Beating Greedy Matching in Sublinear Time

Soheil Behnezhad*   Mohammad Roghani   Aviad Rubinstein†   Amin Saberi

Abstract

We study sublinear time algorithms for estimating the size of maximum matching in graphs. Our main result is a $(\frac{1}{2} + \Omega(1))$-approximation algorithm which can be implemented in $O(n^{1+\varepsilon})$ time, where n is the number of vertices and the constant $\varepsilon > 0$ can be made arbitrarily small. The best known lower bound for the problem is $\Omega(n)$, which holds for any constant approximation.

Existing algorithms either obtain the greedy bound of $\frac{1}{2}$-approximation [Behnezhad FOCS'21], or require some assumption on the maximum degree to run in $o(n^2)$-time [Yoshida, Yamamoto, and Ito STOC'09]. We improve over these by designing a less "adaptive" augmentation algorithm for maximum matching that might be of independent interest.

Contents

1 Introduction

Linear-time algorithms have long been considered the gold standard in algorithm design. With the rapid increase in the size of data, however, even linear-time algorithms may be slow in some settings. A natural question is whether it is possible to solve a problem of interest in *sublinear time* in the input size. That is, without even reading the whole input. In this work, we focus on the problem of estimating the size of *maximum matching* in sublinear time. Recall that a *matching* is a set of edges no two of which share an endpoint, and a *maximum matching* is a matching of the largest size. This is a central problem in the study of sublinear time algorithms and several general techniques of the area have emerged from the study of matchings [PR07, NO08, YYI09, ORRR12, KMNT20, CKK20, Beh21].

There is a simple greedy algorithm for constructing a *maximal* (but not necessarily *maximum*) matching: Iterate over the edges and greedily add each edge whose endpoints are yet unmatched. Every maximal matching is at least half the size of a maximum matching. Thus, this simple algorithm is a 1/2-approximation. Note, however, that the algorithm's running time is not sublinear in the input size as we have to go over all the possibly $\Omega(n^2)$ edges one by one where n is the number of vertices. In fact, there are lower bounds showing that $\Omega(n^2)$ time is necessary to *find* any $O(1)$-approximate matching. Nonetheless, approximating the *size* of the maximum matching can be done much faster. Indeed, a beautiful line of work in the literature [NO08, YYI09, ORRR12, Beh21] led to an $\widetilde{O}(n)$ time algorithm for estimating the size of a (random) greedy maximal matching [Beh21].

Unfortunately, the main shortcoming of the greedy maximal matching algorithm is that it only provides a 1/2-approximation, even when the edges are processed in a random order [DF91]. There are techniques to improve the approximation by "augmenting" the greedy matching [YYI09], but such techniques only work well when the degrees in the graph are rather small. In particular, when we go even slightly above 1/2-approximation, then all known algorithms take a (large) polynomial time in the maximum degree. Unfortunately, this can be as large as $\Omega(n^2)$ for general n-vertex graphs which is no longer sublinear time in the input size. This raises a natural question:

Question 1. *Is it possible to $(\frac{1}{2} + \Omega(1))$-approximate maximum matching size in $n^{2-\Omega(1)}$ time?*

We remark that this question has been open even for bipartite graphs.

In this work, we answer Question 1 in the affirmative. The running time of our algorithm can, in fact, be made abritrarily close to linear in n. Our algorithm can be adapted to both the *adjacency list* and *adjacency matrix* query models, which are the two standard graph representations studied in the literature. We also consider both multiplicative approximations as well as multiplicative-additive approximations. See Section 3 for the formal definitions of these query models and approximations.

Using n, m, Δ, and $\bar{d}$ to respectively denote the number of vertices, the number of edges, the maximum degree, and the average degree in the graph, our results can be summarized as follows:

Theorem 1.1. *For any constant $\varepsilon > 0$, there is a constant $\delta > 2^{-O(1/\varepsilon)}$ along with an algorithm that w.h.p. estimates the size of maximum matching up to a:*

(1) *multiplicative factor of $(\frac{1}{2} + \delta)$ in the <u>adjacency list</u> model in $\widetilde{O}(n + \Delta^{1+\varepsilon})$ time,*

(2) *multiplicative-additive factor of $(\frac{1}{2} + \delta, o(n))$ in the <u>adjacency list</u> model in $\widetilde{O}(\bar{d} \cdot \Delta^{\varepsilon})$ time,*

(3) *multiplicative-additive factor of $(\frac{1}{2} + \delta, o(n))$ in the <u>adjacency matrix</u> model in $O(n^{1+\varepsilon})$ time.*

A few remarks about the three results of Theorem 1.1:

- The constant $\delta > 0$ in Theorem 1.1 is miniscule. We did not attempt to optimize it, but do not expect our techniques to lead to a better than, say, .51-approximation in $n^{2-\Omega(1)}$ time.

- Theorem 1.1–(1) comes close to an $\Omega(n)$ lower bound that holds for any $O(1)$-approximation in the adjacency list model. Any such algorithm must distinguish an empty graph from one with a single edge. This clearly requires $\Omega(n)$ queries in the adjacency list model.

- Theorem 1.1–(2) comes close to an $\Omega(\bar{d})$ lower bound for any $(O(1), o(n))$-approximation in the adjacency list model due to [PR07]. Observe that $\bar{d}\Delta^\varepsilon \ll m$ for any $\varepsilon < 1$. Therefore, this algorithm *always* runs in sublinear time in the number of edges in the graph.

- Theorem 1.1–(3) comes close to an $\Omega(n)$ lower bound for any $(O(1), o(n))$-approximation in the adjacency matrix model. Any such algorithm must distinguish an empty graph from one that includes a random perfect matching. This requires $\Omega(n)$ adjacency matrix queries.

- It takes $\Omega(n^2)$ queries to the adjacency matrix to distinguish an empty graph from one with only a single edge. Since this must be done for any multiplicative $O(1)$-approximation, no non-trivial such algorithm (i.e., one with $o(n^2)$ queries) exists for this model.

On Beating Greedy Matching in Various Settings

The greedy 1/2-approximation is a prevalent barrier for maximum matching across various settings. As a result, numerous works in the literature study the possibility of beating it — both on the upper bound side as well as the lower bound side. The answer is not always the same. For instance, for the online model under *edge arrivals*, [GKM+19] showed that 1/2 is provably the best achievable approximation, which can be trivially matched by the greedy algorithm. There are also settings where the answer remains unknown, despite a significant research effort. For instance, in the single-pass streaming setting, beating the greedy 1/2-approximation in $\widetilde{O}(n)$ (or even subquadratic) space has been open for nearly two decades [FKM+05], and is often considered as one of the most fundamental open problems of the area. Finally, there are settings for which the greedy 1/2-approximation has been broken. Various models of the online setting [FHTZ20, GKM+19], the random-order streaming setting [KMM12], and the stochastic matching setting [AKL17] are examples of this. In many of these settings, the approximation has been improved well beyond 1/2 after the greedy bound was first broken. For instance, in the random-order streaming the current best known bound is slightly above 2/3 [AB21], and in the stochastic matching setting a $(1 - \varepsilon)$-approximation has been achieved [BDH20]. We hope that our work in this paper also inspires future work on going tangibly above 1/2-approximation in the sublinear time model.

Discovering short *augmenting paths* has been a central technique in many of the works discussed above in beating the greedy algorithm. What varies significantly is whether it is possible to find these augmenting paths effeciently in the particular model at hand. In particular, a common approach is to first construct a maximal matching in full, and then augment it via the vertices left unmatched. This "adaptivity" complicates things in our model, making it hard to estimate the size of the solution in subquadratic time. One of our main contributions in this work is to give a less "adaptive" algorithm that interleaves the construction of a maximal matching and the augmentation phase. We believe this technique, which is overviewed in Section 2, might be of independent interest.

2 Technical Overview

In Section 2.1, we first give a brief overview of existing approaches and discuss why the take quadratic time to implement. Then, in Section 2.2, we overview our main tool in breaking this quadratic time barrier through a less "adaptive" augmentation algorithm.

2.1 The Quadratic Barrier: A Brief Discussion of Earlier Techniques

Discovering (short) *augmenting paths*[1] is a natural way of improving the approximation for the maximum matching problem. For example, the Hopcroft-Karp [HK73] algorithm starts with an empty matching and iteratively applies a maximal set of vertex disjoint (short) augmenting paths. Each step of Hopcroft-Karp can essentially be viewed as a *maximal independent set* (MIS) instance. Put one vertex for each (short) augmenting path and connect two vertices if their corresponding augmenting paths share a vertex. An MIS in this graph corresponds to a maximal set of vertex disjoint augmenting paths. Building on this idea and by giving a size estimator for MIS, Yoshida, Yamamoto, and Ito [YYI09] showed that for any integer $k \geq 1$, a $(\frac{k}{k+1}, o(n))$-approximation can be obtained in $O_k(\Delta^{6k(k+1)})$ time[2]. This is sublinear when Δ is sufficiently small.

Although the MIS-based approach is powerful enough to go well beyond 1/2-approximation in poly(Δ) time, it does not seem to help with general graphs where Δ can be large. This holds even if we limit ourselves to length-3 augmenting paths, which is needed for beating 1/2. The MIS size estimator of [YYI09] crucially requires time at least linear in the average degree. Since every edge of a maximal matching can belong to $\Omega(\Delta^2)$ length-3 augmenting paths (with $\Omega(\Delta)$ choices from each endpoint of the edge), this average degree in the MIS graph can be $\Omega(\Delta^2)$ where Δ is the original graph's maximum degree. This makes it unlikely for this approach to yield an $o(\Delta^2)$ time algorithm. Additionally, not being able to construct the MIS graph explicitly in whole, and not having the edges of the maximal matching we are trying to augment also impose other Δ factors in the running time, arriving at the rather large poly(Δ) bound of [YYI09].

There is an alternative way of discovering length-3 augmenting path which works directly with matchings instead of independent sets. The idea is to first find a maximal matching M of G, then find another maximal matching S on a subgraph H of G which includes a subset of the edges that have exactly one endpoint matched by M. Note that if both endpoints of an edge $e \in M$ are matched in S, then we get a length-3 augmenting path. This framework was first used by [KMM12] in the context of random-order streaming algorithms, but has since been applied to various other settings [BHN16, BLM20, GS17]. Because the second graph H is defined *adaptively* based on M, we cannot simply run two independent instances of existing maximal matching estimators as black-box. In fact, this framework also hits a quadratic-in-degree time barrier as we describe next. To describe this barrier, we first briefly overview the key ideas behind the maximal matching size estimator of [Beh21].

Consider a maximal matching S that is constructed greedily by iterating over the edges in some ordering π. It is not hard to see that $e \in S$ iff there is no edge e' incident to e such that $\pi(e') < \pi(e)$ and $e' \in S$. Therefore to determine whether $e \in S$, it suffices to go over the lower rank neighboring edges of e (in the increasing order of their ranks) and recursively query them to either find one that belongs to S, or conclude that e must be in S. This approach was first suggested by Nguyen and Onak [NO08]. The main question is the total number of recursive calls needed for the process to

---

[1]See Section 3 for the formal definition of augmenting paths.
[2]O_k ignores dependencies on k.

finish. The result of Behnezhad [Beh21] is that for a *random* vertex v and a *random* permutation π, the process terminates in $O(\bar{d} \cdot \log n)$ expected total time, coming close to an $\Omega(\bar{d})$ lower bound.

Now, let us revisit the above two-step algorithm which first constructs a maximal matching M of G and then another maximal matching S of a subgraph H of G. Consider the task of determining whether a vertex v is matched by S. From [Beh21], we get that for a random vertex v, this should be doable by exploring $\widetilde{O}(\bar{d}_H)$ edges of H in expectation where here we use $\bar{d}_X$ to denote the average degree of graph X. The challenge, however, is that since H is defined adaptively based on M, we do not a priori know the neighbors of v in H. In particular, to know whether an edge (u, v) exists in H, we have to ensure that exactly one of u and v is matched in M. Therefore, for exploring $\widetilde{O}(\bar{d}_H)$ edges in H, the naive approach would make $\widetilde{O}(\bar{d}_H)$ vertex queries to M. Note that these calls are not necessarily to random vertices anymore, which is crucial for the bound of [Beh21] to work. But even if we manage to get an $\widetilde{O}(\bar{d}_G)$ time bound on each one of these calls we arrive at a total running time of $\widetilde{\Theta}(\bar{d}_H \cdot \bar{d}_G)$ which again can be as large as $\Omega(n^2)$.
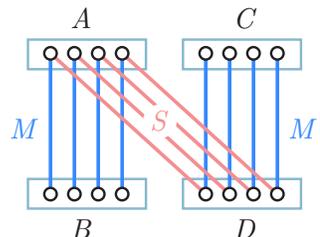
2.2 Our Contribution: A Less "Adaptive" Augmentation Algorithm

The two-stage algorithm we discussed earlier, constructs M fully and then adaptively picks the edges of S based on M. Our first step towards proving Theorem 1.1 is introducing a less "adaptive" algorithm that interleaves the construction of the two matchings M and S.

Our algorithm starts by constructing a sequence T, containing a single element (e, EXTEND) and $K \geq 1$ distinct elements (e, START), corresponding to every edge e in the graph, where K is a parameter of the algorithm. We will process T in a *random order*. The role of having multiple copies of the START elements is to bias them to appear earlier in the random permutation. We start by initializing two empty matchings M and S and then iterate over these randomly sorted $m(K+1)$ elements of T. Whenever we see an (e, START) element, we add e to M iff both endpoints of e are unmatched in M. Therefore, M will be a random greedy maximal matching of G. Whenever we see an (e, EXTEND) element, we add e to S under a few conditions. The first condition is that both endpoints of e must be yet unmatched by S; this is to ensure that S continues to be a matching. The second condition is that at most one endpoint of e can be already matched by M. Our final algorithm, formalized as Algorithm 1, also checks two more technical conditions before adding e to S which are needed for the approximation ratio analysis. Once all of T is processed, the algorithm returns a maximum matching of $M \cup S$.

Observe that even though at the time of adding an edge (u, v) to S, at most one of its endpoints is matched in M, the other endpoint may get matched in M later in the process. Such edges of S cannot be used in length-3 augmenting paths for M. One way to avoid these bad events is to set K large enough so that all the START elements appear before all EXTEND elements. However, this will negatively impact the running time. Specifically, the local query process to determine whether a random vertex v is matched in either of S or M takes $\widetilde{O}(\bar{d} \cdot K)$ time. This means that we need to set K to be much smaller than Δ to beat the quadratic-time-barrier. Indeed we set $K \approx \Delta^\varepsilon$ to get the bounds of Theorem 1.1.

When $K \ll \Delta$, we will inevitably have many edges of S for which both endpoints are matched in M. For example, consider the construction illustrated on the right with four vertex parts A, B, C, D. There is a regular bipartite graph of degree $\Theta(\Delta)$ between A and B, a regular bipartite graph of degree $\Theta(\Delta^{0.99})$ between A and D, and a regular bipartite graph of degree $\Theta(\Delta^{0.98}/K)$ between C and D. In

4

this construction, the edges of M match A to B and C to D nearly completely. The edges of S match A to D nearly completely. This happens because there are many more EXTEND elements in (A, D) than there are START elements in (C, D). Therefore, at the time of adding S, the part of M from C to D is not yet constructed.

Despite this bad event, we show that the edges of S are still useful in augmenting M. One key insight apparent in the above example is that the edges of (C, D) in M tend to have rank (in the permutation of T) roughly K times larger than those edges of (A, B) in M. We generalize this to all graphs and show that if an edge of S connects two edges of M, then the ranks of these edges of M must differ by a factor of roughly K.[3] Therefore, if instead of considering all edges of M for augmentation, we consider a subset $M_{j^\star}$ of M that is a constant fraction of M and at the same time any two edges in $M_{j^\star}$ have ranks within K factor of each other, then S cannot connect two edges of $M_{j^\star}$ and so we can focus on augmenting just $M_{j^\star}$ instead of the whole matching M. See Sections 4.1 and 4.2 for the full details of the algorithm and its approximation analysis.

So far we have only described an algorithm that finds a $(\frac{1}{2} + \Omega(1))$-approximate matching and have not yet described how to estimate its output size in sublinear time. To do this, we first define a query process akin to the one described above for maximal matching. That is, for a given edge e we define two query processes that respectively return whether $e \in M$ and $e \in S$. We then analyze the expected number of the recursive calls for a random start vertex by building on the techniques of [Beh21, YYI09]. Several challenges arise along the way that are unique to our algorithm and require new ideas. For instance, an EXTEND element remains relevant (i.e., can still be added to S) until seeing two START elements in M, one from each endpoint. This is unlike the greedy approach, say for MIS or maximal matching, where an element (respectively a vertex and an edge) becomes irrelevant right after seeing one neighbor in the solution. Also note that we should not simply count the number of edges in M and S. Rather, we have to count the number of edges of M plus the number of length-3 augmenting paths that we find. To do this, when we find an edge $(u, v) \in S$ and an edge $(v, w) \in M$, we also query whether w is matched in S to a vertex left unmatched by M or not. This complicates the analysis because this vertex w is not picked uniformly at random anymore. The details of the query process and its analysis are provided in Section 5.

3 Preliminaries

Notation: Throughout the paper we use $G = (V, E)$ to denote the input graph. We use n to denote the number of vertices in G, m to denote the number of edges in G, Δ to denote the maximum degree of G, and $\bar{d}$ to denote the average degree of G. We write $\mu(G)$ to denote the size of the maximum matching in G.

We use $A \oplus B := (A \cup B) \setminus (A \cap B)$ to denote the symmetric difference of two sets A and B. Also for any positive integer k, we use $[k]$ to denote the set $\{1, \ldots, k\}$. Throughout the paper, we use the $\widetilde{O}(\cdot)$ to suppress poly $\log n$ factors, that is $\widetilde{O}(f) = O(f \cdot \text{poly} \log(n))$.

Problem Definition: Given a graph G, represented in one of the following two ways, we study the problem of estimating the size of maximum matching:

- *Adjacency List:* In this model, the neighbors of each vertex are stored in a list sorted in an arbitrary order. Each query of the algorithm specifies a vertex v and an index i. The answer

---

[3]To be more precise, we only prove this for most edges of S, but not all.

is the ID of the i-th vertex in the list of v's neighbors, or *empty* if v has less than i neighbors.

- *Adjacency Matrix:* In this model, each query of the algorithm specifies a pair of vertices u and v. The answer is 1 if u and v are adjacent, and 0 otherwise.

For $\alpha \in (0, 1]$ and $\gamma \in [0, 1]$, we say $\widetilde{\mu}(G)$ is a multiplicative-additive $(\alpha, \gamma n)$-approximation of the size of maxmimum matching of G if $\alpha\mu(G) - \gamma n \leq \widetilde{\mu}(G) \leq \mu(G)$. Additionally, it is a multiplicative α-approximation if $\alpha\mu(G) \leq \widetilde{\mu}(G) \leq \mu(G)$.

Augmenting/Alternating Paths: Given a matching M of G, a path in G is an *alternating path* for M if its edges alternatively belong to M. An alternating path is an *augmenting path* for M if the first and the last edges of the path do not belong to M.

It is well-known that if a maximal matching is nearly half the size of a maximum matching, then almost all of its edges belong to length-three augmenting paths. The following statement is folklore. For the sake of completeness, we provide a simple proof in Appendix A.

