

# Time-Optimal Sublinear Algorithms for Matching and Vertex Cover\*

Soheil Behnezhad<sup>†</sup>  
*Stanford University*

## Abstract

We study the problem of estimating the size of maximum matching and minimum vertex cover in sublinear time. Denoting the number of vertices by  $n$  and the average degree in the graph by  $\bar{d}$ , we obtain the following results for both problems:<sup>1</sup>

- A multiplicative  $(2 + \varepsilon)$ -approximation that takes  $\tilde{O}(n/\varepsilon^2)$  time using adjacency list queries.
- A multiplicative-additive  $(2, \varepsilon n)$ -approximation in  $\tilde{O}((\bar{d} + 1)/\varepsilon^2)$  time using adjacency list queries.
- A multiplicative-additive  $(2, \varepsilon n)$ -approximation in  $\tilde{O}(n/\varepsilon^3)$  time using adjacency matrix queries.

All three results are provably time-optimal up to polylogarithmic factors culminating a long line of work on these problems.

Our main contribution and the key ingredient leading to the bounds above is a new and near-tight analysis of the *average query complexity* of the randomized greedy maximal matching algorithm which improves upon a seminal result of Yoshida, Yamamoto, and Ito [STOC'09].

---

\*A preliminary version of this paper appeared in proceedings of FOCS 2021.

<sup>†</sup>Research was done in part while the author was funded by a Google PhD Fellowship at the University of Maryland.

<sup>1</sup>The  $\tilde{O}(\cdot)$  notation hides poly log  $n$  factors throughout the paper.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Applications of the New Query-Complexity Analysis . . . . .	2
1.2	Our Techniques & Background on the Query-Complexity of RGMM . . . . .	4
<b>2</b>	<b>Preliminaries</b>	<b>5</b>
<b>3</b>	<b>Average Query-Complexity of RGMM</b>	<b>6</b>
3.1	Proof of Lemma 3.12 . . . . .	9
3.2	Proof of Lemma 3.13 . . . . .	12
<b>4</b>	<b>The Final Algorithms for the Adjacency List Query Model</b>	<b>13</b>
4.1	Proof of Theorem 1.2: Multiplicative Approximation . . . . .	14
4.2	Proof of Theorem 1.3: Multiplicative-Additive Approximation . . . . .	16
<b>5</b>	<b>The Final Algorithm for the Adjacency Matrix Query Model</b>	<b>17</b>
<b>6</b>	<b>Acknowledgements</b>	<b>21</b>
<b>A</b>	<b>Implementation Details</b>	<b>24</b>

# 1 Introduction

We study algorithms for estimating the size of maximum matching or minimum vertex cover in general graphs. A *maximal* matching, which can be trivially constructed in linear time, leads to simple 2-approximations for both problems. But can we achieve the same in *sublinear* time?

Since sublinear-time algorithms cannot even read the whole data, it is important to specify how the input is presented. For graphs, two models have been commonly considered in the literature:

- **The adjacency list model:** In this model, for any vertex  $v$  of its choice, the algorithm may query the degree of  $v$  in the graph and, for any  $1 \leq i \leq \deg(v)$ , may query the  $i$ -th neighbor of  $v$  stored in an arbitrarily ordered list.
- **The adjacency matrix model:** In this model, the algorithm may query, for any vertex-pair  $(u, v)$  of its choice, whether or not  $u$  and  $v$  are adjacent in the graph.

We consider both models in this work.

Although at the first glance it may seem impossible to do much without reading the whole input, numerous sublinear-time algorithms have been designed over the years for various optimization problems. In addition to matching and vertex cover, which have been studied extensively in the area [25, 23, 26, 24, 19, 9], the list includes estimating the weight/size of minimum spanning tree (MST) [8, 10], traveling salesman problem (TSP) [9],  $k$ -nearest neighbor graph [12], graph’s average degree [15, 18], as well as problems such as vertex coloring [2], metric linear sampling [13], and many others. (This is by no means a comprehensive list of all the prior works.) For some of the classic results of the area, see the excellent survey of Czumaj and Sohler [11].

