. Let $H = (P, Q, E)$ be any bipartite β -HEDCS $_k$ with at least $(1 - \delta)\beta|P|/2$ edges, where $0 \leq \delta \leq 1 - 3 \sum_{i=1}^k \sqrt{h(i)}$. Then*

$$|Q| \geq (1 + h(k) - 4\delta) \frac{|P|}{2}.$$

Let us first see how [Claim A.2](#) proves [Lemma A.1](#).

Proof of Lemma A.1. First, it can be confirmed that $1 - 3 \sum_{i=1}^k \sqrt{h(i)} > 0.2$ for any $k \geq 1$. As a result, the condition $0 \leq \delta \leq 0.2$ of [Lemma A.1](#) always satisfies the condition $0 \leq \delta \leq 1 - 3 \sum_{i=1}^k \sqrt{h(i)}$ of [Claim A.2](#).

Next, note from [Definition 3.4](#) that since [Claim A.2](#) holds for any bipartite β -HEDCS $_k$, we have

$$f(k, \beta, \beta^-) \geq \frac{1}{2} \cdot (1 + h(k) - 4\delta).$$

On the other hand, by [Definition 3.5](#) we have $\alpha(k, \beta, \beta^-) = \frac{2f(k, \beta, \beta^-)}{2f(k, \beta, \beta^-) + 1}$, which means

$$\alpha(k, \beta, \beta^-) \geq \frac{1 + h(k) - 4\delta}{2 + h(k) - 4\delta} \geq \frac{1}{2} + \frac{h(k) - 4\delta}{6} = \frac{1}{2} + \frac{h(k)}{6} - \frac{2}{3}\delta.$$

The second inequality above comes from the fact that $\frac{1+x}{2+x} \geq \frac{1}{2} + \frac{x}{6}$ for any $0 \leq x \leq 1$. \square

So it remains to prove [Claim A.2](#).

Proof. The function h in the statement is specifically defined in a way that for any integer $k \geq 2$, it satisfies the following condition which will be useful later in the proof

$$h(k-1) - 12\sqrt{h(k)} \geq h(k). \quad (11)$$

Let us, for brevity, define $d_P := m/|P| = (1 - \delta)\beta/2$ to be the average degree of the P side of H . Let us also assume that H_1, \dots, H_k is a hierarchical decomposition for β -HEDCS $_k$ which must exist by definition of HEDCS.

We now prove the claim by induction on k .

Base case $k = 1$: Since for $k = 1$ a β -HEDCS₁ is by definition equivalent to a β -EDCS, from the known bounds for EDCS we have $|Q| \geq \frac{d_P}{\beta - d_P} |P|$ (see e.g. [Beh21, Lemma 2.2] or [AB19]). Thus:

$$|Q| \geq \frac{d_P}{\beta - d_P} |P| = \frac{(1 - \delta)\beta/2}{\beta - (1 - \delta)\beta/2} |P| = \frac{1 - \delta}{1 + \delta} |P| \geq (1 - 2\delta) |P| = (2 - 4\delta) \frac{|P|}{2} > (1 + \frac{1}{2^{2^2}} - 4\delta) |P|.$$

Induction step: Now let us assume that the claim holds for $k - 1$ and prove it for k . Let us for brevity define $\xi := \sqrt{h(k)}$ and assume for the sake of contradiction that

$$|Q| < (1 + h(k) - 4\delta) \frac{|P|}{2} = (1 + \xi^2 - 4\delta) \frac{|P|}{2}. \quad (12)$$

Define $Q_T := \{w \in Q \mid \deg_H(w) \leq (1 - \xi)\beta\}$. Since the maximum degree of any HEDCS is bounded by β as proved in Observation 3.1, we have

$$(|Q| - |Q_T|) \cdot \beta + |Q_T| \cdot (1 - \xi)\beta \geq m \geq (1 - \delta)\beta |P|/2.$$

The LHS can be simplified to $\beta|Q| - \xi\beta|Q_T|$. Then by canceling the β terms on both sides, we get

$$|Q| - \xi|Q_T| \geq (1 - \delta)|P|/2.$$

This, in turn, implies that

$$|Q_T| \leq \frac{|Q| - (1 - \delta)\frac{|P|}{2}}{\xi} \stackrel{(12)}{\leq} \frac{(1 + \xi^2 - 4\delta)\frac{|P|}{2} - (1 - \delta)\frac{|P|}{2}}{\xi} = \frac{\xi^2 - 3\delta}{\xi} \cdot \frac{|P|}{2} \stackrel{(\delta \geq 0)}{\leq} \xi \cdot \frac{|P|}{2}. \quad (13)$$