Claim 3.1 (Folklore)**.** *Let M be a maximal matching and $M^\star$ a maximum matching. Suppose $|M| < (\frac{1}{2} + \delta)|M^\star|$. In $M \oplus M^\star$, there are at least $|M| - 4\delta|M^\star|$ length-3 augmenting paths for M.*

Greedy Matching: Given a graph $G = (V, E)$ and a permutation π of its edge-set E, we use $\mathsf{GreedyMM}(G, \pi)$ to denote the greedy maximal matching obtained by iterating over the edges of E in the order of π and greedily adding each encountered edge that does not violate matching constraints to the matching.

We use the following proposition about the size of greedy matchings in vertex-subsampled subgraphs. It was first proved in [BLM20] using the techniques developed in [KMM12].

Proposition 3.2 ([BLM20, Lemma 5.2])**.** *Let $G(V, U, E)$ be a bipartite graph, let π be an arbitrary permutation over E, let $p \in (0, 1)$, and let M be an arbitrary matching in G. Let W be a subsample of V including each vertex independently with probability p. Define X to be the number of edges in M whose endpoint in V is matched in $\mathsf{GreedyMM}(G[W \cup U], \pi)$; then*

$$\mathbf{E}_W[X] \geq p(|M| - 2p|V|).$$

Probabilistic tools: We use the following version of Chernoff bound.

Proposition 3.3 (Chernoff Bound)**.** *Suppose $X_1, X_2, \ldots, X_n$ are independent Bernoulli random variables and $X = \sum_{i=1}^{n} X_i$. For any $t > 0$, we have*

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

4   A Meta Algorithm for Beating the $\frac{1}{2}$-Approximation

4.1   The Algorithm

In this section, we formalize our new "less adaptive" meta algorithm that we informally overviewed in Section 2. We show in Section 4.2 that its approximation ratio is strictly better-than-half. We later show in Section 5 that the size of its output matching can be estimated in sublinear time.

Before formalizing the algorithm, let us give a few useful definitions. Given an n-vertex graph of maximum degree Δ, and a parameter $\varepsilon \in (0, .25)$, define:

- $p := 0.007$.
- $D := (c \cdot \Delta \cdot \log n)^\varepsilon$ where we later fix $c \geq 1$ to be sufficiently large function of ε.
- $K := 10D \log^2 n$.
- $\alpha_i := 1/D^{i-1}$ for $i \in [2/\varepsilon]$ and $\alpha_{2/\varepsilon+1} = 0$. Note that $0 = \alpha_{2/\varepsilon+1} < \alpha_{2/\varepsilon} < \ldots < \alpha_2 < \alpha_1 = 1$.

We are now ready to state the algorithm, which is formalized below as Algorithm 1.

Algorithm 1: An algorithm for beating half-approximate matching.

Input: An n-vertex m-edge graph $G = (V, E)$ of max degree Δ. **Parameter:** $\varepsilon \in (0, .25)$.

1   Define K, p, and $\alpha_{2/\varepsilon}, \ldots, \alpha_1$ as above.
2   Construct a sequence T, which for any edge $e \in E$ includes K copies of (e, START) and one copy of (e, EXTEND). Then random shuffle the elements in T.
3   For any vertex $v \in V$ pick a color $c_v \in \{\text{BLUE}, \text{RED}\}$ uniformly and independently.
4   Initialize $M \leftarrow \emptyset, S \leftarrow \emptyset$.;   // Both M and S will be matchings of G.
5   Initialize $M_1 \leftarrow \emptyset, \ldots, M_{2/\varepsilon} \leftarrow \emptyset$.;   // These will partition the edges in M.
6   Draw $j^\star$ from $[2/\varepsilon]$ uniformly at random.
7   **for** $i = 1$ *to* $|T|$ **do**
8   Let $(e = \{u, v\}, X)$ be the i-th element in T.
9   **if** $X = \text{START}$ *and* $\deg_M(u) = \deg_M(v) = 0$ **then**
10   Add e to M.
11   Add e to the unique M_i, $i \in [2/\varepsilon]$ where $\alpha_{i+1}|T| < i \leq \alpha_i|T|$.
12   **if** $c_v = c_u$ *or* $e \notin M_{j^\star}$ **then**
13   Mark both u and v as *frozen*.
14   **else**
15   With probability $1 - p$ mark both u and v as *frozen*.
16   **if** $X = \text{EXTEND}$, $\deg_S(u) = \deg_S(v) = 0$, $\deg_M(v) + \deg_M(u) \leq 1$, $c_u \neq c_v$, *and neither endpoint of e is frozen* **then**
17   Add e to S.
18   **return** the maximum matching in $M \cup S$.

4.2   The Approximation Guarantee

In this section, we prove the following approximation guarantee for Algorithm 1. (We note that we have not attempted to optimize the constants in the statement.)

Theorem 4.1. *Let G be any n-vertex graph. Let M and S be the matchings produced by Algorithm 1*

run on G for parameter $\varepsilon \in (0, .25)$. Then for some $\delta > 2^{-O(1/\varepsilon)}$,

$$\mathbf{E}[\mu(M \cup S)] \geq \left(\frac{1}{2} + \delta\right)\mu(G).$$

4.2.1 Basic Notation and Definitions

For any element ℓ we use $\pi(\ell)$ to denote the location of ℓ in T. For any edge $e \in E$ we use $\pi_{\text{START}}(e)$ (resp. $\pi_{\text{EXTEND}}(e)$) to denote the minimum $i \in [|T|]$ such that the i-th index of T includes element (e, START) (resp. (e, EXTEND)). For any element $\ell \in T$ we say "*at the time of processing ℓ*" to refer to the iteration of the for loop in Algorithm 1 when i equals $\pi(\ell)$.

Next, we define unusual edges as follows:

Definition 4.2 (unusual edges). *We say an edge $e \in M$ is* unusual, *if there is some edge $e' = (u, v)$ incident to e such that one of the following holds:*

- $\pi_{\text{START}}(e) < \pi_{\text{EXTEND}}(e') < D \cdot \pi_{\text{START}}(e)$, *and at the time of processing (e', EXTEND) the only edge in M that is incident to e' is e.*

- $\pi_{\text{EXTEND}}(e') < \pi_{\text{START}}(e)$, *and at the time of processing (e', EXTEND) no edge in M is incident to e' in M.*

We use A to denote the subset of unusual edges of M.

In Section 4.2.4 we prove the following Lemma 4.3 which shows only a small fraction of the edges in M will be unusual. This is one of our key insights towards our proof of Theorem 4.1, and is the main place where having K copies of (e, START) compared to one copy of (e, EXTEND) in T is used crucially.

Lemma 4.3. $\mathbf{E}|A| = o(|M|).$

4.2.2 The Main Argument

In this section, we present the main building blocks of the proof of Theorem 4.1, deferring the proof of one key lemma (Lemma 4.9) to a later section.

Our proof of Theorem 4.1 relies on four independent sources of randomization, which with a slight abuse of notation we denote by $j^\star$, T, C, and F:

- T: The order in which Algorithm 1 processes T.

- C: The colors assigned to the vertices in Line 3 of Algorithm 1.

- $j^\star$: The index chosen in Line 6 of Algorithm 1.

- F: The set of edges in M that get frozen in Line 15 of Algorithm 1.

All four sources of randomization are needed for the proof of Theorem 4.1. But it would be convenient to first condition on T because:

8

Observation 4.4. *Conditioning on T fully reveals the maximal matching M, the set A of unusual edges in M, and all of $M_1, \ldots, M_{2/\varepsilon}$.*

Note, however, that S remains random even after conditioning on T as it depends on the other three sources of randomization too.

Because M is a maximal matching of G, we immediately get $|M| \geq \mu(G)/2$. Our plan is to show that if M is only half the size of $\mu(G)$, then in expectation S augments it well enough that M and S together include a larger matching. More formally, recall that our goal in Theorem 4.1 is to prove that $\mathbf{E}[\mu(M \cup S)] \geq (\frac{1}{2} + \delta)\mu(G)$. This clearly holds if $|M| \geq (\frac{1}{2} + \delta)\mu(G)$. So let us for the rest of the proof assume that M is smaller.

Assumption 4.5. $|M| < (\frac{1}{2} + \delta)\mu(G)$.

Plugging this assumption into Claim 3.1 gives:

Observation 4.6. *At least $|M| - 4\delta|M^\star|$ edges of M belong to length-3 augmenting paths in $M \oplus M^\star$.*

Our next claim shows that there is one subset M_j of M that has several nice properties. We will later argue that M_j will be well augmented by S in expectation.

Claim 4.7. *Define $q(x) := 2^{20(x-3/\varepsilon)}$. There is M_j such that all the following hold:*

1. *$|M_j| \geq q(j)|M|$,*

2. *$|M_1| + \ldots + |M_{j-1}| \leq 2^{-19}|M_j|$,*

3. *For any two edges $e, e' \in M_j$, $\pi_{\text{START}}(e')/D \leq \pi_{\text{START}}(e) \leq \pi_{\text{START}}(e') \cdot D$.*

Proof. First, there should exist $j \in [2/\varepsilon]$ such that $|M_j| \geq q(j)|M|$ as otherwise

$$|M_1| + \ldots + |M_{2/\varepsilon}| < \sum_{i=1}^{2/\varepsilon} q(i)|M| = \frac{2^{20} + \ldots + 2^{40/\varepsilon}}{2^{60/\varepsilon}}|M| < |M|,$$

which contradicts the fact that M_i's partition M. Take the smallest j with $|M_j| \geq q(j)|M|$, noting that the first property of the lemma is satisfied for M_j. We have

$$|M_1| + \ldots + |M_{j-1}| < \sum_{i=1}^{j-1} q(i)|M| < \frac{2^{20} + \ldots + 2^{20(j-1)}}{2^{60/\varepsilon}}|M| < 2^{20j-19-60/\varepsilon}|M| < 2^{-19}|M_j|,$$

so the second property also holds for M_j.

For the third property, note from the definition of $\alpha_1, \ldots, \alpha_{2/\varepsilon+1}$ that if $j \neq \frac{2}{\varepsilon}$, then all edges e in M_j have the same $\pi_{\text{START}}(e)$ up to a factor of D. So it suffices to show $j \neq \frac{2}{\varepsilon}$. Observe that for every $e \in M_{2/\varepsilon}$, by definition we have

$$\pi_{\text{START}}(e) \leq \alpha_{\frac{2}{\varepsilon}}|T| = \frac{|T|}{D^{2/\varepsilon-1}} \leq \frac{2mK}{D^{2/\varepsilon-1}} = \frac{20m \log n}{D^{2/\varepsilon-1}} = \frac{20m \log n}{D^{2/\varepsilon-2}} = \frac{20m \log n}{(c\Delta \log n)^{\varepsilon(2/\varepsilon-2)}} < \frac{20m \log n}{c\Delta \log n} = \frac{20m}{c\Delta}.$$

Now by choosing c to be a sufficiently large function of ε, we can further guarantee that

$$\pi_{\text{START}}(e) < \frac{q(1)m}{4\Delta} \leq q(1)|M|,$$

9

where the last inequality follows because[4] $\mu(G) \geq \frac{m}{2\Delta}$ and $|M| \geq \mu(G)/2$. Since there are less than $q(1)|M|$ edges e for which $\pi_{\text{START}}(e) < q(1)|M|$, we get $|M_{2/\varepsilon}| < q(1)|M|$. Combined with the first property of the claim that $|M_j| \geq q(j)|M|$ and given that $q(x) \geq q(1)$ for every $x \geq 1$, we get that $j \neq 2/\varepsilon$, implying the third property and completing the proof. $\qquad \square$

Now consider the set P_j all length three augmenting paths in $M^\star \oplus M$ where the middle edge belongs to M_j and that the vertices along these paths are alternatively BLUE and RED as follows:

$$P_j := \left\{ (x, u, v, y) \,\middle|\, \begin{array}{l} (x, u, v, y) \text{ is an augmenting path for } M, (x, u) \in M^\star, (v, y) \in M^\star, \\ (u, v) \in M_j, \, c_x = \text{RED}, \, c_u = \text{BLUE}, \, c_v = \text{RED}, \, c_y = \text{BLUE} \end{array} \right\}.$$

Observation 4.8. *Conditioning on T and C fully reveals P_j.*

Proof. Definition P_j only depends on the vertex colors which are determined by C, and matchings M_j and M which are fully determined by T. $\qquad \square$

The following lemma, which conditions on everything except F, is the key to Theorem 4.1. We defer its proof to Section 4.2.3.

Lemma 4.9. *Let us condition on T, C, $j^\star = j$, and let P_j and M_j be as above. Let Y denote the number of length three augmenting paths for M in $M \oplus S$. Then*

$$\mathbf{E}_F\left[Y \mid T, C, j^\star = j\right] \geq p|P_j| - \left(4p^2 + 2^{-18}\right)|M_j| - 12|A|.$$

Let us first see how Lemma 4.9 implies Theorem 4.1.

Proof of Theorem 4.1. We assume Assumption 4.5 holds, or otherwise the theorem is trivial. Recall from Observation 4.6 that at most $4\delta\mu(G)$ edges of M (and thus M_j) are not in length-three augmenting paths in $M \oplus M^\star$. Since the colors are random and independent, each of these length-three augmenting paths is colored in the way specified in the definition of P_j with probability exactly $1/2^4$. Hence,

$$\mathbf{E}_C[|P_j|] \geq \frac{1}{2^4}(|M_j| - 4\delta\mu(G)). \tag{1}$$

Additionally, recall from Claim 4.7 that

$$|M_j| \geq q(j)|M| \geq q(1)|M| \geq \frac{q(1)}{2}\mu(G). \tag{2}$$

Also recall from Lemma 4.3 that
$$\mathbf{E}_T[|A|] = o(\mu(G)). \tag{3}$$

Taking expectation over C and T from both sides of the inequality of Lemma 4.9, we get

$$\mathbf{E}_{F,C,T}[Y \mid j^\star = j] \geq \mathbf{E}_{C,T}\left[p|P_j| - \left(4p^2 + 2^{-18}\right)|M_j| - 12|A|\right]$$
$$\geq \frac{p}{16}(|M_j| - 4\delta\mu(G)) - \left(4p^2 + 2^{-18}\right)|M_j| - o(\mu(G)) \qquad \text{(By (1) and (3))}$$

---

[4]Edge color the graph greedily using 2Δ colors and pick the largest color class which will be a matching.

$$= \left(\frac{p}{16} - 4p^2 - 2^{-18} \right) |M_j| - \frac{p\delta}{4} \mu(G) - o(\mu(G))$$

$$\geq \left(\left(\frac{p}{16} - 4p^2 - 2^{-18} \right) \frac{q(1)}{2} - \frac{p\delta}{4} - o(1) \right) \mu(G) \qquad \text{(By (2).)}$$

$$> \left(10^{-4} q(1) - 0.002 \cdot \delta - o(1) \right) \mu(G) \qquad \text{(Since } p = 0.007.\text{)}$$

$$> \frac{2\delta}{\varepsilon} \mu(G). \qquad \text{(Since } q(1) = 2^{20-60/\varepsilon}, \, \delta = 2^{-70/\varepsilon}.\text{)}$$

This, in turn, implies that

$$\mathbf{E}_{F,C,T,j^\star}[Y] \geq \Pr[j^\star = j] \cdot \mathbf{E}_{F,C,T}[Y \mid j^\star = j] \geq \frac{\varepsilon}{2} \cdot \frac{2\delta}{\varepsilon} \mu(G) = \delta\mu(G).$$

Since Y is a lower bound on the number of length three augmenting paths for M in $M \oplus S$, we get

$$\mathbf{E}_{F,C,T,j^\star}[\mu(M \cup S)] \geq |M| + \mathbf{E}_{F,C,T,j^\star}[Y] \geq \left(\frac{1}{2} + \delta \right) \mu(G),$$

which is the desired bound. □

4.2.3 Proof of Lemma 4.9

Proof. For brevity, we use $\mathbf{E}'[X]$ as a shorthand for $\mathbf{E}_F[X \mid T, C, j^\star = j]$ throughout the proof.

Recall that in Algorithm 1, when visiting an element $((u, v), \text{EXTEND})$ in T, we add it to S if all the following conditions hold, where we have deliberately broken the last condition in Line 16 of Algorithm 1 into two sub-conditions (C4) and (C5):

(C1) $\deg_S(u) = \deg_S(v) = 0$,

(C2) $\deg_M(v) + \deg_M(u) \leq 1$,

(C3) $c_u \neq c_v$,

(C4) neither of u or v is frozen in Line 13 of Algorithm 1.

(C5) neither of u or v is frozen in Line 15 of Algorithm 1.

Since we have conditioned on T and C, both the matching M and the vertex colors are fully revealed. Thus, every element $((u, v), \text{EXTEND}) \in T$ which upon being visited violates one of the conditions (C2), (C3), or (C4) is fully revealed a priori and can be discarded. On the flip side, condition (C5) depends on F and so remains random.

Now suppose for the sake of the analysis that we also discard all elements (e, EXTEND) where e is incident to an unusual edge in M. We emphasize that discarding these elements might change matching S, but we will show later that this effect is not significant.

Useful definitions: Next, we give a few useful definitions that we use in the proof. First, define

$$\hat{B} := \{v \mid \exists (u, v) \in M_j, c_v = \text{BLUE}, c_u = \text{RED}\}$$

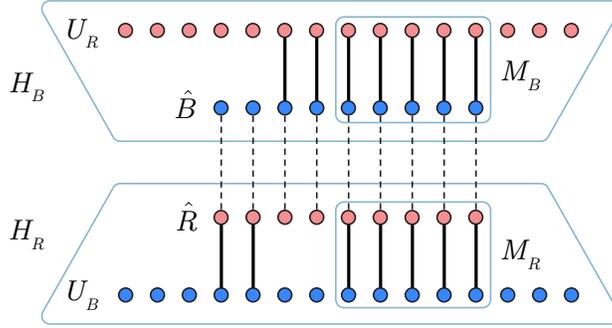$$\hat{R} := \{u \mid \exists (u, v) \in M_j, c_v = \text{BLUE}, c_u = \text{RED}\}.$$

Figure 1: Illustration of $\hat{B}$, $\hat{R}$, U_B, U_R, H_B, H_R, and matchings M_B, M_R. The solid edges are the edges of $M^\star$, and the dashed edges are the edges of M_j whose endpoints have different colors.

Also let U be the set of vertices that are left unmatched by $M_j, \ldots, M_{2/\varepsilon}$, and define

$$U_R := \{u \mid u \in U, c_u = \text{RED}\}, \qquad U_B := \{v \mid v \in U, c_v = \text{BLUE}\}.$$

Moreover, define matchings M_B and M_R as

$$M_B := \{(x, u) \mid \exists (x, u, \cdot, \cdot) \in P_j\}, \qquad M_R := \{(y, v) \mid \exists (\cdot, \cdot, v, y) \in P_j\}.$$

Finally, define H_B (resp. H_R) to be the bipartite graph with vertex parts $\hat{B}$ and U_R (resp. $\hat{R}$ and U_B), including an edge e of G between its vertex parts iff (e, EXTEND) is not discarded.

It can be confirmed from the definitions above that the sets $\hat{B}$, $\hat{R}$, U_R, U_B are all disjoint. This implies that H_B and H_R are vertex disjoint. The next observation also follows immediately from the definitions above.