The famous *randomized greedy maximal matching* (RGMM) algorithm is the basis of one of the most successful approaches for estimating the size of matching and vertex cover in sublinear time. The RGMM algorithm iterates over the edges in a *random* order and greedily adds each encountered edge to the matching if not already incident to a matching edge. Although it takes a linear time to run this algorithm on the whole graph, one can locally determine if a single vertex is part of the solution much faster. The key observation is the fact that an edge is in the output matching of RGMM, if and only if it has no neighboring edge earlier in the ordering that belongs to the matching. This naturally gives rise to a recursive query process, suggested first by Nguyen and Onak [23], that explores the local neighborhood of a given edge or vertex, and determines if it is matched in an instance of RGMM (we formalize this in Section 3).

To utilize this local simulation of RGMM, the most common approach is to plug it into the framework of Parnas and Ron [25]: Pick a number of random vertices in the graph, simulate the local RGMM on them, and report the fraction of them that are matched as an estimate for the fraction of vertices matched in the whole graph. The final time-complexity of the algorithm, therefore, depends on the size of the local neighborhood that one has to explore for each randomly chosen vertex, which is also known as the “average query-complexity” of RGMM [23, 26, 24].

Indeed, our main technical contribution is the following near-tight analysis of the average query-complexity of RGMM, which improves upon a seminal work of Yoshida, Yamamoto, and Ito [26].

**Theorem 1.1** (See Theorem 3.5 for the formal statement). *For any  $n$ -vertex graph with average degree  $\bar{d}$ , the average query-complexity of RGMM is  $O(\bar{d} \cdot \log n)$ .*

We note that in a complete bipartite graph with  $\Theta(\bar{d})$  vertices in one part and  $\Theta(n)$  vertices in the other, for  $n \gg \bar{d}$ , the average query-complexity is  $\Omega(\bar{d})$ . Hence Theorem 1.1 is almost tight.

The result of Yoshida *et al.* [26], in comparison, bounds the query-complexity of a random vertex by  $O(L/n)$ , where  $L$  is the number of edges in the line-graph. In general,  $L/n$  can be upper bounded by  $O(\bar{d} \cdot \Delta)$  where  $\Delta$  is the maximum degree, and there are graphs<sup>2</sup> for which  $L/n = \Omega(\bar{d} \cdot \Delta)$ . We elaborate more on this result of [26] and further compare it to Theorem 1.1 in Section 1.2.

## 1.1 Applications of the New Query-Complexity Analysis

Theorem 1.1 leads to a number of sublinear-time algorithms for matching and vertex cover that are provably time-optimal up to logarithmic factors. We elaborate on these applications here and also mention some other applications of Theorem 1.1.

As before, in all the statements below (and throughout the paper) we use  $n$  to denote the number of vertices,  $\Delta$  to denote the maximum degree, and  $\bar{d}$  to denote the average degree.

**Application 1:** The first application is a multiplicative approximation in the adjacency list model.

**Theorem 1.2.** *For any  $\varepsilon > 0$ , there is an algorithm that with probability  $1 - 1/\text{poly}(n)$  reports a  $(2 + \varepsilon)$ -approximation to the size of maximum matching and that of minimum vertex cover using  $O(n) + \tilde{O}(\Delta/\varepsilon^2)$  time and queries in the adjacency list model.*

Observe that  $\Omega(n)$  queries are necessary even to distinguish an empty graph from one that includes a single edge. As such,  $\Omega(n)$  time and queries are necessary for any multiplicative approximation in this model, implying that Theorem 1.2 is nearly time-optimal.

Theorem 1.2, notably, gives the *first* multiplicative estimator in the literature that runs in  $\tilde{O}(n)$  time for all graphs. For a  $(2 + \varepsilon)$ -approximation, in particular, no  $o(n^2)$  time algorithm was known for general graphs prior to our work. Allowing a larger  $O(1)$ -approximation, a recent result of [20] with some work leads to a  $O(n + \Delta^2/\bar{d})$  time algorithm which can take  $\tilde{\Omega}(n\sqrt{n})$  time.

Interestingly, under the rather mild assumptions that  $\bar{d} = \Omega(1)$  (which, e.g., holds if there are no singleton vertices) and that (estimates of)  $\bar{d}$  and  $\Delta$  are given, the  $\Omega(n)$  lower bound is avoidable and the algorithm of Theorem 1.2 actually runs in  $\tilde{O}(\Delta/\varepsilon^2)$  time.

**Application 2:** The second application of Theorem 1.1 is a multiplicative-additive approximation (see Section 2 for the formal definition), also in the adjacency list model.