Now let P_k be the vertices in P that have at least one edge in $H_k \setminus H_{k-1}$ with the other endpoint in $Q \setminus Q_T$. Observe that by definition of Q_T , for any $v \in Q \setminus Q_T$ we have $\deg_H(v) > (1 - \xi)\beta$. As such, for property (i) of HEDCS to hold for H , every vertex in P_k must have degree at most $\xi\beta$. (As otherwise, its edge in $H_k \setminus H_{k-1}$ that goes to $Q \setminus Q_T$ violates property (i) of HEDCS.)

We remove all the *vertices* in Q_T (along with their edges) from graph H . We also remove all the *edges* of P_k from H , leaving their vertices as singletons in the graph. Observe that this removes all the edges of $H_k \setminus H_{k-1}$ from the graph since any such edge must be either connected to P_k or to Q_T , and so we end up with a β -HEDCS _{$k-1$} and can apply the induction hypothesis. Before applying it, though, let us upper bound the total number of removed edges. Since as discussed every vertex in H has degree at most β , and every vertex in P_k has degree at most $\xi\beta$,

$$\# \text{ of removed edges} \leq |Q_T| \cdot \beta + |P_k| \cdot \xi\beta \stackrel{(13)}{\leq} \xi \frac{|P|}{2} \cdot \beta + |P_k| \cdot \xi\beta \leq \frac{3}{2} \xi |P| \beta.$$

Hence, the resulting β -HEDCS _{$k-1$} still has $|P|$ vertices in one part, and the number of its edges is

$$(1 - \delta)\beta |P|/2 - \frac{3}{2}\xi |P| \beta = (1 - \delta - 3\xi)\beta |P|/2.$$

We have $\delta \leq 1 - 3 \sum_{i=1}^k \sqrt{h(i)} = 1 - 3 \sum_{i=1}^{k-1} \sqrt{h(i)} - 3\xi$ from the statement of the claim. This means that, first, $1 - \delta - 3\xi \geq 3 \sum_{i=1}^{k-1} \sqrt{h(i)} \geq 0$, and second, $\delta + 3\xi \leq 1 - 3 \sum_{i=1}^{k-1} \sqrt{h(i)}$. Hence, the number of edges of the resulting β -HEDCS _{$k-1$} satisfies the constraint of the lemma (i.e., the induction hypothesis) with $\delta' = \delta + 3\xi$ and so we get

$$|Q| \geq |Q \setminus Q_T| \geq (1 + h(k-1) - 4(\delta + 3\xi)) \frac{|P|}{2}$$

$$\begin{aligned}
&= (1 + h(k - 1) - 12\xi - 4\delta) \frac{|P|}{2} \\
&= (1 + h(k - 1) - 12\sqrt{h(k)} - 4\delta) \frac{|P|}{2} \\
&\geq (1 + h(k) - 4\delta) \frac{|P|}{2}. \tag{By (11).}
\end{aligned}$$

This is exactly the needed inequality for k and so the proof is complete. \square

B Proof of Proposition 3.6

Proposition 3.6 (restated). *Let H and U be subgraphs of a graph G , let $\beta \geq 2$ be any integer, and suppose that H is a β -HEDCS $_k$ of $G \setminus U$. Then:*

- If G is bipartite, $\mu(H \cup U) \geq \alpha(k, \beta, \beta - 1)\mu(G)$.
- If G is not necessarily bipartite, and $\beta \geq c(\beta'k)^2 \log(\beta'k)$ for some integer β' and a sufficiently large constant $c \geq 1$, then $\mu(H \cup U) \geq \alpha(k, \beta' + 2k - 1, \beta')\mu(G)$.

As discussed, the proof of is obtained by adapting the technique of Assadi and Bernstein [AB19] for the analysis of EDCS. Particularly, the analysis for the bipartite case is completely due to [AB19], and the analysis of the general case is also based on their approach, with an additional idea.

It would be convenient to consider a slightly more definition of HEDCS.