Observation 4.10. *All edges of M_B (resp. M_R) belong to H_B (resp. H_R).*

See Figure 1 for an illustration of some of these definitions.

We show that H_B and H_R include all the remaining EXTEND elements.

Claim 4.11. *For every element (e, EXTEND) that is not discarded, e belongs to H_B or H_R.*

Proof. Take an undiscarded element $(e = (u, v), \text{EXTEND})$. Take the edge $e' = (u, w) \in M$ with the smallest $\pi_{\text{START}}(e')$ that is incident to e. Note that such e' should exist because M is a maximal matching of G. There are three possible cases, and only the last one does not lead to a contradiction:

- **Case 1** − $\pi_{\text{EXTEND}}(e) < \pi_{\text{START}}(e')$: In this case, e' is unusual by the second condition of Definition 4.2 and so (e, EXTEND) must be discarded, a contradiction.

- **Case 2** − $\pi_{\text{EXTEND}}(e) \geq \pi_{\text{START}}(e')$ and $(e' \notin M_{j^\star}$ or $c_u = c_w)$: In this case, the endpoints of e' must be frozen in Line 13 of Algorithm 1. So condition (C4) does not hold for e when processing (e, EXTEND) and we should discard it, a contradiction.

- **Case 3** − $\pi_{\text{EXTEND}}(e) \geq \pi_{\text{START}}(e')$ and $e' \in M_{j^\star}$ and $c_u \neq c_w$: This is the only case that does not lead to a contradiction.

The condition of Case 3 immediately implies that if $c_u = \text{BLUE}$ then $u \in \hat{B}$ otherwise $u \in \hat{R}$.

Next, we show that v must be unmatched by $M_{j^\star}, \ldots, M_{2/\varepsilon}$, and so $v \in U$. First, note that if v is also matched by M through an edge e'', then by definition of e' it must satisfy $\pi_{\text{START}}(e'') > \pi_{\text{START}}(e')$. Since $e' \in M_{j^\star}$ and the ranks of the edges of $M_{j^\star+1}, \ldots, M_{2/\varepsilon}$ are all smaller than those in $M_{j^\star}$, this implies $e'' \notin M_{j^\star+1}, \ldots, M_{2/\varepsilon}$. So it remains to show $e'' \notin M_{j^\star}$. To see this, note that if $e'' \in M_{j^\star} = M_j$, then from Claim 4.7 part 3, we get $\pi_{\text{START}}(e'') < \pi_{\text{START}}(e') \cdot D$. Now if $\pi_{\text{EXTEND}}(e) > \pi_{\text{START}}(e'')$, then at the time of processing (e, EXTEND) both e' and e'' are in M and we have $\deg_M(u) + \deg_M(v) = 2$, which means (e, EXTEND) violates (C2) and must be discarded, a contradiction. So we should have $\pi_{\text{EXTEND}}(e) < \pi_{\text{START}}(e'') < \pi_{\text{START}}(e') \cdot D$. This is again a contradiction because by the first condition of Definition 4.2, e' must be unusual, and so e must be discarded.

Finally, note that since (e, EXTEND) is not discarded, it should satisfy (C3) and so $c_u \neq c_v$. Combined with the discussion above this means that if $u \in \hat{B}$ then $v \in U_R$ and if $u \in \hat{R}$ then $v \in U_B$. Hence e must belong to one of H_B and H_R, completing the proof. $\qquad\square$

Now consider the construction of S from the remaining undiscarded elements. We iterate over these elements in the order specified by T, and whenever we see an element (e, EXTEND) that satisfies all of (C1)–(C5) we add it to S. As discussed, conditions (C2)–(C4) are automatically satisfied by all undiscarded elements, which only leaves (C1) and (C5). Condition (C1) is simply the greedy matching constraint. Condition (C5) depends on the randomization in F. An edge in H_B (resp. H_R) satifies (C5) if its endpoint in $\hat{B}$ (resp. $\hat{R}$) is not frozen. An important observation is that each vertex in $\hat{B}$ (resp. $\hat{R}$) is frozen independently from the other vertices of $\hat{B}$ (resp. $\hat{R}$) with probability $1 - p$. (We emphasize though that these decisions are not mutually independent when we consider the vertices of both $\hat{B}$ and $\hat{R}$ together.)

Let S_B and S_R be the subset of edges of S that respectively belong to H_B and H_R. Our goal is to apply Proposition 3.2 on both H_B and H_R.

First, we apply Proposition 3.2 by letting $G = H_B$, $V = \hat{B}$, $U = U_R$, the subsample W being the subset of vertices in $\hat{B}$ that are not frozen, and $M = M_B$ (recalling from Observation 4.10 that M_B is completely inside H_B). Using X_B to denote the set of vertices in $V(M_B) \cap \hat{B}$ that get matched in S to U_R, we get from Proposition 3.2 that:

$$\mathbf{E}'[|X_B|] \geq p(|M_B| - 2p|\hat{B}|) \geq p|P_j| - 2p^2|M_j|.$$

Next, we apply Proposition 3.2 by letting $G = H_R$, $V = \hat{R}$, $U = U_B$, the subsample W being the subset of vertices in $\hat{R}$ that are not frozen, and $M = M_R$ (recalling from Observation 4.10 that M_R is completely inside H_R). Using X_R to denote the set of vertices in $V(M_R) \cap \hat{R}$ that get matched in S to U_B, we get from Proposition 3.2 that:

$$\mathbf{E}'[|X_R|] \geq p(|M_R| - 2p|\hat{R}|) \geq p|P_j| - 2p^2|M_j|.$$

Adding back the discarded EXTEND **elements:** We now add back the EXTEND elements incident to unusual edges that we discarded earlier. To do so, we iteratively take an arbitrary vertex of an arbitrary unusual edge in M, and add back all the EXTEND elements incident to it that we discarded, and re-compute matching S.

Claim 4.12. *Let S_1 and S_2 be the edges in matching S before and after adding back the discarded edges of a vertex v. At most two vertices can be matched in one of S_1, S_2 bot not the other.*

Proof. Since S_1 and S_2 are both matchings, $S_1 \Delta S_2$ is a collection of paths and cycles. Thus any vertex whose matching-status differs in S_1 and S_2 must be an endpoint of a path in $S_1 \Delta S_2$. Suppose for contradiction that there are more than two such vertices. Then we have more than one path in $S_1 \Delta S_2$. Take the lowest rank edge e in the path that does not include v. It can be confirmed that whether the conditions (C1)–(C5) are satisfied for e remains the same in both S_1 and S_2, so either e belongs to both or neither, contradicting that $e \in S_1 \Delta S_2$. $\square$

Let us now define X'_B and X'_R to be the analogs of X_B and X_R after we add back the discarded edges incident to unusual edges of M. More precisely, let X'_B (resp. X'_R) denote the set of vertices in $V(M_B) \cap \hat{B}$ (resp. $V(M_R) \cap \hat{R}$) that are matched in S to U_R (resp. U_B).

Note that a vertex v may belong to $X_B \setminus X'_B$ for two reasons: either v was matched in S before adding back the discarded edges but then got unmatched after doing so, or v remains matched in S, but to a vertex not in U_R. Claim 4.12 bounds the total number of vertices of the former type by $2 \times 2|A| = 4|A|$. For the latter type, note from Claim 4.11 that any edge of v that is not discarded goes to U_R. So the new match of v after adding the discarded edges must be a discarded edge. But each discarded edge that belongs to S must match one endpoint of one of the $|A|$ unusual edges in M, so the total number of such edges is no more than $2|A|$. Thus, overall

$$\mathbf{E}'[|X'_B|] \geq \mathbf{E}'[|X_B|] - 6|A| \geq p|P_j| - 2p^2|M_j| - 6|A|. \tag{4}$$

Applying the same argument on X'_R gives

$$\mathbf{E}'[|X'_R|] \geq \mathbf{E}'[|X_R|] - 6|A| \geq p|P_j| - 2p^2|M_j| - 6|A|. \tag{5}$$

Now, let UF_j denote the set of M_j edges of P_j that are unfrozen. Each edge $e \in UF_j$ has one endpoint colored BLUE and one that is colored RED by definition of P_j. The set of BLUE (resp. RED) endpoints of UF_j that are matched in S to a vertex in U_R (resp. U_B) is exactly X'_B (resp. X'_R). We have:

$$\mathbf{E}'[\# \text{ of } (u,v) \in UF_j \text{ s.t. } u \in X'_B, v \in X'_R] \geq \mathbf{E}'[|UF_j| - (|UF_j| - |X'_B|) - (|UF_j| - |X'_R|)]$$
$$= \mathbf{E}'[|X'_B|] + \mathbf{E}'[|X'_R|] - \mathbf{E}'[|UF_j|]$$
$$\geq p|P_j| - 4p^2|M_j| - 12|A|. \tag{6}$$

The last inequality follows from (4) and (5), and the fact that $\mathbf{E}[|UF_j|] = p|P_j|$ because P_j has $|P_j|$ edges in M_j and each one is unfrozen with probability p.

To finish the proof, note that if for an edge $(u,v) \in UF_j$ we have $u \in X'_B$ and $v \in X'_R$, then S matches both u and v to vertices that are unmatched by $M_j, \ldots, M_{2/\varepsilon}$. Therefore, only if these vertices are also unmatched by $M_1, \ldots, M_{j-1}$ we have a length three augmenting path. Therefore,

$$\mathbf{E}'[Y] \geq \mathbf{E}'[\# \text{ of } (u,v) \in UF_j \text{ s.t. } u \in X'_B, v \in X'_R] - 2\sum_{i=1}^{j-1} |M_i|$$

$$\geq p|P_j| - 4p^2|M_j| - 12|A| - 2 \times \frac{1}{1000}|M_j| \qquad \text{(By (6) and Claim 4.7.)}$$

$$= p|P_j| - \left(4p^2 + \frac{1}{500}\right)|M_j| - 12|A|.$$

This finishes the proof of Lemma 4.9 $\square$

4.2.4 Bounding Unusual Edges

In this section we prove Lemma 4.3 that $\mathbf{E}\,|A| = o(|M|)$.

We start by proving the following auxiliary claim.

Claim 4.13. *With probability $1 - 1/n^2$, every edge $e \in E$ satisfies $\pi_{\text{START}}(e) \leq 8m \log n$.*

Proof. Let $z := 8m \log n$. We show that with probability $1 - 1/n^2$ every edge $e \in E$ satisfies $\pi_{\text{START}}(e) \leq z$. This statement is trivial if $|T| \leq z$ so assume $|T| > z$. Fix an arbitrary edge $e \in E$. For the event $\pi_{\text{START}}(e) > z$ to happen, all K copies of (e, START) should appear after the first z elements in T. Thus, noting that $|T| = m(K+1)$, $z = 8m \log n$, and $K \geq 1$, we have

$$\Pr[\pi_{\text{START}}(e) \leq z] \geq 1 - \left(1 - \frac{z}{|T|}\right)^K = 1 - \left(1 - \frac{8m \log n}{m(K+1)}\right)^K \geq 1 - e^{-\frac{8K \log n}{K+1}} \geq 1 - n^{-4}.$$

A union bound over all choices of e, which there are less than n^2 many, proves our first claim. $\square$

We are now ready to prove Lemma 4.3.

Proof of Lemma 4.3. We count the number of unusual edges separately based on the two cases in Definition 4.2. Consider the second case. Let A' be the set of (e', EXTEND) elements in T such that at the time of processing (e', EXTEND), both endpoints of e' are unmatched in M. If we show that $\mathbf{E}_\pi\,|A'| = o(|M|)$, since each edge in A' has at most two incident edges in M, the number of unusual edges in the second case of Definition 4.2 will be $o(|M|)$. Consider the following equivalent construction of A'. We iterate over T, processing its elements one by one. If we see an element (e, START) we add e to M and remove all the unprocessed elements (e', X), $X \in \{\text{EXTEND}, \text{START}\}$ from T such that e' shares an endpoint with e. If we see an element (e, EXTEND), we add it to A'.

At any time during this process, the remaining START elements are at least K times more than the EXTEND element since for each EXTEND element (e, EXTEND) that remains in T, all K copies of (e, START) must also remain in T. This means that every time we reveal the next remaining element of T, it is an EXTEND element with probability at most $1/(K+1)$. Furthermore, there are at most $\mu(G)$ steps where we process a START element since each time we add an edge to M and clearly $|M| \leq \mu(G)$. This implies that $\mathbf{E}\,|A'| \leq \frac{\mu(G)}{K}$. Combining $\mu(G) \leq 2|M|$ and $K = \Omega(\log n)$, we have $\mathbf{E}\,|A'| = o(|M|)$.

Now consider the first case in Definition 4.2. Define $\sigma : T \to \mathbb{R}$ as $\sigma((e, \text{START})) := \pi((e, \text{START}))$ and $\sigma((e, \text{EXTEND})) := \pi((e, \text{EXTEND}))/D$. Similar to the proof of the second case, we process the elements in T one by one. However, instead of π, we process them in the increasing order of their σ values. Let A'' be the set of EXTEND elements in T such that at the time of processing (e', EXTEND) both endpoints of e' are unmatched in M. Note that if an edge e is unusual because of the first case of Definition 4.2, there exists an element (e', EXTEND) such that $\pi_{\text{EXTEND}}(e')/D < \pi_{\text{START}}(e)$ and at the time of processing (e', EXTEND), the only incident edge to e' in M is e. Hence, we must process (e', EXTEND) before (e, START) according to new ordering of elements and at the time of processing (e', EXTEND), there is no incident edge to e' in M. Therefore, $|A''|$ is an upperbound for number of unusual edges in first case of Definition 4.2. Similar to the previous case, it suffices to show $\mathbf{E}_\pi\,|A''| = o(|M|)$ since each edge in A'' has at most two incident edges in M. Consider again the following equivalent construction of A'', where we iterate over the elements based on σ one by one. If we see an element (e, START) we add e to M and remove all the unprocessed elements (e', X),

15

$X \in \{\text{EXTEND}, \text{START}\}$ such that e' shares an endpoint with e. If we see an element (e, EXTEND), we add it to A''.

Let E_S be the event that $\pi_{\text{START}}(e) \leq 8m \log n$ for all edges $e \in E$, and let $\bar{E}_S$ be the complement event. Noting from Claim 4.13 that $\Pr[\bar{E}_S] \leq 1/n^2$. We get

$$
\begin{aligned}
\Pr[\sigma((e, \text{EXTEND})) \leq \pi_{\text{START}}(e)] &= \Pr\left[\pi_{\text{EXTEND}}(e) \leq D \cdot \pi_{\text{START}}(e)\right] \\
&\leq \Pr[\bar{E}_S] + \Pr\left[\pi_{\text{EXTEND}}(e) \leq D \cdot \pi_{\text{START}}(e) \mid E_S\right] \\
&\leq \frac{1}{n^2} + \frac{8mD \log n}{m(K+1)} \\
&\leq \frac{8D \log n}{K}.
\end{aligned}
$$

Hence, the probability of the element that we are processing to be an EXTEND element is at most $\frac{8D \log n}{K}$. Similar to the former case, since there are at most $\mu(G)$ steps in the process that an START element is added to M, we have $\mathbf{E}\,|A''| \leq \frac{\mu(G) \cdot 16D \log n}{K}$. Combining with $\mu(G) \leq 2|M|$, $D = (c \cdot \Delta \cdot \log n)^\varepsilon$, and $K = 10D \log^2 n$, implies $\mathbf{E}\,|A''| = o(|M|)$ which completes the proof. $\qquad\square$

5 A Local Query Algorithm and its Complexity

In this section, we present a local query process that determines whether a given vertex has any edge in matchings M and S of Algorithm 1 without constructing the whole output of Algorithm 1.

While all the randomizations in Algorithm 1 were crucial for the approximation guarantee of Section 4.2 to hold, the bounds of this section only rely on the random shuffling of the sequence T. In particular, the result of this section continues to hold even if all the other coin flips of Algorithm 1 besides the order of T are picked adversarially.

To help the discussion above simplify the presentation of this section, we regard the index $j^\star$ and the color c_v of any vertex v as given. Alternatively, one could pick the color c_v of any vertex uniformly at random whenever we access c_v.

Note from Algorithm 1 that whether a vertex is frozen depends on its edge in M, if any. In particular, a vertex v is frozen iff it is matched in M and its match is also frozen. This can essentially be seen as freezing the edges of M instead of the vertices, which will be the more convenient view for our purpose in this section. More generally, instead of just the START elements that end up in M, we define (Definition 5.1) whether a START element of T is frozen or not. This way, we say a vertex v is frozen iff there is an edge e incident to v such that $e \in M$ and the corresponding (e, START) element in T that adds e to M is frozen.

Definition 5.1. *We say an element* $\ell = ((u, v), \text{START}) \in T$ *is* frozen *if either of the following conditions holds:*

- $c_u = c_v$,
- $\pi(\ell) \leq \alpha_{j^\star+1}|T|$ *or* $\pi(\ell) > \alpha_{j^\star}|T|$,
- *The corresponding* Bernoulli$(1 - p)$ *variable in Line 15 of Algorithm 1 is one.*

We emphasize again that the randomization in the last bullet of Definition 5.1 is only needed for the approximation guarantee, and can be regarded as (adversarially) fixed for our purpose in this section.

Now let π be a permutation of the sequence T consisting of edge copies in graph G. We write $\mathrm{MS}(G, \pi)$ to denote the subgraph $M \cup S$ constructed by Algorithm 1 when processing T in the order of π. The argument π in $\mathrm{MS}(G, \pi)$ is meant to emphasize that once we feed π into Algorithm 1, its output is uniquely determined as we have fixed the other sources of randomization.

Having discussed our basic analysis setup, our local query process for determining whether a given vertex v is matched in M or S is formalized below as Algorithm 2. The algorithm calls two other edge subroutines formalized as Algorithms 3 and 4. Subroutine Algorithm 3 determines if a START element is matched in M by recursively calling the subroutine for incident START elements with a lower rank. Similarly, subroutine Algorithm 4 determines if an EXTEND element is matched in S by recursively calling the subroutines for incident START and EXTEND elements with lower ranks.

Let $F(v, \pi)$ denote the total number of recursive calls to the edge oracles of Algorithms 3 and 4 during the execution of $\mathrm{VO}(v, \pi)$. The following theorem is the main result of this section.

Theorem 5.2. *For a randomly chosen vertex v and a random permutation π of T,*

$$\mathbf{E}_{v, \pi}[F(v, \pi)] = O(K\bar{d} \cdot \log^4 n),$$

where $\bar{d}$ is the average degree of G.

5.1 Correctness of the Oracles

In this section, we prove the correctness of the vertex oracle. Namely, we prove that:

Claim 5.3. *Let $v \in V$ and ST, EX be the outputs of $\mathrm{VO}(v, \pi)$. It holds:*

- $ST = \mathrm{TRUE}$ *iff v has an incident START element in $\mathrm{MS}(G, \pi)$.*

- $EX = \mathrm{TRUE}$ *iff v has an incident EXTEND element in $\mathrm{MS}(G, \pi)$.*

Let us first prove two auxiliary claims about the correctness of the two edge oracles.