**Theorem 1.3.** *For any  $\varepsilon > 0$ , there is an algorithm that with probability  $1 - 1/\text{poly}(n)$  reports a  $(2, \varepsilon n)$ -approximation to the size of maximum matching and that of minimum vertex cover using  $\tilde{O}((\bar{d} + 1)/\varepsilon^2)$  time and queries in the adjacency list query model.*

Theorem 1.3 (nearly) matches an  $\Omega(\bar{d} + 1)$  lower bound of Parnas and Ron [25] that holds for any  $(O(1), \varepsilon n)$ -approximation of maximum matching or minimum vertex cover in this model. This culminates a long line of work [25, 23, 26, 24, 20] on this problem that we overview next.

Parnas and Ron [25] were the first to give a multiplicative-additive approximation for matching and vertex cover. They showed how to obtain a  $(2, \varepsilon n)$ -approximation in  $\Delta^{O(\log(\Delta/\varepsilon))}$  time by simulating a distributed local algorithm for each sampled vertex. A quasi-polynomial dependence on  $\Delta$ , however, is unavoidable with this approach due to existing distributed lower bounds [22].

The next wave of results [23, 26, 24] were based on the RGMM algorithm. Yoshida *et al.* [26] built on the work of Nguyen and Onak [23] and, notably, gave the first algorithm with a polynomial-

<sup>2</sup>Consider a complete bipartite graph with  $\Delta + 1$  vertices in one part and  $\Theta(\bar{d})$  vertices in the other part.

in- $\Delta$  time-complexity of  $O(\Delta^4/\varepsilon^2)$  for a  $(2, \varepsilon n)$ -approximation. Onak *et al.* [24] later shaved off some of the  $\Delta$  factors from the bound of [26]. Although a bound of  $\tilde{O}_\varepsilon(\Delta)$  was first claimed in [24], Chen, Kannan, and Khanna [9] discovered a subtlety with a claim of [24] that happens to be crucial for their final result. Nonetheless, as also observed by Chen *et al.* [9], the techniques developed in [24] combined with the average query-complexity analysis of [26] discussed above, still implies an improved bound of  $\tilde{O}((\bar{d} \cdot \Delta + 1)/\varepsilon^2) = \tilde{O}(\Delta^2/\varepsilon^2)$  for a  $(2, \varepsilon n)$ -approximation.<sup>3</sup>

Observe that all the algorithms of the literature discussed above require some upper bound on the degrees to run in sublinear-time and can take up to  $\Omega(n^2)$  time for general graphs. A more recent algorithm by Kapralov *et al.* [20] has a more desirable time-complexity of  $O(\Delta/\varepsilon^2)$ . However, it only obtains an  $(O(1), \varepsilon)$ -approximation and as pointed out by Chen, Kannan, and Khanna [9]:

“Unfortunately, the constant hidden in the  $O(1)$  notation [of [20]] is very large, and efficiently obtaining a  $(2, \varepsilon n)$ -approximation to matching size remains an important open problem” [9].

Theorem 1.3, in light of the lower bound of [25], resolves this open problem of [9] in a strong sense.

**Application 3:** The third application is in the adjacency matrix model.

**Theorem 1.4.** *For any  $\varepsilon > 0$ , there is an algorithm that with probability  $1 - 1/\text{poly}(n)$  reports a  $(2, \varepsilon n)$ -approximation to the size of maximum matching and that of minimum vertex cover using  $\tilde{O}(n/\varepsilon^3)$  time and queries in the adjacency matrix query model.*

A similar bound to Theorem 1.4 was claimed in [24]. However, as discussed above, the proof of [24] had a subtlety. Chen *et al.* [9] proposed a fix, but their algorithm runs in  $\tilde{O}_\varepsilon(n\sqrt{n})$  time. Theorem 1.4 improves this latter bound by a factor of  $\sqrt{n}$ .

On the lower bound side, suppose that the graph is either a random perfect matching or is empty. It is not hard to see that distinguishing the two cases requires  $\Omega(n)$  queries to the adjacency matrix. As such,  $\Omega(n)$  queries are necessary for any multiplicative-additive approximation in the adjacency-matrix model, implying that Theorem 1.4 is nearly time-optimal. Note that distinguishing an empty graph from one that includes only one edge requires  $\Omega(n^2)$  queries in the adjacency-matrix model. This implies that, unlike Theorem 1.2 for the adjacency-list model, no non-trivial multiplicative approximation can be obtained in the adjacency-matrix model.