Definition B.1. *Let $\beta > \beta^- \geq 1$ and $k \geq 1$ be integers. We say graph H is a (β, β^-) -HEDCS $_k$ of G iff there is a hierarchical decomposition $\emptyset = H_0 \subseteq H_1 \subseteq H_2 \subseteq \dots \subseteq H_k = H$ satisfying:*

- (i) *For every $i \in [k]$ and any edge $e \in H_i \setminus H_{i-1}$, $\deg_{H_i}(e) \leq \beta$.*
- (ii) *For any edge $e \in G \setminus H$, $\deg_H(e) \geq \beta^-$.*

Note that only property (ii) has changed from the original [Definition 1.2](#) of HEDCS. In particular, a β -HEDCS $_k$ is simply a $(\beta, \beta - 1)$ -HEDCS $_k$.

We start with the following lemma that directly follows from the approach of [AB19]:

Lemma B.2. *Let H and U be subgraphs of a bipartite graph G and suppose H is a (β, β^-) -HEDCS $_k$ of $G \setminus U$. Then $\mu(H \cup U) \geq \alpha(k, \beta, \beta^-)\mu(G)$.*

Proof sketch. The proof is essentially identical to the proof of [AB19, Lemma 3.1] and we follow the same terminology. Let L and R be the vertex parts of the base graph G . Consider an extended Hall's witness set A for graph $H \cup U$, suppose w.l.o.g. that $A \subseteq L$ and let $B = N_{H \cup U}(A) \subseteq R$. Also define $\bar{A} := L \setminus A$ and $\bar{B} := R \setminus B$.

There must be a matching M in G going from A to \bar{B} such that $|M| = \mu(G) - \mu(H \cup U)$. Additionally, it must hold that $\mu(H \cup U) \geq |\bar{A}| + |B|$ (see [AB19] Eq (1)).

Let H' be the subset of edges of H with one endpoint matched by M . Let P be the subset of vertices of H' that touch M and let Q the rest of the vertices of H' . Note that H' must be bipartite with P and Q being a valid partitioning. Now it can be confirmed that H' has at least

$|P|\beta^-/2$ edges because every edge in M is missing from $H \cup U$ and so its edge-degree by the second property of HEDCS must be at least β^- in H . From the definition of $f(k, \beta, \beta^-)$ applied to graph H' , we get that the Q side of H' has at least $f(k, \beta, \beta^-)|P| = 2f(k, \beta, \beta^-)|M|$ vertices. But since $Q \subseteq \bar{A} \cup B$, this implies that

$$\mu(H \cup U) \geq |\bar{A}| + |B| \geq 2f(k, \beta, \beta^-)|M| \geq 2f(k, \beta, \beta^-)(\mu(G) - \mu(H \cup U)).$$

Moving the terms, we get

$$\mu(H \cup U) \geq \frac{2f(k, \beta, \beta^-)}{2f(k, \beta, \beta^-) + 1}\mu(G) = \alpha(k, \beta, \beta^-)\mu(G). \quad \square$$

Note that [Lemma B.2](#) immediately implies the first part of [Proposition 3.6](#) for bipartite graphs. This can also be extended to the general case using the Lóvasz Local Lemma (LLL) as in [\[AB19\]](#), leading to the following result (which note is slightly different from [Proposition 3.6](#)):

Lemma B.3. *Let H and U be subgraphs of a general (i.e., not necessarily bipartite) graph G and suppose that H is a β -HEDCS $_k$ of $G \setminus U$. Then for some $\gamma = O(\sqrt{\beta \log(k\beta)})$, it holds that $\mu(H \cup U) \geq \alpha(k, \frac{1}{2}\beta + \gamma, \frac{1}{2}\beta - \gamma)\mu(G)$.*

Proof. Fix an arbitrary maximum matching M^* of G and then construct a random bipartite subgraph $\tilde{G} = (L, R, \tilde{E})$ of G by putting one endpoint of each $e \in M^*$ in L and the other in R chosen randomly, and allocating the rest of the vertices not matched by M^* independently and uniformly to L and R . This way, the maximum matching of \tilde{G} remains exactly equal to G . Now let $\tilde{H} := H \cap \tilde{G}$, let (H_1, \dots, H_k) be the hierarchical decomposition of H , and let $\tilde{H}_i = H_i \cap \tilde{G}$.