Claim 5.4. *For any $\ell = ((u, w), \mathrm{START})$, if $\mathrm{EOS}(\ell, u, \pi)$ is called during computing $\mathrm{VO}(v, \pi)$, then $\mathrm{EOS}(\ell, u, \pi) = \mathrm{TRUE}$ iff $\ell \in \mathrm{MS}(G, \pi)$.*

Proof. We prove Claim 5.4 by induction on $\pi(\ell)$. Suppose that the statement holds for all START elements with a ranking lower than $\pi(\ell)$. If $\mathrm{VO}(v, \pi)$ directly calls $\mathrm{EOS}(\ell, u, \pi)$, it had already called $\mathrm{EOS}(\ell'', u, \pi)$ for every START elements ℓ'' incident on u with a lower ranking. Moreover, if $\mathrm{EOS}(\ell, u, \pi)$ is called by $\mathrm{EOS}(\ell', w, \pi)$ or $\mathrm{EOE}(\ell', w, \pi, \cdot)$, then all START elements on edges (w, u') with a smaller rank must be queried before ℓ by the description of oracles. All these calls to the edge oracle, EOS, must return FALSE since the edge oracle queries $\mathrm{EOS}(\ell, u, \pi)$. By the induction hypothesis, there is no START element incident to w with a smaller rank in $\mathrm{MS}(G, \pi)$. Also, $\mathrm{EOS}(\ell, u, \pi)$ queries all the START elements incident to u with lower rank and returns TRUE if none of them is in $\mathrm{MS}(G, \pi)$. By the induction hypothesis, these calls are answered correctly, completing the proof. $\qquad\square$

Claim 5.5. *Let $\ell = ((u, w), \mathrm{EXTEND})$, and let ST_w indicate if w has a START element incident to it in $\mathrm{MS}(G, \pi)$ with a rank smaller than $\pi(\ell)$. If $\mathrm{EOE}(\ell, u, \pi, ST_w)$ is called during computing $\mathrm{VO}(v, \pi)$, then $\mathrm{EOE}(\ell, u, \pi, ST_w) = \mathrm{TRUE}$ iff $\ell \in \mathrm{MS}(G, \pi)$.*

17

Algorithm 2: The vertex oracle $\mathrm{VO}(u, \pi)$ which determines the matching-status of u in the outputs M and S according to permutation π.

---

1 Let $\ell_1 = (e_1, X_1), \ldots, \ell_r = (e_r, X_r)$ be the elements in T such that $e_i = (u, v_i)$, and $\pi(\ell_1) < \ldots < \pi(\ell_r)$.

2 $ST \leftarrow$ FALSE, $EX \leftarrow$ FALSE.

3 for i *in* $1 \ldots r$ **do**

4   **if** $X_i =$ EXTEND and $c_u = c_{v_i}$ **then continue**.

5   **if** $ST =$ FALSE *and* $X_i =$ START *and* $\mathrm{EOS}(\ell_i, v_i, \pi) =$ TRUE **then**

6 $ST \leftarrow$ TRUE

7   **if** $EX =$ FALSE *and* $X_i =$ EXTEND *and* $\mathrm{EOE}(\ell_i, v_i, \pi, ST) =$ TRUE **then**

8 $EX \leftarrow$ TRUE

9 return ST, EX.

Algorithm 3: The edge oracle $\mathrm{EOS}(\ell = (e, \mathrm{START}), u, \pi)$ for START elements. Here u in the input must be an endpoint of e.

---

1 if $\mathrm{EOS}((e, \mathrm{START}), u, \pi)$ is already computed **then return** the computed answer.

2 Let $\ell_1 = (e_1, X_1), \ldots, \ell_r = (e_r, X_r)$ be the elements in T such that $e_i = (u, v_i)$, $X_i =$ START for all i, and $\pi(\ell_1) < \ldots < \pi(\ell_r) < \pi(\ell)$.

3 for i *in* $1 \ldots r$ **do**

4   **if** $\mathrm{EOS}(\ell_i, v_i, \pi) =$ TRUE **then return** FALSE.

5 return TRUE.

Algorithm 4: The edge oracle $\mathrm{EOE}(\ell = (e, \mathrm{EXTEND}), u, \pi, ST_w)$ for EXTEND elements. Here $e = (u, w)$ and ST_w indicates whether vertex w is matched by M in $\mathrm{MS}(G, \pi)$ through an element of rank smaller than $\pi(\ell)$.

---

1 if $\mathrm{EOE}((e, \mathrm{EXTEND}), u, \pi, ST_w)$ is already computed **then return** the computed answer.

2 Let $\ell_1 = (e_1, X_1), \ldots, \ell_r = (e_r, X_r)$ be the elements in T such that $e_i = (u, v_i)$, and $\pi(\ell_1) < \ldots < \pi(\ell_r) < \pi(\ell)$.

3 $ST_u \leftarrow$ FALSE

4 for i *in* $1 \ldots r$ **do**

5   **if** $X_i =$ EXTEND and $c_u = c_{v_i}$ **then continue**.

6   **if** $ST_u =$ FALSE, $X_i =$ START, *and* $\mathrm{EOS}(\ell_i, v_i, \pi) =$ TRUE **then**

7 $ST_u \leftarrow$ TRUE

8   **if** ℓ_i is frozen **then return** FALSE.

9   **if** $ST_w =$ TRUE **then return** FALSE.

10   **if** $X_i =$ EXTEND and $\mathrm{EOE}(l_i, v_i, \pi, ST_u) =$ TRUE **then return** FALSE.

11 return TRUE.

Proof. We prove the statement by induction on $\pi(\ell)$. With a similar argument as in the proof of Claim 5.4, we can show that EXTEND elements incident to w with lower ranks are queried before ℓ, and that the results of all these calls are FALSE. Also, if there exists a START element incident to w with a lower rank in $\mathrm{MS}(G, \pi)$, then by the description of EOE, it is queried before and ST_w is TRUE. Note that if such an edge exists, it must not be frozen, otherwise, the oracle does not query $\mathrm{EOE}(\ell, u, \pi, ST_w)$. Hence, if $\ell \notin \mathrm{MS}(G, \pi)$, there must be a frozen or an EXTEND element incident to u, or $ST_w = $ TRUE and there exists a START element incident to u. Since $\mathrm{EOE}(\ell, u, \pi, ST_w)$ queries all the edges incident to u with lower rank to determines if a START or EXTEND element exists in $\mathrm{MS}(G, \pi)$, the proof is complete by induction hypothesis. $\square$

We are now ready to prove Claim 5.3.

Proof of Claim 5.3. Since the vertex oracle queries the edge oracle EOS for all incident START elements in increasing order of their ranking until it finds a START element in $\mathrm{MS}(G, \pi)$, by Claim 5.4, the first property holds. Since Algorithm 2 and Algorithm 4 query incident edges in increasing order, if $\mathrm{EOE}(\ell, u, \pi, ST_w)$ is queried for an EXTEND element $\ell = ((u, w), \mathrm{EXTEND})$, ST_w is computed correctly before calling $\mathrm{EOE}(\ell, u, \pi, ST_w)$ which implies that the condition in Claim 5.5 holds. Therefore, with the similar argument for EXTEND elements and correctness of edge oracle EOE by Claim 5.5, the second property holds. $\square$

5.2 Query Complexity of the Oracles

In this section, we prove Theorem 5.2. Consider $\ell = (e, X)$, an element in T. If $X = $ START, we write $Q(\ell, v, \pi)$ to denote the total number of recursive calls to $\mathrm{EOS}(\ell, \cdot, \pi, \cdot)$ during the execution of $\mathrm{VO}(v, \pi)$. If $X = $ EXTEND, we write $Q(\ell, v, \pi)$ to denote the total number of recursive calls to $\mathrm{EOE}(\ell, \cdot, \pi, \cdot)$ during the execution of $\mathrm{VO}(v, \pi)$.

Observation 5.6. *For a permutation π, at most one* START *copy of every edge is queried in recursive calls of* EOS.

Proof. For a fixed edge, the edge oracle on its START copies only generates a new recursive call on the first call because of the caching in Line 1 of Algorithm 3. $\square$

Observation 5.7. *For every element $\ell = (e, X)$ in T and permutation π, $Q(\ell, \pi) \le O(n^2)$.*

Proof. Let $e = \{x, y\}$. For a fixed vertex u, either the vertex oracle $\mathrm{VO}(u, \pi)$ queries the edge oracle for ℓ directly, or through some incident elemenet ℓ'. Note that by Observation 5.6, for each edge e' incident to e, at most one of its START copies generates a new recursive call. Hence, the edge oracle of ℓ is called through at most $2(\deg(x) - 1) + 2(\deg(y) - 1)$ of its incident edges (one of its START copy and one EXTEND element) which implies that there are at most $4(n-1) + 1$ calls during the course of $\mathrm{VO}(u, \pi)$. In other words, we have that $Q(\ell, u, \pi) \le 4n - 3$. Therefore,

$$Q(\ell, \pi) \le \sum_{u \in V} Q(\ell, u, \pi) \le n(4n - 3) \le O(n^2). \quad \square$$

The main contribution of this section is to show that the expected $Q(\ell, \pi)$ for a random permutation π is $O(\log^4 n)$ which is formalized in the following lemma.

Lemma 5.8. *For any element $\ell \in T$, we have $\mathbf{E}_\pi[Q(\ell, \pi)] = O(\log^4 n)$.*

19

Assuming the correctness of Lemma 5.8, we can complete the proof of Theorem 5.2.

Proof of Theorem 5.2.

$$\mathbf{E}_{v,\pi}[F(v,\pi)] = \frac{1}{n}\mathbf{E}_\pi\left[\sum_{v\in V}F(v,\pi)\right] = \frac{1}{n}\mathbf{E}_\pi\left[\sum_{v\in V}\sum_{\ell\in T}Q(\ell,v,\pi)\right]$$

$$= \frac{1}{n}\mathbf{E}_\pi\left[\sum_{\ell\in T}\sum_{v\in V}Q(\ell,v,\pi)\right] = \frac{1}{n}\mathbf{E}_\pi\left[\sum_{\ell\in T}Q(\ell,\pi)\right]$$

$$= \frac{1}{n}\sum_{\ell\in T}\mathbf{E}_\pi[Q(\ell,\pi)] = \frac{1}{n}\sum_{\ell\in T}O(\log^4 n)$$

$$= \frac{1}{n}O(Km\cdot\log^4 n) = O(K\bar{d}\cdot\log^4 n). \qquad \square$$

During the recursive calls to the edge oracles that start from $\mathrm{VO}(v,\pi)$, edges corresponding to the elements in the stack of recursive calls create a path. Because, we query $\mathrm{VO}(v,\pi)$ at the beginning, one of the endpoints of this path is vertex v. We direct the elements in the path away from v towards the other endpoint. We call such a directed path starting from the bottom of the stack all the way to the top, a (v,π)-query-path. For an edge $e = (x,y)$, let $\vec{e}$ denote the directed edge from x to y and $\overleftarrow{e}$ denote a directed edge from y to x. Similarly, for an element $\ell = ((x,y),X)$, $\vec{\ell}$ and $\overleftarrow{\ell}$ represent the direction of e.

Let $Q(\vec{\ell},\pi) \subseteq Q(\ell,\pi)$ be the set of all query paths that end at $\vec{\ell}$ (with the same direction). In the following lemma, we bound the query complexity based on the direction of ℓ. We use this lemma to prove Lemma 5.8.

Lemma 5.9. *For any element $\ell \in T$, we have $\mathbf{E}_\pi[Q(\vec{\ell},\pi)] = O(\log^4 n)$.*

Proof of Lemma 5.8. Since $Q(\ell,\pi) = Q(\vec{\ell},\pi) \cup Q(\overleftarrow{\ell},\pi)$, by Lemma 5.9 we have

$$\mathbf{E}_\pi[Q(\ell,\pi)] \leq \mathbf{E}_\pi[Q(\vec{\ell},\pi)] + \mathbf{E}_\pi[Q(\overleftarrow{\ell},\pi)] = O(\log^4 n) + O(\log^4 n) = O(\log^4 n). \quad \square$$

In the rest of this section, we focus on proving Lemma 5.9. The first helpful observation is that all the EXTEND elements in a (v,π)-query-path appear before the START elements.

Observation 5.10. *Let $\vec{P} = (\vec{\ell_r},\ldots,\vec{\ell_1})$ be a (v,π)-query-path and $\vec{\ell_i} = (\vec{e_i},X_i)$ for all i. Then there exists a j $(0 \leq j \leq r)$ such that $X_i = \mathrm{START}$ for $0 < i \leq j$, and $X_i = \mathrm{EXTEND}$ for $j < i \leq r$.*

Proof. Let j be the maximum index such that $X_j = \mathrm{START}$. If there is no such index, the proof is complete since we can assume that $j = 0$. Now we claim that for all $i < j$, we have $X_i = \mathrm{START}$. This can be verified by noting that Algorithm 3 only queries the edge oracle for START elements. $\square$

Given a permutation π and a path of elements $\vec{P} = (\vec{\ell_r},\ldots,\vec{\ell_1})$, we define $\phi(\pi,\vec{P})$ to be another permutation σ over edges such that:

$$(\sigma(\ell_1),\sigma(\ell_2),\ldots,\sigma(\ell_{r-1}),\sigma(\ell_r)) := (\pi(\ell_2),\pi(\ell_3),\ldots,\pi(\ell_r),\pi(\ell_1))$$
$$\pi(\ell') = \sigma(\ell') \qquad \forall \ell' \notin \vec{P}.$$

Given an element $\vec{\ell}$, by using the above mapping function we can construct a bipartite graph H with two parts A and B such that each part has $((K+1)m)!$ vertices showing different permutations of elements. For a permutation $\pi \in A$ and a (v, π)-query-path $\vec{P}$ that ends at ℓ for some arbitrary vertex v, we connect π in A to $\phi(\pi, \vec{P})$ in B. Note that by the construction of H, $\deg(\pi_A) = Q(\vec{\ell}, \pi_A)$ for all $\pi_A \in A$, since we have a unique element for each query-path that ends at $\vec{\ell}$ with permutation π_A. Hence, in order to prove Lemma 5.8, it is sufficient to prove that $\mathbf{E}_{\pi_A \sim A}[\deg_H(\pi_A)] = O(\log^4 n)$.

Let $\mathcal{Q}(\vec{\ell}, \pi)$ be the set of all query-paths for permutation π that ends at $\vec{\ell}$. Let $\beta = c \log^2 n$ for some large constant c and Π be the set of all possible $((K+1)m)!$ permutations. We partition Π into two subsets of *likely* and *unlikely* permutations, denoted by L and U, respectively, as follows:

$$L := \left\{ \pi \in \Pi \ \Big| \ \max_{\vec{P} \in \mathcal{Q}(\vec{\ell}, \pi)} |\vec{P}| \leq \beta \right\} \qquad U := \Pi \setminus L.$$

In words, likely permutations are those where the longest query-path ending at $\vec{\ell}$ has a length of at most β and unlikely permutations are the remaining ones. Let A_L be the set of vertices corresponding to the likely permutations in A and A_U be the set of vertices corresponding to unlikely permutations. The intuition behind this partitioning is that the set of unlikely permutations makes up a tiny fraction of all permutations which is formalized in Lemma 5.11.

Lemma 5.11. *If c is a large enough constant, then we have $|A_U| \leq ((K+1)m)!/n^2$.*

Furthermore, we show that each vertex $\pi_B \in B$ has at most $O(\beta^2)$ neighbors among the likely permutations in part A of our bipartite graph H.

Lemma 5.12. *Let π_B be a vertex in B. Then π_B has at most β^2 neighbors in A_L.*

Proof of Lemma 5.9. We provide an upper bound on $|E(H)|$. First, note that by Lemma 5.11, we have $|A_U| \leq ((K+1)m)!/n^2$. Furthermore, by Observation 5.7, degree of each vertex $\pi_A \in A$ is at most $O(n^2)$ which implies that the total number of edges incident to vertices of A_U is at most

$$E(A_U, B) \leq ((K+1)m)!/n^2 \cdot O(n^2) \leq O(((K+1)m)!).$$

Moreover, by Lemma 5.12, each vertex $\pi_B \in B$ has at most $O(\beta^2)$ neighbors in A_L. Since H is a bipartite graph, total number of incident edges to A_L is at most $O(\beta^2) \cdot |A_L|$. Therefore, sum of degrees of all vertices in A is at most

$$O(\beta^2) \cdot |A_L| + E(A_U, B) \leq O(\beta^2 \cdot ((K+1)m)!).$$

For a random vertex in A, the expected degree is $\frac{O(\beta^2 \cdot ((K+1)m)!)}{((K+1)m)!} = O(\beta^2)$, which completes the proof since $\beta = c \log^2 n$ and $\deg(\pi_A) = Q(\vec{\ell}, \pi_A)$. $\qquad \square$

In the following two subsections, we prove Lemmas 5.11 and 5.12.

5.2.1 Proof of Lemma 5.12

This proof is the most technical part of this part of the analysis. We show that for two query-paths $\vec{P}$ and $\vec{P'}$ that end at $\vec{\ell}$, and two permutations π and π', if $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P'})$, then either one of

the query paths is a subpath of the other one, or they "branch" through a specific configuration of START and EXTEND elements illustrated in Figure 2.

To formalize this, let us formally define what a "branch" is:

Definition 5.13. *Let $\vec{P} = (\vec{\ell}_{r_1}, \ldots, \vec{\ell}_1)$ and $\vec{P}' = (\vec{\ell}'_{r_2}, \ldots, \vec{\ell}'_1)$ be two query-paths. Let j be an such that $\vec{\ell}_i = \vec{\ell}'_i$ for all $i \leq j < \min(r_1, r_2)$, but $\vec{\ell}_{j+1} \neq \vec{\ell}'_{j+1}$. Then $\vec{P}$ and $\vec{P}'$ branch at element $\vec{\ell}_j$.*

And now let us define *valid* and *invalid* branches (see Figure 2).

Definition 5.14. *Let $\vec{P} = (\vec{\ell}_{r_1}, \ldots, \vec{\ell}_1)$ and $\vec{P}' = (\vec{\ell}'_{r_2}, \ldots, \vec{\ell}'_1)$ be two query-paths, where $\vec{\ell}_i = (\vec{e}_i, X_i)$, and $\vec{\ell}'_i = (\vec{e}'_i, X'_i)$. Assume that $\vec{P}$ and $\vec{P}'$ branch at element $\vec{\ell}_j$ for $j < \min(r_1, r_2)$. Moreover, let u be the shared vertex between $\vec{e}_j$ and $\vec{e}_{j+1}$. We call this branch valid if $X_j = \text{START}$ and $\{X_{j+1}, X'_{j+1}\} = \{\text{EXTEND}, \text{START}\}$. For all other possible configurations of X_j, X_{j+1}, and X'_{j+1}, we say the branch is invalid.*
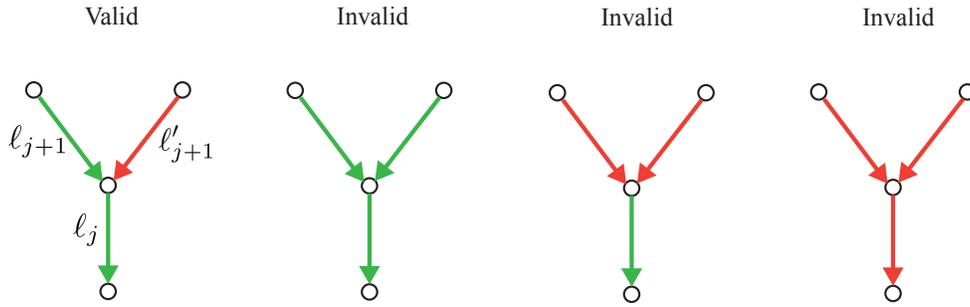

Figure 2: All valid and invalid branches. Green edges represent START elements and red edges represent EXTEND elements.