To prove Theorem 1.4, in addition to Theorem 1.1, we give a new reduction from the adjacency matrix model to the adjacency list model that, unlike the reduction of [24], does not lead to parallel edges or self-loops. This is crucial for our result; see Section 5 for details.

**Other Applications – TSP:** Chen *et al.* [9] gave sublinear time algorithms for estimating the size of variants of TSP such as graphic TSP in the adjacency matrix model. Their algorithms utilize a matching size estimator as a subroutine. Plugging Theorem 1.4 into their framework implies that:

**Corollary 1.5** (of using Theorem 1.4 in the algorithms of [9]). *There is an  $\tilde{O}(n)$ -time randomized algorithm that estimates the cost of graphic TSP to within a factor of  $(27/14)$ .*

Instead of Theorem 1.4, Chen *et al.* [9] used their  $\tilde{O}(n\sqrt{n})$ -time matching size estimator in their algorithms. As a result, their algorithms were slower by a factor of  $\sqrt{n}$  than those in Corollary 1.5.

<sup>3</sup>We note that while Theorem 1.3 can be seen as a fix to [24], our techniques are very different from the approach of [24]. Particularly, [24] did not claim the tight upper bound of Theorem 1.1 that we prove in this paper, but rather claimed a weaker bound of  $O(\bar{d} \cdot \rho)$  where  $\rho$  is the ratio of the maximum degree over the min degree. As a result, even if one manages to fix the proof of [24], one does not get the strong multiplicative approximation of Theorem 1.2.

**Other Applications – AMPC:** The *adaptive massively parallel computations* (AMPC) model was first introduced by [3] and has been further studied by [4, 7]. While the precise definition of the AMPC model is out of the scope of this paper, it augments the standard MPC model of [21] with a distributed hash table. Combined with the techniques developed for the maximal independent set problem in [3], Theorem 1.1 implies an  $O(1)$ -round AMPC algorithm for maximal matching using a strongly sublinear space of  $O(n^\delta)$  per machine (for any constant  $\delta > 0$ ) and an optimal total space of  $\tilde{O}(m)$  in  $m$ -edge graphs. This has to be compared with an  $O(\log \log n)$ -round algorithm presented in [4] that has the same space complexity.

The essence of the improved AMPC algorithm mentioned above is the surprising fact, implicit in Theorem 1.1, that if one starts the local simulation of RGMM from *every* vertex in the graph all in parallel, then the expected sum of the query-complexities of all these parallel calls, or equivalently the “total work” of the algorithm, is  $n \cdot O(\bar{d} \log n) = O(m \log n)$ , i.e., near-linear in the input size!

## 1.2 Our Techniques & Background on the Query-Complexity of RGMM

As discussed above, our main technical contribution is the upper bound of Theorem 1.1 on the average query-complexity of the randomized greedy maximal matching algorithm. In this section, we provide more context about this result, further compare it to the previous bounds, and give an overview of our techniques and new insights for proving it.

Let us start by giving a slightly more formal definition of the local simulation of RGMM due to [23]. Suppose that we would like to define an oracle,  $\text{EO}(e, \pi)$ , to determine whether a given edge  $e$  belongs to the matching  $\text{GMM}(G, \pi)$  returned by the greedy maximal matching algorithm processing the edges in the order of  $\pi$ . Nguyen and Onak [23] observed that  $e$  belongs to  $\text{GMM}(G, \pi)$  if and only if no edge incident to  $e$  with a lower  $\pi$ -rank than  $e$  belongs to  $\text{GMM}(G, \pi)$ . Therefore, one can recursively run  $\text{EO}(e', \pi)$  on all incident edges  $e'$  to  $e$  with  $\pi(e') < \pi(e)$ . If none of these neighbors join  $\text{GMM}(G, \pi)$ , we know for sure that  $e \in \text{GMM}(G, \pi)$ , and otherwise  $e \notin \text{GMM}(G, \pi)$ .

The next crucial observation is that as soon as we find just one neighboring edge  $e'$  of  $e$  that is in the matching, we can terminate  $\text{EO}(e, \pi)$  and report  $e \notin \text{GMM}(G, \pi)$  without going through the rest of the neighbors of  $e$ . Intuitively, this “pruning” should have a significant effect on the average query-complexity due to the recursive nature of the oracle. But how can we prove it? An immediate problem is that analyzing the expected query-complexity for a *given* edge  $e$  seems to be challenging. In fact, obtaining a  $\text{poly}(\Delta, \log n)$  bound for this problem remains a major open question in the study of local computation algorithms (LCA’s) [1, 17].