As in [\[AB19\]](#), the key observation is that the expected degree of every vertex v in \tilde{H}_i is essentially $\deg_{H_i}(v)/2$ up to an additive error of one. Letting $\lambda = 6\sqrt{\log(k\beta)/\beta}$, the Chernoff bound gives

$$\Pr \left[|\deg_{\tilde{H}_i}(v) - \deg_{H_i}/2| \geq \lambda\beta + 1 \right] \leq 2 \exp \left(-\frac{\lambda^2\beta^2}{3\beta} \right) = 2 \exp(-12\log(k\beta)) < (k\beta)^{-10}.$$

Now let $\mathcal{E}_{v,i}$ be the event that $|\deg_{\tilde{H}_i}(v) - \deg_{H_i}/2| \geq \lambda\beta + 1$. Note that $\mathcal{E}_{v,i}$ depends only on the part that vertices in $N_H(v)$ are assigned to. Noting that $|N_H(v)| \leq \beta - 1$ for all v by [Observation 3.1](#), we get that $\mathcal{E}_{v,i}$ depends on at most $\beta^2 k$ other events $\mathcal{E}_{u,j}$. As such, the events $\{\overline{\mathcal{E}_{v,i}}\}_{v \in V, i \in [k]}$ satisfy the LLL condition, and so there is a partitioning where $\cap_{v \in V, i \in [k]} \overline{\mathcal{E}_{v,i}}$ happens. We consider this partitioning from this point on in the analysis.

Let $\gamma := 2\lambda\beta + 3 = O(\sqrt{\beta \log(k\beta)})$. We show that \tilde{H} is a $(\frac{1}{2}\beta + \gamma, \frac{1}{2}\beta - \gamma)$ -HEDCS $_k$ of \tilde{G} . Indeed, we show that $\tilde{H}_1, \dots, \tilde{H}_k$ is a hierarchical decomposition, satisfying the HEDCS properties for \tilde{H} . We just have to prove the two properties of HEDCS holds.

For property (i) of HEDCS, for any $i \in [k]$ and any edge $e = (u, v) \in \tilde{H}_i$ we have $\deg_{\tilde{H}_i}(e) \leq \frac{1}{2}\deg_{H_i}(e) + 2\lambda\beta + 2 \leq \frac{1}{2}\beta + 2\lambda\beta + 2 \leq \frac{1}{2}\beta + \gamma$ where the first inequality holds because of the events $\mathcal{E}_{v,i}$ and $\mathcal{E}_{u,i}$, and the second inequality holds because H is a β -HEDCS $_k$.

For property (ii) of HEDCS, take an edge $e = (u, v) \in \tilde{G} \setminus \tilde{H}$. We have $\deg_{\tilde{H}}(e) \geq \frac{1}{2}\deg_H(e) - 2\lambda\beta - 2 \geq \frac{1}{2}(\beta - 1) - 2\lambda\beta - 2 > \frac{1}{2}\beta - \lambda$, where again the first inequality holds because of the events $\overline{\mathcal{E}_{v,i}}$ and $\overline{\mathcal{E}_{u,i}}$, and the second inequality holds because H is a β -HEDCS $_k$ and $e \in G \setminus H$ by construction.

Plugging [Lemma B.2](#), we thus get $\mu(\tilde{H} \cup U) \geq \alpha(k, \frac{1}{2}\beta + \gamma, \frac{1}{2}\beta - \gamma)\mu(\tilde{G})$. The claim then follows since $\mu(H \cup U) \geq \mu(\tilde{H} \cup U)$ as $\tilde{H} \subseteq H$, and since $\mu(G) = \mu(\tilde{G})$ as discussed. \square

To go from the guarantee of [Lemma B.3](#) to the guarantee of [Proposition 3.6](#), we prove the following claim using the *dependent rounding* scheme of Gandhi *et al.* [[GKPS06](#)].

Claim B.4. *Let $k \geq 1$ and $\beta \geq 1$ be integers, let β be such that $\beta \geq c(\beta'k)^2 \log(\beta'k)$ for some sufficiently large constant $c \geq 1$ and integer $\beta' \geq 2k - 1$, and let γ be as in [Lemma B.3](#). Then it holds that $\alpha(k, \frac{1}{2}\beta + \gamma, \frac{1}{2}\beta - \gamma) \geq (1 - o(1)) \cdot \alpha(k, \beta' + 2k - 1, \beta')$.*

Proof. We first note that to prove this claim, it is sufficient to show that $f(k, \frac{1}{2}\beta + \gamma, \frac{1}{2}\beta - \gamma) \geq f(k, \beta' + 2k - 1, \beta')$. So from here on, we focus on showing this holds.