One key property of invalid branches is formalized in the following observation which is helpful in the rest of this section.

Observation 5.15. *Let $\vec{P} = (\vec{\ell}_{r_1}, \ldots, \vec{\ell}_1)$ and $\vec{P}' = (\vec{\ell}'_{r_2}, \ldots, \vec{\ell}'_1)$ be two query-paths, $\vec{\ell}_i = (\vec{e}_i, X_i)$ and $\vec{\ell}'_i = (\vec{e}'_i, X'_i)$ for all i. If there exists an invalid branch at $\vec{\ell}_j$, then none of the tuples (X_j, X_{j+1}), (X_j, X'_{j+1}), (X_{j+1}, X'_{j+1}), and (X'_{j+1}, X_{j+1}) is $(\text{EXTEND}, \text{START})$.*

Proof. Let u be the shared endpoint between $\vec{e}_j$, $\vec{e}_{j+1}$, and $\vec{e}'_{j+1}$. If $X_j = \text{EXTEND}$, by Observation 5.10, we have $X_{j+1} = X'_{j+1} = \text{EXTEND}$. Therefore, all tuples are $(\text{EXTEND}, \text{EXTEND})$.

Now assume that $X_j = \text{START}$. If one of X_{j+1} or X'_{j+1} is EXTEND and the other one is START, then the branch is valid by Definition 5.14. This leaves two remaining scenarios:

- $X_{j+1} = X'_{j+1} = \text{START}$: In this case, all tuples are $(\text{START}, \text{START})$.

- $X_{j+1} = X'_{j+1} = \text{EXTEND}$: In this case, $(X_{j+1}, X'_{j+1}) = (X'_{j+1}, X_{j+1}) = (\text{EXTEND}, \text{EXTEND})$. Moreover, $(X_j, X_{j+1}) = (X_j, X'_{j+1}) = (\text{START}, \text{EXTEND})$. Thus there is no $(\text{EXTEND}, \text{START})$ among the tuples considered in the statement.

The proof is thus complete. $\qquad\square$

Next, we show that for two query-paths $\vec{P}$ and $\vec{P}'$ that end at $\vec{\ell}$, and two permutations π and π', if $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}')$, then either the shorter query-path is a subpath of the longer one or they have exactly one valid branch. In particular, it is not possible for $\vec{P}$ and $\vec{P}'$ to have an invalid branch or have multiple valid branches under this condition.

Lemma 5.16. *Let π and π' be two different permutations over T, and let $\vec{P}$ and $\vec{P}'$ be (v, π)- and (v', π')-query-path, respectively, that both end at element $\vec{\ell}$. If $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}')$, then $\vec{P}$ and $\vec{P}'$ branch at most once and this branch is valid.*

Before proving Lemma 5.16, we prove a sequence of auxiliary observations and claims. Let $\vec{P} = (\vec{\ell}_{r_1}, \ldots, \vec{\ell}_1)$ and $\vec{P}' = (\vec{\ell}'_{r_2}, \ldots, \vec{\ell}'_1)$ where $\vec{\ell} = \vec{\ell}_1 = \vec{\ell}'_1$. Also, let $\vec{\ell}_i = (\vec{e}_i, X_i)$ and $\vec{\ell}'_i = (\vec{e}'_i, X'_i)$. For the sake of contradiction, suppose that $\vec{P}$ and $\vec{P}'$ branch at element $\vec{\ell}_b$ and the branch is invalid. This means that $\vec{\ell}_i = \vec{\ell}'_i$ for $i \le b$ and $\vec{\ell}_{b+1} \ne \vec{\ell}'_{b+1}$. Note that $\vec{e}_{b+1}$ and $\vec{e}'_{b+1}$ can be the same edge. We do not need to separately investigate these two cases as our proof generally works for both cases.

Observation 5.17. *We have that $\pi(\ell_1) < \pi(\ell_2) < \ldots < \pi(\ell_{r_1})$ and $\pi'(\ell'_1) < \pi'(\ell'_2) < \ldots < \pi'(\ell'_{r_2})$.*

Proof. Algorithms 3 and 4 recursively call on elements with π values less than π value of the current element. Therefore, the stack of recursive calls will be decreasing with respect to π values. The same condition also holds for permutation π'. $\qquad\square$

Observation 5.18. $\pi(\ell_b) = \pi'(\ell_{b+1})$.

Proof. Since $\vec{\ell}_{b+1}$ is not in $\vec{P}'$, we have that $\phi(\pi', \vec{P}')(\ell_{b+1}) = \pi'(\ell_{b+1})$. Also, $\phi(\pi, \vec{P})(\ell_{b+1}) = \pi(\ell_b)$ since $\phi(\pi, \vec{P})$ shifts the elements of path $\vec{P}$ by one. Given that $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}')$, we get $\pi(\ell_b) = \pi'(\ell_{b+1})$. $\qquad\square$

In the rest of the proof, we assume that $\pi(\ell_b) < \pi'(\ell_b)$. This is without loss of generality because we have not distinguished the two permutations π and π' in any other way.

Observation 5.19. $\pi'(\ell_{b+1}) < \pi'(\ell_b)$.

Proof. By combining Observation 5.18, our assumption that $\pi(\ell_b) < \pi'(\ell_b)$, and the fact that π' is a permutation, we have that $\pi'(\ell_{b+1}) < \pi'(\ell_b)$. $\qquad\square$

Claim 5.20. *If $\pi(f) < \pi(\ell_b)$ or $\pi'(f) < \pi(\ell_b)$ for some element f, then $\pi(f) = \pi'(f)$.*

Proof. There are five different possible cases for f:

- $f \notin P \cup P'$: Since ϕ only changes the rank of elements on the query-path and $\phi(\pi, \vec{P})(f) = \phi(\pi', \vec{P}')(f)$, we have $\pi(f) = \pi'(f)$.

- $f \in \{\ell_1, \ldots, \ell_{b-1}\}$: Since $\phi(\pi, \vec{P})(\ell_{i+1}) = \phi(\pi', \vec{P}')(\ell_{i+1})$ for $1 \le i < b$, we have $\pi(\ell_i) = \pi'(\ell_i)$. Hence, $\pi(f) = \pi'(f)$.

- $f = \ell_b$: In this case, condition $\pi(f) < \pi(\ell_b)$ does not hold since $\pi(f) = \pi(\ell_b)$. Also, $\pi'(f) = \pi'(\ell_b) > \pi(\ell_b)$ by our assumption. Therefore, condition $\pi'(f) < \pi(\ell_b)$ does not hold.

- $f \in \{\ell_{b+1}, \ldots, \ell_{r_1}\}$: By Observation 5.17, we have $\pi(f) > \pi(\ell_b)$. Therefore, condition $\pi(f) < \pi(\ell_b)$ does not hold. Let $f = \ell_i$ for $i > b$. Since $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}')$, we have that $\pi'(f) = \pi(\ell_{i-1}) \ge \pi(\ell_b)$. Therefore, none of the conditions in the claim statement hold.

23

- $f \in \{\ell'_{b+1}, \ldots, \ell'_{r_2}\}$: By Observation 5.17, we have $\pi'(f) > \pi'(\ell_b)$. Combined with our assumption $\pi'(\ell_b) > \pi(\ell_b)$, this gives $\pi'(f) > \pi(\ell_b)$. Let $f = \ell'_i$ for $i > b$. Since $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P'})$, we have $\pi(f) = \pi'(\ell_{i-1}) \geq \pi'(\ell_b) > \pi(\ell_b)$. Therefore, none of the conditions in the claim statement hold.

Each of the cases above either contradicts a condition of the claim, or satisfies $\pi(f) = \pi'(f)$. The proof is thus complete. $\qquad\square$

Observation 5.21. *Let f be a* START *element such that $\pi(f) = \pi'(f)$. Then f is frozen in permutation π iff it is frozen in permutation π'.*

Proof. Since we fix the color of vertices, Bernoulli random variables in Line 15 of Algorithm 1, and the index j^* that is used in Algorithm 1 for both permutations π and π' over T, the only way that one of the f and f' is frozen and the other one is not, is when $\pi(f) \neq \pi(f')$. $\qquad\square$

Claim 5.22. $\ell_{b+1} \in \mathrm{MS}(G, \pi')$.

Proof. We prove the claim by contradiction. Assume that $\ell_{b+1} \notin \mathrm{MS}(G, \pi')$. There are two possible scenarios for ℓ_{b+1} not to be in $\mathrm{MS}(G, \pi')$:

- $X_{b+1} =$ EXTEND and $\ell_{b+1} \notin \mathrm{MS}(G, \pi')$ because of two START elements $f, f' \in \mathrm{MS}(G, \pi')$:

  Note that $\pi'(f) < \pi'(\ell_{b+1})$ and $\pi'(f') < \pi'(\ell_{b+1})$ since $f, f' \in \mathrm{MS}(G, \pi')$ and $\ell_{b+1} \notin \mathrm{MS}(G, \pi')$. On the other hand, by Observation 5.18, we have that $\pi(\ell_b) = \pi'(\ell_{b+1})$. Thus, $\pi'(f) < \pi(\ell_b)$ and $\pi'(f') < \pi(\ell_b)$. Hence, by Claim 5.20, $\pi(f) = \pi'(f)$ and $\pi(f') = \pi'(f')$, which implies that $f, f' \in \mathrm{MS}(G, \pi)$ since π and π' are identical for ranks smaller than $\pi(\ell_b)$. Moreover, by Observation 5.21, each of f and f' are either frozen in both π and π' or not. Also, both f and f' are not in path P since $\pi(f) < \pi(\ell_b)$ and $\pi(f') < \pi(\ell_b)$. Let $e_{b+1} = (w, y)$ where y is the shared endpoint with e_b. Without loss of generality, assume that f is incident to w and f' is incident to y. Note that, both of f and f' cannot be incident to the same endpoint of e_{b+1} since START elements create a maximal matching. Since both edge oracle and vertex oracle queries edges in increasing order, when $\mathrm{EOE}(\ell_{b+1}, y, \pi, ST_w)$ was called, ST_w must be TRUE since the edge oracle already queried element f before. Furthermore, $\mathrm{EOE}(\ell_{b+1}, y, \pi, ST_w)$ calls edge oracle for f' before ℓ_b since $\pi'(f) < \pi(\ell_b)$. This implies that (v, π)-query-path $\vec{P}$ is not a valid (v, π)-query-path since the $\mathrm{EOE}(\ell_{b+1}, y, \pi, ST_w)$ terminates after calling the edge oracle for element f' (see Figure 3).

- $\ell_{b+1} \notin \mathrm{MS}(G, \pi')$ because of a single element $f \in \mathrm{MS}(G, \pi')$:

  Let y be the shared endpoint of e_b and e_{b+1}. With the same argument as in the previous case, we get that $f \in \mathrm{MS}(G, \pi)$ and $\pi(f) < \pi(\ell_b)$. Thus, by Observation 5.21, element f is either frozen in both π and π' or in neither one. Note that either both of f and ℓ_{b+1} are START elements, or f is a frozen element, or both of f and ℓ_{b+1} are EXTEND elements. Hence, by Observation 5.10, f must be queried before ℓ_{b+1} if it is not incident to y in the edge oracle for permutation π which implies that the edge oracle will not query ℓ_{b+1}. Furthermore, if f is incident to y, edge oracle queries f before ℓ_b which implies that it terminates and (v, π)-query-path $\vec{P}$ is not a valid (v, π)-query-path. $\qquad\square$
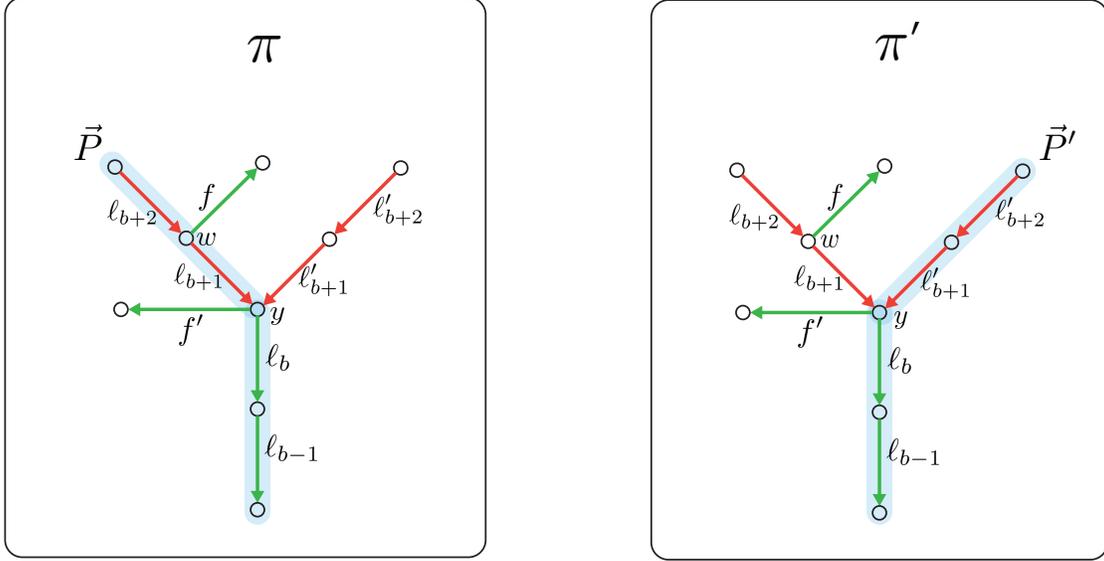
Figure 3: Illustration of a possible case of the first scenario in the proof of Claim 5.22. Green edges represent START elements and red edges represent EXTEND elements. The left figure shows a query-path $\vec{P}$ in permutation π which is highlighted in light blue. Similarly, the right figure shows a query-path $\vec{P'}$ in permutation π'. Query-path $\vec{P}$ is not a valid query-path since the $\text{EOE}(\ell_{b+1}, y, \pi, ST_w)$ terminates after calling the edge oracle on element f'.

Proof of Lemma 5.16. Assume for the sake of contradiction that the branch at $\vec{\ell}_b$ is an invalid branch. We prove that query-path $\vec{P'}$ is not a valid (v, π')-query-path. By Definition 5.14, we have that $X_{b+1} = X'_{b+1}$. Also, by Claim 5.22, edge oracle of ℓ'_{b+1} for permutation π' queries ℓ_{b+1} before ℓ_b since $\pi'(\ell_{b+1}) < \pi'(\ell_b)$ by Observation 5.19. Thus, the edge oracle for ℓ'_{b+1} terminates at this point. Therefore, P and P' contains no invalid branch. □

Now we are ready to complete the proof of Lemma 5.12.

Proof of Lemma 5.12. As above, consider two query paths $\vec{P} = (\vec{\ell}_{r_1}, \ldots, \vec{\ell}_1)$ and $\vec{P'} = (\vec{\ell}_{r_2}, \ldots, \vec{\ell}_1)$ that end at $\vec{\ell} = \vec{\ell}_1 = \vec{\ell}_1$ and suppose that $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P'})$.

If ℓ is an EXTEND element, then all the elements in the two query paths $\vec{P}$ and $\vec{P'}$ must be EXTEND by Observation 5.10. Therefore, $\vec{P}$ and $\vec{P'}$ cannot have a valid branch since a valid branch, by Definition 5.14, requires two START elements. Since Lemma 5.16 asserts $\vec{P}$ and $\vec{P'}$ cannot have invalid branches either, one of P and P' must be a subpath of the other. Note that all paths that end at ℓ must be a subpath of the longest query path since we do not have a branch which implies that all paths must have different lengths. Therefore, there are at most β query-paths that end at $\vec{\ell}$ since the length of the longest path is at most β.

Now suppose that ℓ is a START element. Assume that $\vec{P}$ and $\vec{P'}$ branch at $\vec{\ell}_b$. Since this cannot be an invalid branch by Lemma 5.16, without a loss of generality, assume that ℓ'_{b+1} is an EXTEND element. By Observation 5.10, all $\ell'_{b+1}, \ell'_{b+2}, \ldots, \ell'_{r_2}$ must be EXTEND elements. Let $\mathcal{P}$ be the set of all query-paths that end at $\vec{\ell}$, and $\vec{P}_s \in \mathcal{P}$ be the path with maximum number of START elements (break the tie with the longest length of EXTEND elements of the path). Note that there is no other

START element in the paths of $\mathcal{P}$ other than START elements of P_s. Otherwise, we have an invalid branch since START elements appear before EXTEND elements by Observation 5.10.

Let $(\vec{\ell}_s, \ldots, \vec{\ell}_1)$ be the subpath of $\vec{P}_s$ consisting of START elements. We claim that for each $\vec{\ell}_i$ ($i < s$), if there exist two query-paths in $\mathcal{P}$ that branch with $\vec{P}_s$ at $\vec{\ell}_i$, then one of them must be a subpath of the other. To see this, assume that there are two paths $\vec{P}$ and $\vec{P}'$ such that they both have a branch with $\vec{P}_s$ at $\vec{\ell}_i$. Let $\vec{f}$ and $\vec{f}'$ be the next elements on $\vec{P}$ and $\vec{P}'$. Note that it is possible that $\vec{f} = \vec{f}'$. By Definition 5.14, f and f' are EXTEND elements. Hence, the first elements that are not shared in both $\vec{P}$ and $\vec{P}'$ are EXTEND elements which means $\vec{P}$ and $\vec{P}'$ has an invalid branch.
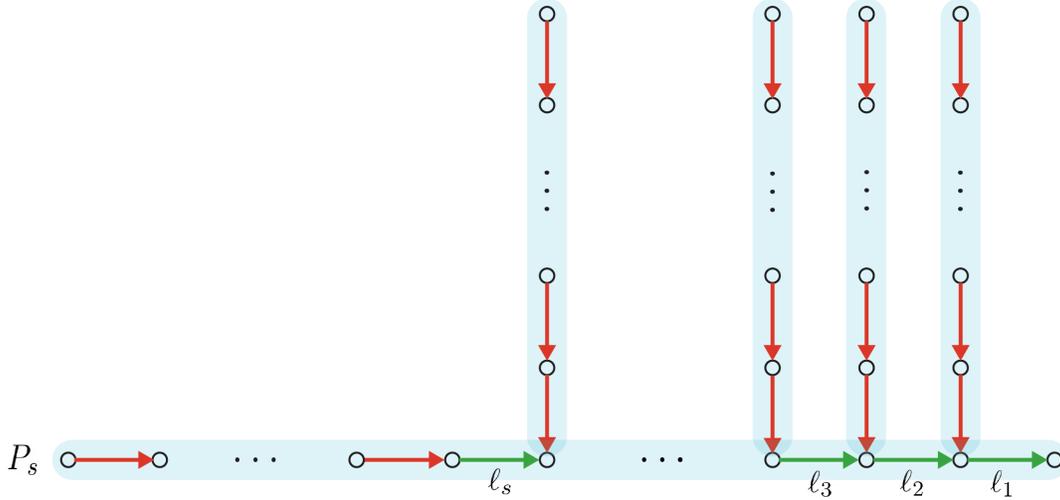

Figure 4: Illustration of $\hat{\mathcal{P}}$ and P_s. The green edges represent START elements and the red edges represent EXTEND elements. Paths in $\hat{\mathcal{P}}$ are highlighted in light blue.