In a beautiful paper, Yoshida, Yamamoto, and Ito [26] got around this challenge and proved a  $\text{poly}(\Delta)$  upper bound on the *average* query-complexity with a global analysis. Namely, instead of bounding the expected number of queries that an edge  $e$  generates, they showed that for any edge  $e = (u, v)$  the expected number of edges in the graph whose query process reaches  $e$  is at most  $O(\deg(u) + \deg(v))$ .<sup>4</sup> This way, the expected sum of all queries starting from all the edges in the graph, can be upper bounded by  $\sum_{(u,v) \in E} O(\deg(u) + \deg(v)) = O(L)$ , where  $L$  is the number of edges in the line-graph. This implies that for an *edge* chosen uniformly at random, the expected query-complexity can be upper bounded by  $O(L/m)$ . Although it is not immediate, we note that essentially the same proof also implies that for a *vertex* chosen uniformly at random, the expected query-complexity is  $O(L/n)$  which is  $O(\bar{d} \cdot \Delta)$  and can also be as large as  $\Omega(\bar{d} \cdot \Delta)$  (see Footnote 2).

Our new analysis builds on some of the ideas introduced by Yoshida *et al.* [26]. Similar to their work and, crucially, instead of directly analyzing the number of recursive calls *out* of an edge or

---

<sup>4</sup>The analysis of [26] proceeds on the line-graph and  $\deg(u) + \deg(v)$  is the degree of  $e$  in the line-graph.

a vertex, we analyze the recursive calls made *to* an edge or a vertex. There are, however, several crucial differences between the two approaches that leads to our near-tight analysis in Theorem 1.1.

We prove that for any edge  $e = (u, v)$  in the graph, the expected number of starting queries that reach  $e$  can be upper bounded by  $O(\log n)$ . This improves over the previous upper bound of  $O(\deg(u) + \deg(v))$  by Yoshida *et al.* [26] and is the key component of our analysis.

Let us fix edge  $e$  and overview how we prove the bound of  $O(\log n)$  on the total number of queries to  $e$ , from all possible starting points. To prove this upper bound, if in a permutation  $\pi$  we happen to query  $e$  for  $q$  times, we charge  $q$  other permutations  $\sigma_1, \dots, \sigma_q$ . Observe that by a simple double counting argument, the expected number of times that a random permutation  $\sigma \in \Pi$  is charged by all other permutations combined, is exactly equal to the expected number of queries to  $e$  in a random permutation  $\pi$ . As such, if we prove an upper bound of  $T$  on the number of times that each permutation is charged, then we get that  $e$  is queried by at most  $T$  starting points in expectation. Unfortunately, however, we remark that there will be permutations that get charged up to even  $\Omega(n)$  times with our charging method.

To get past this hurdle, we draw a novel connection to an orthogonal line of work on bounding the *parallel depth* of RGMM pioneered by Blelloch, Fineman, and Shun [5] whose bounds were tightened by Fischer and Noever [16], showing that these algorithms have parallel depth  $O(\log n)$  w.h.p. By carefully analyzing the structure of RGMM queries, we show that if a permutation  $\sigma$  is charged  $b$  times with our charging method, then for any  $1 \leq i \leq b$ , at least  $i$  of these charges should be from permutations with parallel depth at least  $\Omega(b - i)$ . In particular, if a permutation  $\sigma$  is charged  $\omega(\log n)$  times, most of these charges must be from those “unlikely” permutations that have a high parallel depth  $\omega(\log n)$ . With a few additional ideas (particularly by caching previous queries) we show that the queries of these unlikely permutations can be effectively bounded, and get an  $O(\log n)$  upper bound on the number of charges to an average permutation  $\sigma$ .

## 2 Preliminaries

Unless otherwise stated, we use  $n$  to denote the number of vertices,  $m$  to denote the number of edges,  $\Delta$  to denote the maximum degree, and  $\bar{d}$  to denote the average degree in the original graph. The graphs considered in this paper are unweighted and simple. We use  $\mu(G)$  to denote the size of the maximum matching in  $G$  and use  $\nu(G)$  to denote the size of the minimum vertex cover in  $G$ . For any  $U \subseteq V$ , we use  $G[U]$  to denote the induced subgraph of  $G = (V, E)$  on  $U$ , which is the subgraph obtained by removing all vertices in  $V \setminus U$  (along with their edges) from  $G$ . For any positive integer  $k$  we use  $[k]$  to denote the set  $\{1, \dots, k\}$ .