Take any bipartite graph $G = (P \cup Q, E)$ that is a $(\frac{1}{2}\beta + \gamma)$ -HEDCS $_k$ with at least $|E| \geq (\frac{1}{2}\beta - \gamma)|P|/2$ edges. We construct from G another bipartite graph $G' = (P \cup Q, E')$ that is a $(\beta' + 2k - 1)$ -HEDCS $_k$ with at least $|P|\beta'/2$ edges. It follows then from [Definition 3.4](#) that $f(k, \frac{1}{2}\beta + \gamma, \frac{1}{2}\beta - \gamma) \geq f(k, \beta' + 2k - 1, \beta')$.

To go from G to G' , we first construct a fractional solution G_1 obtained by assigning a weight of $\beta' / (\frac{1}{2}\beta + \gamma)$ to each edge in G . This ensures that G_1 is a *fractional β' -HEDCS $_k$* (where the edge-degree constraints are generalized to fractional edge-degrees) with total weight at least

$$\frac{\beta'}{\frac{1}{2}\beta + \gamma} \cdot |E| \geq \frac{\beta'}{\frac{1}{2}\beta + \gamma} \cdot (\frac{1}{2}\beta - \gamma)|P|/2 = (1 - o(1))\beta'|P|/2,$$

Note that the average degree of the vertices in P is now lower bounded by $(1 - o(1))\beta'/2$ instead of $\beta'/2$ but it is straightforward to show that for any $\beta_1 > \beta_2 > 1$, we have $\alpha(k, \beta_1, (1 - o(1))\beta_2) \geq (1 - o(1)) \cdot \alpha(k, \beta_1, \beta_2)$. So we will now simply focus on showing that we can round the fractional solution above, into an integral one containing at least as many edges such that the degrees of end-points of any edge increases by at most $2k - 1$ as a result of the rounding.

Our rounding algorithm proceeds in iterations such that in the iteration i of rounding, we round all edges whose level is i . In doing this rounding, we will ensure that the degree of an edge at a level higher than i can increase by at most 2 due to the rounding of level i edges. As such this implies that after all k levels have been rounded, the resulting integral solution G' satisfies the property that all edge degrees are bounded by $\beta' + 2k$. However, with just a slightly more careful analysis, we will be able to improve this to $\beta' + (2k - 1)$, giving us the desired result.

We now describe the rounding scheme. We start by considering edges at level 1. We can view this as a fractional solution $x^{(1)}$ (obtained by restricting the fractional solution to only level 1 edges) that induces a fractional degree $x^{(1)}(u)$ at each vertex u . Now using the dependent rounding scheme in [Theorem 2.3](#) of [[GKPS06](#)], we know that there exists an integral solution that has (i) at least as many edges as the weight of x , and (ii) ensures that the degree of each vertex u is either $\lfloor x^{(1)}(u) \rfloor$ or $\lceil x^{(1)}(u) \rceil$. Thus we can obtain an integral solution where the degree of each vertex goes up by only 1 while preserving the total fractional mass of edges at level 1. This means that any edge at levels 1 through k sees an increase of at most 2 in the total degree of its end-points as a result of this rounding. We can now repeat this process on edges at levels 2 through k , ultimately obtaining an integral solution such that degree of each edge in the final rounded solution can be bounded by $\beta' + 2k$.

To improve the bound to $\beta' + (2k - 1)$, we observe the following. Fix any edge (u, v) at some level i . If each of $x^{(1)}(u), x^{(2)}(u), \dots, x^{(i)}(u)$, $x^{(1)}(v), x^{(2)}(v), \dots, x^{(i)}(v)$ are integral, then the dependent rounding scheme leaves these degrees unaltered, and hence the degree of the edge (u, v) in the rounded solution continues to be bounded by β' . On the other hand, if at least one of

$x^{(1)}(u), x^{(2)}(u), \dots, x^{(i)}(u), x^{(1)}(v), x^{(2)}(v), \dots, x^{(i)}(v)$ is fractional, then it must be that before rounding, the degree of the edge (u, v) satisfies

$$\sum_{j=1}^i \left(\lfloor x^{(j)}(u) \rfloor + \lfloor x^{(j)}(v) \rfloor \right) \leq \beta' - 1.$$