On the other hand, note that there is no path in $\mathcal{P}$ that has a branch with $\vec{P}_s$ in EXTEND elements of $\vec{P}_s$ since there is no invalid branch by Lemma 5.16. Let $\hat{\mathcal{P}}$ be the union of $\vec{P}_s$ and the set of paths in $\mathcal{P}$ such that they have a valid branch with $\vec{P}_s$, and they are not a subgraph of other paths in $\mathcal{P}$ (see Figure 4).

By the above claim, $|\hat{\mathcal{P}}| \leq s + 1$. To complete the proof, assume that for each edge between $\pi_A \in A_L$ and $\pi_B \in B$ in graph H, we write a label $(\vec{P}', |\vec{P}|)$ where $\vec{P}$ is the query-path corresponding to the edge between π_B and π_A, and $\vec{P}' \in \hat{\mathcal{P}}$ such that $\vec{P} \subseteq \vec{P}'$ (if there are multiple choices for $\vec{P}'$, choose the longest path). Since all labels must be different, and by definition of A_L we have that all query-paths have a length of at most β, there are at most $(s+1)\beta \leq 2\beta^2$ different labels, where the last inequality followed by the fact that $|\vec{P}_s| \leq \beta$. □

5.2.2 Proof of Lemma 5.11

In this section, we prove that it is very unlikely to have long query-paths during the recursive calls of edge oracle. Our proof is inspired by [BFS12], who proved that the parallel round complexity of greedy maximal independent set is $O(\log^2 n)$. Our arguments are slightly different because our algorithm is not an instance of greedy MIS.

We define a θ-prefix of permutation π over elements T to be the first $\theta|T|$ elements in permu-

tation π. Suppose that instead of running the edge oracle on the whole permutation of elements, we choose a θ-prefix of elements and run the edge oracle for elements of the prefix. The first useful observation is that if the algorithm calls edge oracle for one of the elements in a θ-prefix of T, all recursive calls during this call will be on elements in the prefix.

Observation 5.23. *Let ℓ be an element in T and π be a permutation over T. Then $\mathrm{EOS}(\ell, \cdot, \pi)$ or $\mathrm{EOE}(\ell, \cdot, \pi, \cdot)$ only recursively calls edge oracle for elements with a lower rank.*

Proof. Proof can be easily obtained by how we defined Algorithm 3 and Algorithm 4 since each element only recursively calls elements with a lower rank. □

The high-level approach for this section is that we partition elements of T to $O(\log n)$ continuous ranges and show that the length of the longest query-path inside each of these ranges is $O(\log n)$. Therefore, since for each element the edge oracle only calls elements with a lower rank, the length of the longest path in total should be at most $O(\log^2 n)$.

Element Partitioning: Let $\mathcal{P}_1$ be a θ_1-prefix of T. Suppose that we run Algorithm 1 on $\mathcal{P}_1$. Let $\mathcal{A}_1$ be the set of elements that are chosen by Algorithm 1 for the selected subgraph $\mathrm{MS}(G, \pi)$. Also, let $\mathcal{D}_1$ be the set of elements outside the prefix $\mathcal{P}_1$ that cannot be in $\mathrm{MS}(G, \pi)$ due to the existence of some elements in $\mathcal{A}_1$. We delete $\mathcal{P}_1 \cup \mathcal{D}_1$ from T and similarly choose θ_2-prefix of the remaining elements. Let $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_d$ be the partitions with parameters $\theta_1, \theta_2, \ldots, \theta_d$, and $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_d$ be sets of deleted elements as defined above. We let $\theta_i = \Omega(2^i \log(n)/(2K\Delta))$ for the i-th partition and $d = O(\log n)$.

Observation 5.24. *Let ℓ be an element in $\mathcal{P}_j \cup \mathcal{D}_j$ as defined above. Then $\mathrm{EOS}(\ell, \cdot, \pi)$ or $\mathrm{EOE}(\ell, \cdot, \pi, \cdot)$ only recursively calls edge oracle for elements in $\{\mathcal{P}_i\}_{i \leq j} \cup \{\mathcal{D}_i\}_{i < j}$.*

Proof. If $\ell \in \mathcal{P}_j$, since all the elements with lower ranks are in $\{\mathcal{P}_i\}_{i \leq j} \cup \{\mathcal{D}_i\}_{i < j}$, by Observation 5.23, the proof is complete. If $\ell \in \mathcal{D}_j$, there are some elements in $\mathcal{P}_j$ that their existence in $\mathrm{MS}(G, \pi)$ caused ℓ not to be in $\mathrm{MS}(G, \pi)$. Hence, since edge oracle recursively calls incident elements in their increasing ranks, edge oracles for ℓ will only query incident elements in $\{\mathcal{P}_i\}_{i \leq j} \cup \{\mathcal{D}_i\}_{i < j}$. □

Note that the START elements that appear in matching M of Algorithm 1 form a randomized greedy maximal matching, and our query process for START elements also coincides with the query process for greedy matchings. Therefore, we can use the following result of [Beh21] which itself builds on the bound on [FN20] as black-box.

Lemma 5.25 ([Beh21, Lemma 3.13]). *Let π be a permutation over elements of T. With high probability, the maximum length of a query-path consisting of START elements is $O(\log n)$.*

In the rest of this section, we show that it is unlikely to have a query-path consisting of EXTEND elements with length larger than $O(\log^2 n)$. Since by Observation 5.10 all START elements appear after EXTEND elements in a query-path, this is enough to show that the length of a query-path is bounded by $O(\log^2 n)$ with high probability.

The following two lemmas are similar to [BFS12, Lemma 3.1 and Lemma 3.3], however, both are adapted to our setting.

Lemma 5.26. *Suppose that we choose a θ-prefix of a uniformly at random permutation π of elements of T, where $\theta = a/b$ for positive numbers $a \leq b \leq |T|$ such that $a/b \geq 2/|T|$. Let*

$\mathrm{MS}_\theta(G, \pi)$ *be the subgraph produced by* [Algorithm 1](#) *on this prefix. Assume that we remove all other elements outside the prefix that cannot join subgraph* $\mathrm{MS}(G, \pi)$ *based on the current elements in* $\mathrm{MS}_\theta(G, \pi)$. *All remaining* Extend *elements have at most* b *incident elements with probability of at least* $1 - \frac{2|T|^2}{e^{a/2}}$.

Proof. We say an element is *live* if we can add it to subgraph $\mathrm{MS}_\theta(G, \pi)$ and *dead* otherwise. Assume that we sequentially pick $(a|T|/b)$ elements. If the chosen element is live, we add the element to $\mathrm{MS}_\theta(G, \pi)$, and mark all incident elements that cannot be in $\mathrm{MS}_\theta(G, \pi)$ after adding this element, as dead. If the chosen element is dead, we do nothing. This sequential algorithm is equivalent to choosing a prefix of a permutation and then processing the permutation.

Let ℓ be an Extend element that is live and has more than b incident live elements after processing the prefix. We show that this event is unlikely. Since at the end of $(a|T|/b)$ rounds, ℓ has more than b incident live elements, before all of $(a|T|/b)$ rounds, it has more than b incident live elements. Hence, in round i of the sequential process, with probability of at least $b/(|T|-i) > b/|T|$, one of the incident live elements of ℓ will be selected. Note that if more than one incident element of ℓ is added to $\mathrm{MS}_\theta(G, \pi)$, then $\ell \notin \mathrm{MS}(G, \pi)$. (It is possible that ℓ is not in $\mathrm{MS}(G, \pi)$ because of one incident element, however, we provide a looser bound by only considering the existence of two incident elements.) Thus, the probability that at most one of its incident elements is selected is at most

$$\left(1 - \frac{b}{|T|}\right)^{\frac{a|T|}{b}} + \left(\frac{a|T|}{b}\right)\left(1 - \frac{b}{|T|}\right)^{\frac{a|T|}{b}-1} < \frac{2a|T|}{b}\left(1 - \frac{b}{|T|}\right)^{\frac{a|T|}{2b}} = \frac{2a|T|}{b}\left(1 - \frac{b}{|T|}\right)^{\frac{|T|}{b} \cdot \frac{a}{2}}, \quad (7)$$

where the first term of left-hand side is the probability that none of the incident elements is chosen and the second term is an upper bound for the probability that exactly one of the incident elements is selected. Combining equations (7), $(1 - \frac{b}{|T|})^{\frac{|T|}{b}} < (1/e)$, and $\frac{a}{b} < 1$, the probability of this event is at most $\frac{2|T|}{e^{a/2}}$. Taking a union bound over all elements completes the proof. $\square$

Corollary 5.27. *If* $\theta_i = \Omega(2^i \log n/(2K\Delta))$, *then all remaining elements have at most* $2K\Delta/2^i$ *incident elements after round* i.

Proof. At the beginning, each element is incident to at most $2K\Delta$ elements since the degree of each vertex is at most Δ and we create K copies of each edge. Since $\log |T| = O(\log n)$, the proof can be obtained by [Lemma 5.26](#) with $a = \Omega(\log n)$ and $b = 2K\Delta/2^i$ for partition i. $\square$

In the previous lemma, by choosing the appropriate θ_i, the number of incident elements for each Extend element reduce by half after removing partition i. Hence, there will be at most $O(\log n)$ partitions. In the next lemma, we will show that the length of the longest query-path for Extend elements in each of $\mathcal{P}_i$ is bounded by $O(\log n)$.

Lemma 5.28. *Suppose that all* Extend *elements have at most* x *incident elements. Let* a *and* b *be two positive integers such that* $a \geq b$ *and consider a randomly ordered* θ-*prefix from* T *with* $\theta < b/x$. *Then the longest query-path consist of* Extend *elements has length* $O(a)$ *with probability of at least* $1 - |T|(b/a)^a$.

Proof. Let $(i_1, i_2, \ldots, i_{k+1})$ be $k+1$ different indices in the θ-prefix. Choosing the prefix elements sequentially is equivalent to choosing a randomly ordered prefix. Let $(\ell_{i_1}, \ell_{i_2}, \ldots, \ell_{i_{k+1}})$ be elements in these indices that create a query-path. Hence, the probability that ℓ_{i_1} and ℓ_{i_2} are incident is at

most $x/(|T|-1)$ since when we select ℓ_{i_2}, element ℓ_{i_1} is already chosen and there are $|T|-1$ choices remaining and ℓ_1 has at most x incident elements. With the same argument, the probability that ℓ_2 and ℓ_3 be incident is at most $x/(|T|-2)$. Hence, the probability of having such a path is at most $(x/(|T|-k))^k$. Taking a union bound over all possible $k+1$ indices of the prefix, we get the following bound for having a query-path of length $k+1$. By assuming that $k < |T|/2$, we have

$$
\begin{aligned}
\binom{\theta|T|}{k+1} \cdot (x/(|T|-k))^k &\leq \left(\frac{e\theta|T|}{k+1}\right)^{k+1} \cdot \left(\frac{x}{|T|-k}\right)^k \\
&= \frac{e\theta|T|}{k+1} \cdot \left(\frac{ex\theta|T|}{(k+1)(|T|-k)}\right)^k \\
&\leq \frac{e\theta|T|}{k+1} \cdot \left(\frac{2ex\theta}{k+1}\right)^k \qquad\qquad (k \leq |T|/2) \\
&\leq |T|\left(\frac{2ex\theta}{k+1}\right)^{k+1}.
\end{aligned}
$$

By setting $k = 2ea - 1$ and $\theta < b/x$ the above bound will be at most $|T|(b/a)^a$. Therefore, with probability $1 - |T|(b/a)^a$, the longest query-path has length $O(a)$. Note that if $k \geq |T|/2$, it implies that $2ea \geq |T|/2$ which the lemma clearly holds since $a = O(|T|)$. $\qquad\square$

Corollary 5.29. *Suppose that all* EXTEND *elements have at most x incident elements. The longest query-path consisting of* EXTEND *elements, has length of at most $O(\log n)$ with probability $1 - \frac{1}{n^4}$ for a $O(\log(n)/x)$-prefix of T.*

Proof. Setting $a = 5b = O(\log|T|) = O(\log n)$ in Lemma 5.28 completes the proof. $\qquad\square$

Now we are ready to complete the proof of Lemma 5.11.

Proof of Lemma 5.11. First, if the edge oracle recursively calls a START element, then we get from Lemma 5.25 and Observation 5.10 that the remaining length of the query-path will not exceed $O(\log n)$ with high probability. Therefore, we only need to show that the length of the longest query-path involving only EXTEND elements is bounded by $O(\log^2 n)$ with high probability. Note that according to the partitioning, if we choose $\theta_i = \Omega(2^i \log(n)/(2K\Delta))$, by Lemma 5.26, after round i each EXTEND element has at most $(2K\Delta)/2^i$ incident elements. This implies that the number of parts is $O(\log n)$ (i.e. $d = O(\log n)$). Furthermore, by Lemma 5.28, the longest path consisting of EXTEND elements in each part has length at most $O(\log n)$.
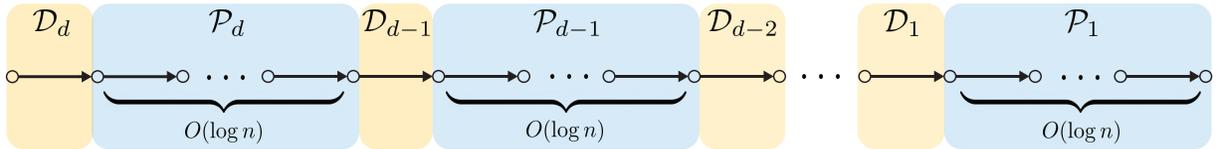

Figure 5: A possible query-path according to the partitioning. Blue boxes represent $\mathcal{P}_i$ and yellow boxes represent $\mathcal{D}_i$.

Now consider a query-path P. By Observation 5.24, at most one element of P is in each D_i for $i \leq d$. Moreover, by the above argument, there are at most $O(\log n)$ elements of P in each of $\mathcal{P}_i$

(see Figure 5). Therefore, the longest length of a query-path is bounded by $O(\log^2 n)$ with high probability (i.e. with probability of at least $1 - 1/n^2$).

For each unlikely permutation $\pi \in U$, there exists a query-path of length larger than β. Since $\beta = c \log^2 n$, by choosing c large enough we have that $|U|/|\Pi| \leq 1/n^2$. Therefore, $|U| \leq ((K + 1)m)!/n^2$ which implies that $|A_U| \leq ((K + 1)m)!/n^2$ since A_U represents vertices that correspond to unlikely permutations. $\qquad\square$

6 Our Estimator for the Adjacency List Model

In this section, we use the oracles introduced in the previous section to estimate the size of the matching in Algorithm 1. We assume that without loss of generality, the algorithm knows Δ, $\bar{d}$, and there is no singleton vertex in the graph (note that the algorithm can simply query degree of each vertex and compute Δ and $\bar{d}$).

Note that our upper bound of Section 5 is on $F(v, \pi)$ which recall is the number of recursive calls to the oracles, but not necessarily the running time needed to implement it. Generating the whole permutation π requires $|T| = \Theta(Km)$ time, which is too large for our purpose. Therefore, we have to generate π on the fly during the recursive calls whenever needed. Using the techniques developed first by [ORRR12] and further used by [Beh21], we show in Appendix B that indeed it is possible to get an $\widetilde{O}(F(v, \pi))$ time implementation. In particular, we show that:

Lemma 6.1. *In the adjacency list model, there is a data structure that given a graph G, (implicitly) fixes a random permutation π over its edge set. Then for any vertex v, the data structure returns whether v has any edge in outputs M and S of Algorithm 1 according to a random permutation π. Each query v to the data structure is answered in $\tilde{O}(F(v, \pi))$ time w.h.p. where $F(v, \pi)$ is as defined in Section 5. Additionally, the vertices we feed into the oracle can be adaptively chosen depending on the responses to the previous calls.*

In order to estimate the output of our algorithm, we sample r random vertices, and for each vertex, we check if it is the endpoint of a START element. Moreover, for each sampled vertex, we check if the vertex is an endpoint of a length three augmenting path that is created by a START element as a middle edge and two other EXTEND elements.

6.1 Multiplicative Approximation

We use the following well-known claim about the size of maximum matching, a similar bound to which was also used in [Beh21].

Claim 6.2. *Let G be a graph with maximum degree Δ and average degree $\bar{d}$. Then $\mu(G) \geq \frac{n\bar{d}}{4\Delta}$.*

Proof. Greedily edge color the graph using 2Δ colors. The color with the largest size is a matching of size at least $m/2\Delta = n\bar{d}/4\Delta$. $\qquad\square$

We use this claim to show that the number of samples that is needed to estimate the output of Algorithm 1 is $r = \widetilde{\Theta}(\Delta/\bar{d})$.

Remark 6.3. *In Algorithm 5, we do not estimate $\mu(M \cup S)$. Instead, we estimate the size of the maximal matching M and augment length-three augmenting paths in $M \cup S$. Since the bound*

Algorithm 5: Final algorithm for adjacency list.

---

1 $r \leftarrow (384 \cdot \Delta \log n)/(\delta^2 \bar{d})$.

2 Sample r vertices $u_1, u_2, \ldots, u_r$ uniformly at random from V with replacement.

3 Run vertex oracle for each u_i and let S_i and E_i be the indicator that u_i has an incident START and EXTEND elements that appear in the output of Algorithm 1.

4 for i *in* $1 \ldots r$ **do**

5 $\quad$ $X_i \leftarrow 0$

6 $\quad$ **if** $S_i = 1$ **then** $X_i \leftarrow 1$;

7 $\quad$ **if** $S_i = 0$ *and* $E_i = 1$ **then**

8 $\quad\quad$ Let w_i be the other endpoint of EXTEND element incident to u.

9 $\quad\quad$ Run vertex oracle for w_i (with the same permutation) and let S_i' be the indicator that w_i has an incident START element.

10 $\quad\quad$ **if** $S_i' = 0$ **then continue**;

11 $\quad\quad$ Let x_i be other endpoint of START element incident to w_i.

12 $\quad\quad$ Run vertex oracle for x_i (with the same permutation) and let E_i' be the indicator that x_i has an incident EXTEND element.

13 $\quad\quad$ **if** $E_i' = 0$ **then continue**;

14 $\quad\quad$ Let y_i be other endpoint of EXTEND element incident to x_i.

15 $\quad\quad$ Run vertex oracle for y_i (with the same permutation) and let S_i'' be the indicator that y_i has an incident START element.

16 $\quad\quad$ **if** $S_i'' = 0$ **then** $X_i \leftarrow 1$;

17 Let $X = \sum_{i \in [r]} X_i$ and $f = X/r$.

18 Let $\tilde{\mu} = (1 - \frac{\delta}{2}) fn/2$.

19 return $\tilde{\mu}$

---

in Theorem 4.1 is based on augmenting length three augmenting paths of $M \cup S$, we get the same approximation guarantee.

We can change the vertex oracle and edge oracle to return the incident elements that appear in subgraph $M \cup S$. However, for simplicity of oracles, we return the indicators for having START and EXTEND elements but we assume that we have access to these elements.