As standard in the literature, for  $\alpha \geq 1$  and  $0 \leq \varepsilon \leq 1$ , we say an estimate  $\tilde{\mu}(G)$  for  $\mu(G)$  and an estimate  $\tilde{\nu}(G)$  for  $\nu(G)$  provide “multiplicative-additive”  $(\alpha, \varepsilon n)$ -approximations if

$$\frac{\mu(G)}{\alpha} - \varepsilon n \leq \tilde{\mu}(G) \leq \mu(G) \quad \text{and} \quad \nu(G) \leq \tilde{\nu}(G) \leq \alpha \nu(G) + \varepsilon n.$$

Note that a standard multiplicative  $\alpha$ -approximation is equivalent to an  $(\alpha, 0)$ -approximation.

We will use the following standard version of the Chernoff bound:

**Proposition 2.1** (Chernoff Bound). *Let  $X_1, \dots, X_n$  be independent Bernoulli random variables and define  $X := \sum_{i=1}^n X_i$ . For any  $\lambda > 0$ ,  $\Pr[|X - \mathbf{E}[X]| \geq \lambda] \leq 2 \exp\left(-\frac{\lambda^2}{3\mathbf{E}[X]}\right)$ .*

### 3 Average Query-Complexity of RGMM

Throughout this section, we let  $G = (V, E)$  be a graph with  $n$  vertices,  $m$  edges, and average degree  $\bar{d} = 2m/n$ . We also let  $\Pi$  be the set of all  $m!$  permutations over the edge-set  $E$ .

**Definition 3.1** (Greedy Maximal Matching). *For a permutation  $\pi \in \Pi$ , let  $\text{GMM}(G, \pi)$  be a maximal matching of graph  $G = (V, E)$  obtained by iterating over the edge-set  $E$  in the order of  $\pi$  and greedily adding each encountered edge that is feasible to the matching (an edge is feasible if it is not already incident to a matching edge).*

We are interested in the behavior of the GMM algorithm for a random permutation  $\pi$  chosen uniformly from  $\Pi$ . Particularly, our goal is to determine if a vertex belongs to the produced matching. Of course one can run the algorithm and answer this in  $\Theta(m+n)$  time. But as we will see, better bounds can be achieved via the following local procedure. Similar oracles were analyzed by [23, 26, 24]. But we emphasize that our edge oracle is slightly more efficient than previous ones.

---

**Algorithm 1:** The vertex oracle  $\text{VO}(v, \pi)$ . Determines if vertex  $v$  is matched in  $\text{GMM}(G, \pi)$ .

---

```

1 Let  $e_1 = (v, u_1), \dots, e_k = (v, u_k)$  be the edges incident to  $v$  with  $\pi(e_1) < \dots < \pi(e_k)$ .
2 for  $i$  in  $1 \dots k$  do
3   | if  $\text{EO}(e_i, u_i, \pi) = \text{TRUE}$  then return TRUE
4 return FALSE

```

---



---

**Algorithm 2:** The edge oracle  $\text{EO}(e, u, \pi)$  where  $u$  is an endpoint of edge  $e$ .

---

```

1 if we have already computed  $\text{EO}(e, u, \pi)$  then return the computed answer.
2 Let  $e_1 = (u, w_1), \dots, e_k = (u, w_k)$  be the edges adjacent to  $u$  such that  $\pi(e_i) < \pi(e)$  and
    $\pi(e_1) < \dots < \pi(e_k)$ .
3 for  $i$  in  $1 \dots k$  do
4   | if  $\text{EO}(e_i, w_i, \pi) = \text{TRUE}$  then return FALSE
5 return TRUE

```

---

**Remark 3.2.** *In Line 1 of Algorithm 2 we “cache” previous edge queries. This ensures that the total recursive calls to the edge oracle is bounded by  $\text{poly}(n)$  with probability 1 (see Observation 3.7). This is the only implication of caching that we use in our analysis.*