It then follows that after rounding, the degree of the edge (u, v) is at most

$$\sum_{j=1}^i \left(\lceil x^{(j)}(u) \rceil + \lceil x^{(j)}(v) \rceil \right) \leq \beta' - 1 + (2i) \leq \beta' + (2k - 1),$$

since $i \leq k$, completing the proof. \square

C Proof of Claim 4.7

Claim 4.7 (restated). *Algorithm 3 reaches Line 5 at most $4\mu_i\beta^2$ times.*

Proof. Algorithm 3 reaches Line 5 every time that it encounters an (H_i, β) -underfull edge. Hence, it suffices to show that the total number of (H_i, β) -underfull edges encountered cannot be $\leq 2\mu_i\beta^2$. Our proof combines a potential function defined previously in [BS16, ABB⁺19, AB19, Ber20] (see e.g., [AB19, Proposition 2.4] or [Ber20, Lemma 4.2]) with a simple additional idea.

Define the following potential functions

$$\Phi_1 := (2\beta - 1)|H_i|, \quad \Phi_2 := \sum_{e \in H_i} \deg_{H_i}(e), \quad \Phi := \Phi_1 - \Phi_2.$$

We first show that Φ is non-negative at the start of Algorithm 3 after we set $H_i \leftarrow H_{i-1}$. Recall from Observation 4.6 that the maximum degree in H_{i-1} is at most β . This means that $\Phi_2 \leq \beta|H_{i-1}|$, implying

$$\Phi \geq (2\beta - 1)|H_{i-1}| - \beta|H_{i-1}| = (\beta - 1)|H_{i-1}| \geq 0.$$

The key observation is that every time that we insert an (H_i, β) -underfull edge to H_i or remove an (H_i, β) -overfull edge from H_i , the value of Φ increases by at least 1. The proof of this part is completely due to [BS16, ABB⁺19, AB19, Ber20] and proceeds as described next.

Suppose that we remove an (H_i, β) -overfull edge e from H_i . This reduces Φ_1 by exactly $2\beta - 1$. Let us now analyze the change to Φ_2 . On the one hand, removing e from H_i decreases Φ_2 by at least $\beta + 1$ because e no longer participates in the sum and its edge-degree before removing it was at least $\beta + 1$ for being (H_i, β) -overfull. On the other hand, e must have had at least $\deg_{H_i}(e) - 2 \geq \beta - 1$ incident edges in H_i before being deleted. Deleting e reduces the edge-degree of each of these edges by one, and so Φ_2 overall decreases by $\beta + 1 + \beta - 1 = 2\beta$. Since Φ_1 decreases by exactly $2\beta - 1$ and Φ_2 decreases by at least 2β , Φ increases by at least 1.

Now consider inserting an (H_i, β) -underfull edge e into H_i . This increases Φ_1 by exactly $2\beta - 1$. Now on the one hand, inserting e increases Φ_2 by at most β because e will now participate in the sum and its edge-degree before inserting it was at most $\beta - 2$ for being (H_i, β) -underfull. On the other hand, e has at most $\beta - 2$ incident edges in H_i (or else it would not have been (H_i, β) -underfull before being inserted) and adding e increases their edge-degrees by one. Therefore in

total Φ_2 increases by at most $\beta + (\beta - 2) = 2\beta - 2$. Since Φ_1 increases by exactly $2\beta - 1$ and Φ_2 increases by at most $2\beta - 2$, the value of Φ increases by at least 1 in this case too.

In the next step of the proof, we show that $\Phi \leq 2\mu_i\beta^2$ at all times. From [Claim 4.4](#) recall that $H_i \subseteq G_i$. Since μ_i is the size of a maximal matching of G_i , there exists a vertex cover W of G_i (and thus H_i) with size $|W| = 2\mu_i$. Since each vertex in H_i has maximum degree β by [Observation 4.6](#) and each of these edges are connected to W , we get $|H_i| \leq 2\mu_i\beta$. This in particular implies $\Phi_1 \leq (2\beta - 1)2\mu_i\beta \leq 4\mu_i\beta^2$. By non-negativity of Φ_2 , we get $\Phi \leq 4\mu_i\beta^2$.

To finish the proof, note on the one hand that in every iteration that we encounter an (H_i, β) -underfull edge, we add it to H and thus the value of Φ increases by at least 1 by our discussion above. On the other hand, the value of Φ is upper bounded by $4\mu_i\beta^2$ as discussed. Hence, the number of (H_i, β) -underfull edges encountered by Algorithm 3 is $\leq 4\mu_i\beta^2$, which as discussed at the beginning, completes the proof. \square