Lemma 6.4. *Let $\tilde{\mu}$ be the output of Algorithm 5. With high probability,*

$$\left(\frac{1}{2} + \frac{\delta}{4} \right) \mu(G) \leq \tilde{\mu} \leq \mu(G),$$

where δ is as in the statement of Theorem 4.1.

Proof. Let X_i be the indicator for each vertex i which shows in output of Algorithm 1, either i has a START incident element, or it is an endpoint of a length three augmenting path created by a START element in the middle along with two EXTEND elements. Note that the way X_i is computed in Algorithm 5 is the same as the definition given earlier.

Let $\hat{M}(G, \pi)$ be the set of edges of the matching created by augmenting the length-three aug-

31

menting paths of $M \cup S$. By Remark 6.3 and Theorem 4.1,

$$\left(\frac{1}{2} + \delta\right) \mu(G) \leq \mathbf{E}_\pi \left|\hat{M}(G, \pi)\right| \leq \mu(G). \tag{8}$$

Since the number of matched vertices is twice the number of edges in the matching,

$$\mathbf{E}[X_i] = \Pr[X_i = 1] = \frac{2 \mathbf{E}_\pi \left|\hat{M}(G, \pi)\right|}{n}.$$

Similarly, we have

$$\mathbf{E}[X] = \frac{2r \mathbf{E}_\pi \left|\hat{M}(G, \pi)\right|}{n}. \tag{9}$$

Since X is the sum of r independent Bernoulli random variables, the Chernoff bound (Proposition 3.3) implies

$$\Pr[|X - \mathbf{E}[X]| \leq \sqrt{6 \mathbf{E}[X] \log n}] \leq 2 \exp\left(-\frac{6 \mathbf{E}[X] \log n}{3 \mathbf{E}[X]}\right) = \frac{2}{n^2}. \tag{10}$$

Now, $fn = Xn/r$. Therefore, using the above equation, fn is in the following range with probability $1 - 2/n^2$.

$$fn \in \frac{n(\mathbf{E}[X] \pm \sqrt{6 \mathbf{E}[X] \log n})}{r} = \frac{n \mathbf{E}[X]}{r} \pm \frac{\sqrt{6n^2 \mathbf{E}[X] \log n}}{r}$$

$$= 2 \mathbf{E} \left|\hat{M}(G, \pi)\right| \pm \sqrt{\frac{12n \mathbf{E} \left|\hat{M}(G, \pi)\right| \log n}{r}} \qquad \text{(By (9))}$$

$$= 2 \mathbf{E} \left|\hat{M}(G, \pi)\right| \pm \sqrt{\frac{n \mathbf{E} \left|\hat{M}(G, \pi)\right| \bar{d}}{32 \cdot \delta^{-2} \cdot \Delta}} \qquad \text{(Since } r = \frac{384 \cdot \Delta \log n}{\delta^2 \bar{d}}\text{)}$$

$$\geq 2 \mathbf{E} \left|\hat{M}(G, \pi)\right| \pm \sqrt{\frac{\mu(G) \mathbf{E} \left|\hat{M}(G, \pi)\right|}{8 \cdot \delta^{-2}}} \qquad \text{(Since } \mu(G) \geq \frac{n\bar{d}}{4\Delta}\text{)}$$

By (8), we have $2 \mathbf{E} \left|\hat{M}(G, \pi)\right| \geq \mu(G)$. Combining with Claim 6.2, we get

$$fn \in 2 \mathbf{E} \left|\hat{M}(G, \pi)\right| \pm \sqrt{\frac{\mathbf{E} \left|\hat{M}(G, \pi)\right|^2}{4\delta^{-2}}} = (2 \pm \frac{\delta}{2}) \mathbf{E} \left|\hat{M}(G, \pi)\right|.$$

Since $\tilde{\mu} = (1 - \frac{\delta}{2}) fn / 2$, we have that

$$(1 - \delta) \mathbf{E} \left|\hat{M}(G, \pi)\right| \leq \tilde{\mu} \leq \mathbf{E} \left|\hat{M}(G, \pi)\right|$$

Next, note that by (8), $(\frac{1}{2} + \delta) \mu(G) \leq \mathbf{E} \left|\hat{M}(G, \pi)\right| \leq \mu(G)$. Hence,

$$(1 - \delta) \left(\frac{1}{2} + \delta\right) \mu(G) \leq \tilde{\mu} \leq \mu(G),$$

and thus

$$\left(\frac{1}{2} + \frac{\delta}{4}\right) \mu(G) \leq \tilde{\mu} \leq \mu(G). \quad \square$$

Lemma 6.5. *Algorithm 5 runs in $\tilde{O}(n + \Delta^{1+\varepsilon})$ with high probability.*

Proof. We show that for each of r sampled vertices in Algorithm 5, the iteration corresponding to the vertex in Line 4 takes $\tilde{O}(K\bar{d})$ time. First, note that for a random permutation π, the vertex oracle for a vertex u can be called a constant number of times. The reason is that the algorithm only queries vertex oracle for vertices of length three augmenting paths. Also, by Theorem 5.2, the vertex oracle takes $\tilde{O}(K\bar{d})$ time for random vertex and permutation. Therefore, if we run the vertex oracle a constant time for a fixed vertex, still the expected running time will be $\tilde{O}(K\bar{d})$. Since the number of samples is $O(\Delta \log n/\bar{d})$, the total running time for all samples will be $\tilde{O}(K\Delta)$. Furthermore, we spent $O(n)$ time for computing Δ, $\bar{d}$, and finding the isolated vertices.

In order to achieve a high probability bound on running time, we run $\Theta(\log n)$ instances of the algorithm simultaneously and return the estimation of the first instance that terminates. Since the expected running time is $\tilde{O}(n + K\Delta)$, the first instance terminates with probability $1 - 1/\text{poly}(n)$ in $\tilde{O}(n + K\Delta)$.

Plugging $K = \tilde{O}(\Delta^\varepsilon)$ completes the proof. □

6.2 Multiplicative-Additive Approximation

We use the same algorithm as Algorithm 5, however, the $o(n)$ additive error allows us to sample $\tilde{\Theta}(1)$ vertices instead of $\tilde{\Theta}(\Delta/\bar{d})$ vertices. Moreover, we no longer need to estimate $\bar{d}$ and Δ since the number of samples is independent of these parameters.

Lemma 6.6. *Let $\tilde{\mu}$ be the output of Algorithm 5 with parameter $r = 12 \log^3 n$ and estimation $\tilde{\mu} = fn/2 - \frac{n}{2 \log n}$. With high probability,*

$$\left(\frac{1}{2} + \delta\right)\mu(G) - \frac{n}{\log n} \leq \tilde{\mu} \leq \mu(G).$$

Proof. Let X_i be defined the same as Lemma 6.4. With the exact same argument, inequalities (8), (9), and (10) hold with new parameter and estimation. Hence, with probability of at least $1 - 2/n^2$,

$$fn \in \frac{n(\mathbf{E}[X] \pm \sqrt{6\,\mathbf{E}[X]\log n})}{r} = \frac{n\,\mathbf{E}[X]}{r} \pm \sqrt{\frac{6n^2\,\mathbf{E}[X]\log n}{r^2}}$$

$$= 2\,\mathbf{E}\,|\hat{M}(G,\pi)| \pm \sqrt{\frac{12n\,\mathbf{E}\,|\hat{M}(G,\pi)|\log n}{r}} \qquad \text{(By (9))}$$

$$= 2\,\mathbf{E}\,|\hat{M}(G,\pi)| \pm \sqrt{\frac{n\,\mathbf{E}\,|\hat{M}(G,\pi)|}{\log^2 n}} \qquad \text{(Since } r = 12 \cdot \log^3 n\text{)}$$

$$\in 2\,\mathbf{E}\,|\hat{M}(G,\pi)| \pm \frac{n}{\log n} \qquad \text{(Since } \mathbf{E}\,|\hat{M}(G,\pi)| \leq n\text{)}.$$

Since $\tilde{\mu} = fn/2 - \frac{n}{2 \log n}$, we have that

$$\mathbf{E}\,|\hat{M}(G,\pi)| - \frac{n}{\log n} \leq \tilde{\mu} \leq \mathbf{E}\,|\hat{M}(G,\pi)|.$$

By plugging (8), we get

$$\left(\frac{1}{2} + \delta\right)\mu(G) - \frac{n}{\log n} \leq \tilde{\mu} \leq \mu(G). \quad \square$$

33

Lemma 6.7. *Algorithm 5 with parameter $r = 12 \log^3 n$, runs in $\tilde{O}(\bar{d} \cdot \Delta^\varepsilon)$ with high probability.*

Proof. First, note that we do not need to spend $\tilde{O}(n)$ time to estimate Δ and $\bar{d}$, since the number of samples is independent of these parameters. By the exact same argument as the proof of Lemma 6.5, we can show that the expected running time for each sampled vertex is $\tilde{O}(K\bar{d})$. Since $r = \tilde{(O)}(1)$, the total running time will be $\tilde{O}(K\bar{d})$.

To get a high probability bound on the running time, similar to Lemma 6.5, we run $\Theta(\log n)$ instances of the algorithm simultaneously. Using the same argument, the first instance terminates with probability $1 - 1/\text{poly}(n)$ in $\tilde{O}(K\bar{d})$.

Plugging $K = \tilde{O}(\Delta^\varepsilon)$ completes the proof. □

7 Our Estimator for the Adjacency Matrix Model

In this section, we give a reduction from the adjacency matrix model to the adjacency list model such that each query in the adjacency list can be implemented with a constant number of queries in the adjacency matrix model. Such a reduction appears in [Beh21, Section 5]. We use a similar idea with a minor modification in the parameters of the construction.

Let $\gamma = (4 \log n) \cdot n$. We construct a graph $H = (V_H, E_H)$ as follows:

- V_H is the union of V_1, V_2 and $U_1, U_2, \ldots, U_n$ such that:
    - V_1 and V_2 are two copies of the vertex set of the original graph G.
    - U_i is a vertex set of size γ for each $i \in [n]$.
- We define the edge set such that degree of each vertex is in $\{1, n, n + \gamma\}$:
    - Degree of each vertex $v \in V_1$ is n. The i-th neighbor of v is the i-th vertex in V_1 if $(v, i) \in E$, otherwise, its i-th neighbor is the i-th vertex in V_2 for $i \leq n$. Note that graph $(V_1, E_H \cap (V_1 \times V_1))$ is isomorphic to G.
    - Degree of each vertex $v \in V_2$ is $n + \gamma$. The i-th neighbor of v is the i-th vertex in V_2 if $(v, i) \in E$, otherwise, its i-th neighbor is the i-th vertex in V_1 for $i \leq n$. For all $n < i \leq n + \gamma$, the i-th neighbor of v is i-th vertex in U_v.
    - Degree of each vertex $u \in U_i$ is one for $i \in [n]$. The only neighbor of u is the i-th vertex of V_2.

Observation 7.1. *For each vertex $v \in V_H$ and $i \in [\deg_H(v)]$, the i-th neighbor of vertex v can be determined using at most one query to the adjacency matrix.*

Proof. First, note that the degree of each vertex is not dependent on the original graph G. Hence, we do not need any queries to find the degree of each vertex. For each vertex $v \in U_1 \cup U_2 \cup \ldots \cup U_n$, one can find its only neighbor without any queries by the construction of H. For each vertex $v \in V_1 \cup V_2$, the i-th neighbor is either the i-th vertex of V_1 or the i-vertex of V_2. Therefore, with at most one query, one can determine the i-th neighbor of vertex v. □

Consider a random permutation π over the list of elements T, consisting of START and EXTEND copies of E_H. Intuitively, for almost all vertices $v \in V_2$, the first incident START element to v in T

34

is an edge between v and U_v. Similarly, the first two incident EXTEND elements to v are between v and U_v. We say v is a *bad* vertex, if it violates the mentioned conditions. Let $R \subseteq V_2$ be the set of bad vertices. Since the permutation over edges in $H[V_1 \cup R]$ is uniformly at random, using Theorem 4.1, we can provide a bound on the size of the matching produced by the algorithm.

Observation 7.2. *For each $v \in V_2 \setminus R$, the incident START and EXTEND elements in $\mathrm{MS}(H, \pi)$ are between v and U_v. Moreover, both incident START and EXTEND elements in $\mathrm{MS}(H, \pi)$ appears before all edges between v and $V_1 \cup V_2$.*

Proof. By the definition of R, the first START element incident to v in the permutation is between v and U_v. Let this edge be (v, w). Hence, the algorithm adds (v, w) to $\mathrm{MS}(H, \pi)$. Note that all START elements incident to v after (v, w) in the permutation cannot be in $\mathrm{MS}(H, \pi)$ since START elements create a maximal matching. By definition of R, the first two EXTEND elements incident to v are between v and U_v. Let (v, u_1) and (v, u_2) be these two edges ($u_1 \neq u_2$ since there is one EXTEND copy). Therefore, one of (v, u_1) and (v, u_2) must be added to $\mathrm{MS}(H, \pi)$ and no other EXTEND element incident to v can be added to $\mathrm{MS}(H, \pi)$ since EXTEND elements create a maximal matching. $\square$

Observation 7.3. *It holds that*

$$\left(\frac{1}{2} + \delta\right) \mu(H[V_1 \cup R]) \leq \mathbf{E}_\pi \left|\mathrm{MS}(H, \pi) \cap ((V_1 \cup R) \times (V_1 \cup R))\right| \leq \mu(H[V_1 \cup R]).$$

Proof. Note that by Observation 7.2, all vertices in $V_2 \setminus R$ have incident START and EXTEND elements in $\mathrm{MS}(H, \pi)$ which appear before edges between $V1 \cup R$ and $V_2 \setminus R$ in the permutation. Hence, none of these edges can be added to the subgraph $\mathrm{MS}(H, \pi)$ in Algorithm 1. Since the permutation over edges in $H[V_1 \cup R]$ is uniformly at random, using Theorem 4.1, we have the given bound $\square$

Next, we provide an upper bound for the size of R.

Observation 7.4. *It holds that $\mathbf{E}_\pi |R| \leq \frac{n}{2 \log n}$.*

Proof. For vertex $v \in V_2$ and a random permutation π over E_H, the first incident START element is between v and U_v, with probability of at least $\frac{K\gamma}{(n+\gamma)K} \geq 1 - \frac{1}{4 \log n}$. Furthremore, with probability $\frac{\gamma(\gamma-1)}{(n+\gamma)(n+\gamma-1)} \geq 1 - \frac{1}{4 \log n}$, the first two EXTEND elements are between v and U_v. Since these events are independent, the probability of v not being a bad vertex is at least

$$\left(1 - \frac{1}{4 \log n}\right)^2 \geq 1 - \frac{1}{2 \log n}.$$

Therefore, $|R|$ is at most $\frac{n}{2 \log n}$ in expectation over a random permutation. $\square$

With the intuition that a few vertices in V_2 are matched to vertices in $V_1 \cup V_2$, our goal is to estimate the number of vertices that have a matching edge in V_1. Since we have an upper bound on the size of the R, we are able to estimate the number of matching edges in $H[V_1] = H[G]$. In order to count the number of matching edges in V_1, we need to run the vertex oracle for vertices of $V_1 \cup V_2$. We show that the expected running time of vertex oracle on a random vertex of $V_1 \cup V_2$ is $\tilde{O}(n^{1+\varepsilon})$.

Claim 7.5. *Let v be a random vertex in $V_1 \cup V_2$ and π be a random permutation over E_H. Then*

$$\mathbf{E}_{v \sim (V_1 \cup V_2), \pi}[F(v, \pi)] = \tilde{O}(n^{1+\varepsilon}).$$

Proof. By Theorem 5.2, we have

$$\mathbf{E}_{v \sim V_H, \pi}[F(v, \pi)] = O\left(K \frac{|E_H|}{|V_H|} \log^4 |V_H|\right).$$

Summing over all vertices of V_H, we get

$$\sum_{v \in V_H} \mathbf{E}_\pi[F(v, \pi)] = O(K|E_H| \cdot \log^4 |V_H|) = \tilde{O}(n^{2+\varepsilon}),$$

because $|V_H| = O(n^2)$, $|E_H| = O(n^2 + n\gamma)$, $K = \tilde{O}(n^\varepsilon)$, and $\gamma = (4 \log n) \cdot n$. Therefore,

$$\mathbf{E}_{v \sim (V_1 \cup V_2), \pi}[F(v, \pi)] \leq \left(\sum_{v \in V_H} \mathbf{E}_\pi[F(v, \pi)]\right) / |(V_1 \cup V_2)| = \tilde{O}(n^{1+\varepsilon}). \quad \square$$

Claim 7.6. *Let v be a random vertex in V_1 and π be a random permutation over E_H. Then*

$$\mathbf{E}_{v \sim V_1, \pi}[F(v, \pi)] = \tilde{O}(n^{1+\varepsilon}).$$

Proof. Proof follows by combining Claim 7.5 and $|V_1| = |V_2|$. $\quad \square$

Lemma 7.7. *Let $\tilde{\mu}$ be the output of Algorithm 6. With high probability,*

$$\left(\frac{1}{2} + \delta\right)\mu(G) - \frac{n}{\log n} \leq \tilde{\mu} \leq \mu(G).$$

Proof. Let $\hat{M}(H, \pi)$ be the intersection of edges between V_1 and set of edges of the matching that is created by augmenting the length-three augmenting paths of $M \cup S$. We claim

$$\left(\frac{1}{2} + \delta\right)\mu(H[V_1]) - \frac{n}{2 \log n} \leq \mathbf{E}_\pi |\hat{M}(H, \pi)| \leq \mu(H[V_1]). \tag{11}$$

By combining Observation 7.3 and Observation 7.4, imply that there are at most $\frac{n}{2 \log n}$ edges in output of algorithm in $H[V_1 \cup R]$ with at least one endpoint in R. Therefore, by combining Remark 6.3, Theorem 4.1, and the bound for number of edges with at least one endpoint in R, we have the first inequality. Furthermore, since $\hat{M}(H, \pi)$ is a matching of $H[V_1]$, we have the second inequality.

By definition, X_i is the indicator of the event that a vertex $i \in V_1$ is matched in the output of Algorithm 1 to another vertex in V_1. Since the number of matched vertices is twice the number of matching edges,

$$\mathbf{E}[X_i] = \Pr[X_i = 1] = \frac{2\mathbf{E}|\hat{M}(H, \pi)|}{n}.$$

36

Algorithm 6: Final algorithm for adjacency matrix.

---

1 Let $H = (V_H, E_H)$ as described above.

2 $r \leftarrow 48 \cdot \log^3 n$.

3 Sample r vertices $u_1, u_2, \ldots, u_r$ uniformly at random from V_1 with replacement.

4 Run vertex oracle for each u_i and let S_i and E_i be the indicator that u_i has an incident START and EXTEND elements that appear in output of Algorithm 1.