**Remark 3.3.** *Our edge oracle differs from those of [23, 26, 24] in that the oracle calls generated by  $\text{EO}(e, u, \pi)$  are only for those edges incident to  $u$  and not the other endpoint of  $e$ . This helps in arguing that the stack of recursive calls to the edge oracle form a path in  $G$ .*

**Observation 3.4.** *It holds for the oracles above that:*

- *For any vertex  $v \in V$ ,  $\text{VO}(v, \pi) = \text{TRUE}$  iff  $v$  is matched in  $\text{GMM}(G, \pi)$ .*
- *For any edge  $e = (u, w)$ , if  $\text{EO}(e, u, \pi)$  is (recursively) called while evaluating some vertex oracle  $\text{VO}(v, \pi)$ , then we have  $\text{EO}(e, u, \pi) = \text{TRUE}$  iff  $e \in \text{GMM}(G, \pi)$ .*

*Proof.* The second property immediately implies the first as  $\text{VO}(v, \pi)$  calls the edge oracle on the neighbors of  $v$  until finding a matching edge.

We prove the second property by an induction on  $\pi(e)$ , so assume that this statement holds for all edges with a lower rank than  $\pi(e)$ . Observe that  $\text{EO}(e, u, \pi)$  is either directly called by  $\text{VO}(v, \pi)$  which in that case  $v = w$ , or it is directly called by  $\text{EO}(f, w, \pi)$  for some edge  $f$  incident to  $w$ .



In both cases, by the description of the oracles, the function that calls  $\text{EO}(e, u, \pi)$  should first call  $\text{EO}(e', u', \pi)$  for any edge  $e' = (w, u')$  incident to  $w$  that has a lower rank than  $\pi(e)$ , and that all these calls should have returned **FALSE**. By the induction hypothesis, this implies that no such  $e'$  belongs to  $\text{GMM}(G, \pi)$ . Therefore, if an edge of  $\text{GMM}(G, \pi)$  prevents  $e$  from joining  $\text{GMM}(G, \pi)$ , it must be incident to  $e$  from its other endpoint  $u$ , and should have a lower rank than  $\pi(e)$ . Thus, it is sufficient that the edge oracle  $\text{EO}(e, u, \pi)$  only goes over such edges of  $u$  (for which the statement holds by the induction hypothesis), returning **TRUE** iff none of these calls return **TRUE**.  $\blacksquare$

Let  $T(v, \pi)$  denote the number of recursive calls to the edge oracle  $\text{EO}(\cdot, \cdot, \pi)$  over the course of answering  $\text{VO}(v, \pi)$ . We note that for some edge  $e$ , the edge oracle may be called multiple times during the execution of  $\text{VO}(v, \pi)$  and we count all of these in  $T(v, \pi)$ , though only the first call to  $\text{EO}(e, \pi)$  may generate new recursive calls as the rest of the calls use the cached value.

Our main result in this section is as follows:

**Theorem 3.5.** *For a vertex  $v$  chosen uniformly at random from  $V$  and for a permutation  $\pi$  chosen uniformly at random from  $\Pi$  and independently from  $v$ ,*

$$\mathbf{E}_{v \sim V, \pi \sim \Pi} [T(v, \pi)] = O(\bar{d} \cdot \log n).$$

For any edge  $e \in E$  and a vertex  $v \in V$ , we use  $Q(e, v, \pi)$  to denote the number of times that  $\text{EO}(e, \cdot, \pi)$  is called during the execution of  $\text{VO}(v, \pi)$ , and we define  $Q(e, \pi) := \sum_{v \in V} Q(e, v, \pi)$ . The following immediately follows from the definition:

**Observation 3.6.** *For any  $\pi \in \Pi$  and any  $v \in V$ ,  $T(v, \pi) = \sum_{e \in E} Q(e, v, \pi)$ .*

The next bound holds because we cache query results as discussed in Remark 3.2.