5 for i *in* $1 \ldots r$ **do**

6 $X_i \leftarrow 0$

7   **if** $S_i = 1$ **then**

8   Let w_i be the other endpoint of START edge incident to u.

9   **if** $w_i \in V_1$ **then** $X_i \leftarrow 1$;

10   **if** $S_i = 0$ *and* $E_i = 1$ **then**

11   Let w_i be the other endpoint of EXTEND element incident to u.

12   Run vertex oracle for w_i (with same π) and let S'_i be the indicator that w_i has an incident START edge.

13   **if** $S'_i = 0$ **then continue**;

14   Let x_i be other endpoint of START element incident to w_i.

15   Run vertex oracle for x_i (with same π) and let E'_i be the indicator that x_i has an incident EXTEND element.

16   **if** $E'_i = 0$ **then continue**;

17   Let y_i be other endpoint of EXTEND element incident to x_i.

18   Run vertex oracle for y_i (with same π) and let S''_i be the indicator that y_i has an incident START element.

19   **if** $S''_i = 0$ *and* $w_i \in V_1$ **then** $X_i \leftarrow 1$;

20 Let $X = \sum_{i \in [r]} X_i$ and $f = X/r$.

21 Let $\tilde{\mu} = fn/2 - \frac{n}{4 \log n}$.

22 return $\tilde{\mu}$

Similarly,

$$\mathbf{E}[X] = \frac{2r \, \mathbf{E} \, |\hat{M}(H, \pi)|}{n}. \tag{12}$$

Since X is the sum of r independent Bernoulli random variables, the Chernoff bound (Proposition 3.3) implies

$$\Pr[|X - \mathbf{E}[X]| \leq \sqrt{6 \, \mathbf{E}[X] \log n}] \leq 2 \exp\left(-\frac{6 \, \mathbf{E}[X] \log n}{3 \, \mathbf{E}[X]}\right) = \frac{2}{n^2}.$$

Note that $fn = Xn/r$, so using the above equation, fn is in the following range with probability $1 - 2/n^2$:

$$fn \in \frac{n(\mathbf{E}[X] \pm \sqrt{6 \, \mathbf{E}[X] \log n})}{r} = \frac{n \, \mathbf{E}[X]}{r} \pm \sqrt{\frac{6n^2 \, \mathbf{E}[X] \log n}{r^2}}$$

$$= 2 \, \mathbf{E} \, |\hat{M}(H, \pi)| \pm \sqrt{\frac{12n \, \mathbf{E} \, |\hat{M}(H, \pi)| \log n}{r}} \tag{By (12)}$$

$$= 2\,\mathbf{E}\,|\hat{M}(H,\pi)| \pm \sqrt{\frac{n\,\mathbf{E}\,|\hat{M}(H,\pi)|}{4\log^2 n}} \qquad\qquad \text{(Since } r = 48 \cdot \log^3 n)$$

$$\in 2\,\mathbf{E}\,|\hat{M}(H,\pi)| \pm \frac{n}{2\log n} \qquad\qquad \text{(Since } \mathbf{E}\,|\hat{M}(H,\pi)| \leq n).$$

Now, $\tilde{\mu} = fn/2 - \frac{n}{4\log n}$. Therefore,

$$\mathbf{E}\,|\hat{M}(H,\pi)| - \frac{n}{2\log n} \leq \tilde{\mu} \leq \mathbf{E}\,|\hat{M}(H,\pi)|.$$

Combining the above range with (11) implies

$$\left(\frac{1}{2} + \delta\right)\mu(G) - \frac{n}{\log n} \leq \tilde{\mu} \leq \mu(G). \qquad \square$$

Lemma 7.8. *Algorithm 6 runs in $\tilde{O}(n^{1+\varepsilon})$ with high probability.*

Proof. First, we show that the iteration in Line 5, for each of r sampled vertices from V_1 takes $\tilde{O}(n^{1+\varepsilon})$ time in expectation. With the same argument as Lemma 6.5, the vertex oracle for a vertex u can be called a constant number of times. Moreover, by Claim 7.5 and Claim 7.6, the vertex oracle takes $\tilde{O}(n^{1+\varepsilon})$ time for a random vertex in $V_1 \cup V_2$ and a random permutation (since set R is a uniformly at random set in V_2). Therefore, if we run the vertex oracle constant time for a fixed vertex, still the expected running time will be $\tilde{O}(n^{1+\varepsilon})$.

In order to achieve a high probability bound on the running time, similar to the proof of Lemma 6.5, we run $\Theta(\log n)$ instances of the algorithm simultaneously. Using the same argument, the first instance terminates with probability $1 - 1/\mathrm{poly}(n)$ in $\tilde{O}(n^{1+\varepsilon})$. $\qquad \square$

Proof of Theorem 1.1.

- By combining Lemma 6.4 and Lemma 6.5, we get multiplicative factor of $(\frac{1}{2} + \delta)$ in the adjacency list model in $\widetilde{O}(n + \Delta^{1+\varepsilon})$ time.

- By combining Lemma 6.6 and Lemma 6.7, we get multiplicative-additive factor of $(\frac{1}{2} + \delta, o(n))$ in the adjacency list model in $\widetilde{O}(\bar{d} \cdot \Delta^{\varepsilon})$ time.

- By combining Lemma 7.7 and Lemma 7.8, we get multiplicative-additive factor of $(\frac{1}{2} + \delta, o(n))$ in the adjacency matrix model in $\tilde{O}(n^{1+\varepsilon})$ time. For a constant ε, running this algorithm for a slightly smaller value of ε allows us to get rid of polylogarithmic factor in the running time and achieve an $O(n^{1+\varepsilon})$ time algorithm. $\qquad \square$

8 Conclusion

We presented a new algorithm for finding a $(\frac{1}{2} + \Omega(1))$-approximate maximum matching and showed how its output size can be estimated in sublinear time. The algorithm gives a multiplicative $(\frac{1}{2} + \Omega(1))$-approximation of the size of maximum matching in the adjacency list model that runs in $\widetilde{O}(n + \Delta^{1+\varepsilon}) = O(n^{1+\varepsilon})$ time, where constant $\varepsilon > 0$ can be made arbitrarily small. This is the first algorithm that beats half-approximation in $o(n^2)$ time for all graphs.

Given that the barrier of $1/2$ is now broken, a natural question is what is the best approximation achievable in $n^{2-\Omega(1)}$ time. Is it possible to get a, say, .51-approximation? On the flip side, a lower bound ruling out say a .9999-approximation in $n^{2-\Omega(1)}$ time would also be extremely interesting.

References

[AB21] Sepehr Assadi and Soheil Behnezhad. Beating Two-Thirds For Random-Order Streaming Matching. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[AKL17] Sepehr Assadi, Sanjeev Khanna, and Yang Li. The stochastic matching problem: Beating half with a non-adaptive algorithm. In Constantinos Daskalakis, Moshe Babaioff, and Hervé Moulin, editors, *Proceedings of the 2017 ACM Conference on Economics and Computation, EC '17, Cambridge, MA, USA, June 26-30, 2017*, pages 99–116. ACM, 2017.

[BDH20] Soheil Behnezhad, Mahsa Derakhshan, and MohammadTaghi Hajiaghayi. Stochastic matching with few queries: (1-ε) approximation. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1111–1124. ACM, 2020.

[Beh21] Soheil Behnezhad. Time-Optimal Sublinear Algorithms for Matching and Vertex Cover. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 873–884. IEEE, 2021.

[BFS12] Guy E. Blelloch, Jeremy T. Fineman, and Julian Shun. Greedy Sequential Maximal Independent Set and Matching are Parallel on Average. In Guy E. Blelloch and Maurice Herlihy, editors, *24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '12, Pittsburgh, PA, USA, June 25-27, 2012*, pages 308–317. ACM, 2012.

[BHN16] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In Daniel Wichs and Yishay Mansour, editors, *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.

[BLM20] Soheil Behnezhad, Jakub Lacki, and Vahab S. Mirrokni. Fully Dynamic Matching: Beating 2-Approximation in Δ^ε Update Time. In Shuchi Chawla, editor, *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.

[CKK20] Yu Chen, Sampath Kannan, and Sanjeev Khanna. Sublinear Algorithms and Lower Bounds for Metric TSP Cost Estimation. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, pages 30:1–30:19, 2020.

[DF91] Martin E. Dyer and Alan M. Frieze. Randomized greedy matching. *Random Struct. Algorithms*, 2(1):29–46, 1991.

[FHTZ20] Matthew Fahrbach, Zhiyi Huang, Runzhou Tao, and Morteza Zadimoghaddam. Edge-weighted online bipartite matching. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 412–423. IEEE, 2020.

[FKM+05] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005.

[FN20] Manuela Fischer and Andreas Noever. Tight Analysis of Parallel Randomized Greedy MIS. *ACM Trans. Algorithms*, 16(1):6:1–6:13, 2020.

[GKM+19] Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 26–37. IEEE Computer Society, 2019.

[GS17] Guru Prashanth Guruganesh and Sahil Singla. Online matroid intersection: Beating half for random arrival. In Friedrich Eisenbrand and Jochen Könemann, editors, *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, volume 10328 of *Lecture Notes in Computer Science*, pages 241–253. Springer, 2017.

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[KMM12] Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In Anupam Gupta, Klaus Jansen, José D. P. Rolim, and Rocco A. Servedio, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, volume 7408 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2012.

[KMNT20] Michael Kapralov, Slobodan Mitrovic, Ashkan Norouzi-Fard, and Jakab Tardos. Space Efficient Approximation to Maximum Matching Size from Uniform Edge Samples. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1753–1772, 2020.

[NO08] Huy N. Nguyen and Krzysztof Onak. Constant-Time Approximation Algorithms via Local Improvements. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 327–336, 2008.

[ORRR12] Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. A Near-Optimal Sublinear-Time Algorithm for Approximating the Minimum Vertex Cover Size. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1123–1131, 2012.

[PR07] Michal Parnas and Dana Ron. Approximating the Minimum Vertex Cover in Sublinear Time and a Connection to Distributed Algorithms. *Theor. Comput. Sci.*, 381(1-3):183–196, 2007.

[YYI09] Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum matchings. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 225–234. ACM, 2009.

A   Deferred Proofs

Proof of Claim 3.1. For any odd i, let t_i be the number of augmenting paths of length i for M in $M \oplus M^\star$. Note that since M is maximal, we have $t_1 = 0$. Therefore,

$$|M^\star| - |M| = \sum_{i \geq 1} t_{2i+1} \leq t_3 + \sum_{i \geq 2} \frac{1}{2} i t_{2i+1} \leq \frac{1}{2} t_3 + \frac{1}{2} \sum_{i \geq 1} i t_{2i+1} = \frac{1}{2} t_3 + \frac{1}{2} |M|.$$

Moving the terms and using the assumption of $|M| < (\frac{1}{2} + \delta)|M^\star|$, we get

$$|M| - t_3 \leq |M| - (2|M^\star| - 3|M|) = 4|M| - 2\mu(G) \leq (2 + 4\delta)\mu(G) - 2\mu(G) = 4\delta\mu(G). \quad \square$$

B   Implementation Details

In this section, we provide an implementation of our vertex and edge oracle. The idea is similar to the [Beh21, Appendix A]. The main difference is that instead of having at most one edge between two vertices, here we have $K + 1$ edges where K of them correspond to START copies and one of them corresponds to EXTEND copy. The idea is to generate a random permutation π locally and sort edges based on π to create the permutation.

Note that in the vertex oracle and edge oracle, when an START incident element appears in $\mathrm{MS}(G, \pi)$, the algorithm no longer queries on the START incident elements (the same also holds for EXTEND elements). Therefore, instead of having one graph, we assume that we have two graphs G_s and G_e that are isomorphic to G. Also, we assume that each edge in G_s has K copies that is corresponds to the number of START elements. In other words, graph G_s is the a graph made by all START elements of G and G_e is the a graph made by all EXTEND elements of G.

Let $\mathrm{LOWEST}_{G_t}(u, i)$ for $t \in \{s, e\}$ be a procedure that returns a pair of an edge e in G_t and its ranking such that e is the i-th lowest rank edge incident to u in G_t. We use the same implementation of LOWEST_{G_t} procedure as the [Beh21, Appendix A] (note that in this paper, the procedure only returns the edge. However, in the implementation of the LOWEST, they compute the edge ranking and we can simply return the edge ranking). Let $\deg_{G_t}(u)$ be the degree of vertex u in graph G_t. By definition of the G_s and G_e, we have that $\deg_{G_s}(u) = K \deg_G(u)$ and $\deg_{G_e}(u) = \deg_G(u)$.

Claim B.1 ([Beh21]). *Let u be a vertex and suppose that we call procedure $\mathrm{LOWEST}_{G_t}(u, i)$ for all $1 \leq i \leq j$. The total time to implement all these calls is $\tilde{O}(j)$ with high probability for all $u \in V$.*

We present the implementation of our oracles in the following three algorithms. Note that we can generate vertex colors (i.e. c_v for vertex v) on the fly when it is needed. Also, we can toss a coin with a probability $1 - p$ for each edge to determine if it is a frozen edge or not. Therefore, once we have the rank of edges in our oracles, we can distinguish whether an edge is frozen or not based on the Definition 5.1. However, to make algorithms easier to read, we do not include the technical details of this part.

Algorithm 7: Implementation of the vertex oracle $\text{VO}(u)$.

1 $j_s \leftarrow 1, j_e \leftarrow 1$
2 $v_s, \pi_s \leftarrow \text{LOWEST}_{G_s}(u, j_s)$
3 $v_e, \pi_e \leftarrow \text{LOWEST}_{G_e}(u, j_e)$
4 $ST \leftarrow \text{FALSE}, EX \leftarrow \text{FALSE}$
5 **while** $j_s \leq \deg_{G_s}(u)$ *or* $j_e \leq \deg_{G_e}(u)$ **do**
6   **if** $\pi_s < \pi_e$ **then**
7 $X \leftarrow \text{START}$
8 $\ell = ((u, v_s), X)$
9   **else**
10 $X \leftarrow \text{EXTEND}$
11 $\ell = ((u, v_e), X)$
12   **if** $X = \text{EXTEND}$ *and* $c_u = c_{v_e}$ **then continue**;
13   **if** $ST = \text{FALSE}$ *and* $X = \text{START}$ *and* $\text{EOS}(\ell, v_s) = \text{TRUE}$ **then**
14 $ST \leftarrow \text{TRUE}$
15 $j_s \leftarrow j_s + 1$
16   **if** $j_s \leq \deg_{G_s}(u)$ **then**
17 $v_s, \pi_s \leftarrow \text{LOWEST}_{G_s}(u, j_s)$
18   **else**
19 $v_s, \pi_s \leftarrow \infty, \infty$
20   **if** $EX = \text{FALSE}$ *and* $X = \text{EXTEND}$ *and* $\text{EOE}(\ell, v_e, ST) = \text{TRUE}$ **then**
21 $EX \leftarrow \text{TRUE}$
22 $j_e \leftarrow j_e + 1$
23   **if** $j_e \leq \deg_{G_e}(u)$ **then**
24 $v_e, \pi_e \leftarrow \text{LOWEST}_{G_e}(u, j_e)$
25   **else**
26 $v_e, \pi_e \leftarrow \infty, \infty$
27   **return** ST, EX

Algorithm 8: Implementation of the edge oracle for START elements $\text{EOS}(\ell, u)$.

1 **if** $\text{EOS}(\ell, u)$ *is already computed* **then return** the computed answer;
2 $j \leftarrow 1$
3 $w, \pi_w \leftarrow \text{LOWEST}_{G_s}(u, j)$
4 **while** $w \neq v$ **do**
5 $\ell' \leftarrow ((u, w), \text{START})$
6   **if** $\text{EOS}(\ell', w) = \text{TRUE}$ **then return** FALSE;
7 $j \leftarrow j + 1$
8 $w \leftarrow \text{LOWEST}_{G_s}(u, j)$
9 **return** TRUE

Algorithm 9: Implementation of the edge oracle for EXTEND elements $\text{EOE}(\ell, u, ST_w)$.

---

1 **if** $\text{EOE}(\ell, u, ST_w)$ *is already computed* **then return** the computed answer;
2 $j_s \leftarrow 1, j_e \leftarrow 1$
3 $v_s, \pi_s \leftarrow \text{LOWEST}_{G_s}(u, j_s)$
4 $v_e, \pi_e \leftarrow \text{LOWEST}_{G_e}(u, j_e)$
5 $ST_u \leftarrow \text{FALSE}$
6 **if** $\pi_s < \pi_e$ **then**
7 $\quad\lfloor\ w \leftarrow v_s, X' \leftarrow \text{START}$
8 **else**
9 $\quad\lfloor\ w \leftarrow v_e, X' \leftarrow \text{EXTEND}$
10 $\ell' \leftarrow ((u, w), X')$
11 **while** $w \neq v$ *or* $X' \neq X$ **do**
12 $\quad$ **if** $X' = \text{EXTEND}$ *and* $c_u = c_w$ **then continue**;
13 $\quad$ **if** $X' = \text{START}$ **then**
14 $\quad\quad$ **if** $ST_u = \text{FALSE}$, *and* $\text{EOS}(\ell', w) = \text{TRUE}$ **then**
15 $\quad\quad\quad$ $ST_u \leftarrow \text{TRUE}$
16 $\quad\quad\quad$ **if** ℓ' *is frozen* **then return** FALSE;
17 $\quad\quad\quad\lfloor$ **if** $ST_w = \text{TRUE}$ **then return** FALSE;
18 $\quad\quad$ $j_s \leftarrow j_s + 1$
19 $\quad\quad$ **if** $j_s \leq \deg_{G_s}(u)$ **then**
20 $\quad\quad\quad\lfloor$ $v_s, \pi_s \leftarrow \text{LOWEST}_{G_s}(u, j_s)$
21 $\quad\quad$ **else**
22 $\quad\quad\quad\lfloor$ $v_s, \pi_s \leftarrow \infty, \infty$
23 $\quad$ **if** $X' = \text{EXTEND}$ **then**
24 $\quad\quad$ **if** $\text{EOE}(\ell', w, ST_u) = \text{TRUE}$ **then return** FALSE;
25 $\quad\quad$ $j_e \leftarrow j_e + 1$
26 $\quad\quad$ **if** $j_e \leq \deg_{G_e}(u)$ **then**
27 $\quad\quad\quad\lfloor$ $v_e, \pi_e \leftarrow \text{LOWEST}_{G_e}(u, j_e)$
28 $\quad\quad$ **else**
29 $\quad\quad\quad\lfloor$ $v_e, \pi_e \leftarrow \infty, \infty$
30 $\quad$ **if** $\pi_s < \pi_e$ **then**
31 $\quad\quad\lfloor$ $w \leftarrow v_s, X' \leftarrow \text{START}$
32 $\quad$ **else**
33 $\quad\quad\lfloor$ $w \leftarrow v_e, X' \leftarrow \text{EXTEND}$
34 $\quad\lfloor$ $\ell' \leftarrow ((u, w), X')$
35 **return** TRUE

Lemma B.2. *In the adjacency list model, there is a data structure that given a graph G, (implicitly) fixes a random permutation π over its edge set. Then for any vertex v, the data structure returns whether v has any edge in outputs M and S of [Algorithm 1] according to a random permutation π. Each query v to the data structure is answered in $\tilde{O}(F(v, \pi))$ time w.h.p. where $F(v, \pi)$ is as defined in [Section 5]. Additionally, the vertices we feed into the oracle can be adaptively chosen*

depending on the responses to the previous calls.

Proof. Note that in Algorithm 7, we only call LOWEST procedure for the neighbors that we recursively call edge oracle for them. Similarly, in Algorithm 8 and Algorithm 9 we only call LOWEST procedure for incident elements that we will recursively call the edge oracle on them. Therefore, by Claim B.1, the total time spent on LOWEST procedure calls is $\tilde{O}(F(v, \pi))$ for a random permutation π and every vertex v with high probability. $\qquad\square$