**Observation 3.7.** *For every edge  $e = \{a, b\}$  and every  $\pi \in \Pi$ ,  $Q(e, \pi) \leq O(n^2)$ .*

*Proof.* Take an arbitrary vertex  $v \in V$ . Over the course of answering  $\text{VO}(v, \pi)$ , we either call  $\text{EO}(e, \cdot, \pi)$  directly by the vertex oracle (at most once) or during the execution of  $\text{EO}(f, \cdot, \pi)$  for some edge  $f$  incident to  $e$ . The key observation is that  $\text{EO}(f, \cdot, \pi)$  generates new recursive calls only the first time that it is called (due to caching). Hence, in total  $\text{EO}(e, \cdot, \pi)$  is called at most  $(\deg_G(a) - 1) + (\deg_G(b) - 1) + 1 \leq 2n - 1$  times while answering  $\text{VO}(v, \pi)$ . This means  $Q(e, v, \pi) \leq 2n - 1$ . Since  $Q(e, \pi) = \sum_v Q(e, v, \pi)$ , we get  $Q(e, \pi) \leq \sum_v (2n - 1) = n(2n - 1) = O(n^2)$ .  $\blacksquare$

The main technical part is the proof of Lemma 3.8 below. Intuitively, the lemma states that if we run the vertex oracle on every vertex  $v \in V$  all in parallel (in a way that the cached values stored for one parallel call are not used for another), then in expectation over  $\pi$ , the total number of times that edge oracle  $\text{EO}(e, \cdot, \pi)$  is called can be bounded by  $O(\log n)$ .

**Lemma 3.8.** *For any edge  $e$ ,  $\mathbf{E}_\pi [Q(e, \pi)] = O(\log n)$ .*

Lemma 3.8 easily implies Theorem 3.5 as described next.

*Proof of Theorem 3.5 via Lemma 3.8.* It holds that

$$\begin{aligned} \mathbf{E}_\pi \left[ \sum_{v \in V} T(v, \pi) \right] &\stackrel{\text{Obs 3.6}}{=} \mathbf{E}_\pi \left[ \sum_{v \in V} \sum_{e \in E} Q(e, v, \pi) \right] = \mathbf{E}_\pi \left[ \sum_{e \in E} \sum_{v \in V} Q(e, v, \pi) \right] = \mathbf{E}_\pi \left[ \sum_{e \in E} Q(e, \pi) \right] \\ &= \sum_{e \in E} \mathbf{E}_\pi [Q(e, \pi)] \stackrel{\text{Lemma 3.8}}{=} \sum_{e \in E} O(\log n) = O(m \log n). \end{aligned} \quad (1)$$

Therefore, for a vertex  $v \in V$  that is chosen uniformly at random from  $V$ , we get

$$\mathbf{E}_{v,\pi}[T(v,\pi)] = \frac{1}{n} \cdot \mathbf{E}_\pi \left[ \sum_{v \in V} T(v,\pi) \right] \stackrel{(1)}{=} \frac{1}{n} \cdot O(m \log n) = O(\bar{d} \cdot \log n). \quad \blacksquare$$

The rest of this section is devoted to proving Lemma 3.8.

**Query paths:** Let  $S_{v,\pi}$  be the stack of recursive calls to the edge oracle during the execution of  $\text{VO}(v,\pi)$ . Namely, whenever the edge oracle  $\text{EO}(e,u,\pi)$  is called on some edge  $e$  we push  $e$  to the stack, and pop  $e$  when the value of  $\text{EO}(e,u,\pi)$  is determined. Let  $P = (e_1, \dots, e_k)$  be the ordered set of edges inside  $S_{v,\pi}$  at some point during the execution of  $\text{VO}(v,\pi)$  with  $e_k$  being the last edge pushed into the stack. One can verify from the description of the vertex and edge oracles that  $P$  must be a path in graph  $G$  with  $v$  being one of its endpoints (particularly  $v \in e_1$ ). Now if we make the edges in  $P$  directed such that  $\vec{P} = (\vec{e}_1, \dots, \vec{e}_k)$  is a directed path starting from  $v$ , we call  $\vec{P}$  a  $(v,\pi)$ -query-path. With this definition,  $T(v,\pi)$  is essentially the total number of  $(v,\pi)$ -query-paths.

Let us take an edge  $e = \{a,b\} \in E$  and let  $\vec{e} = (a,b)$  be the same edge made directed from  $a$  to  $b$ . We define  $\mathcal{Q}(\vec{e},v,\pi)$  to be the set of all  $(v,\pi)$ -query-paths that end at  $\vec{e}$  (in this precise direction) and define  $\mathcal{Q}(\vec{e},\pi) := \bigcup_{v \in V} \mathcal{Q}(\vec{e},v,\pi)$ . Furthermore, we define  $Q(\vec{e},v,\pi) := |\mathcal{Q}(\vec{e},v,\pi)|$  and define  $Q(\vec{e},\pi) := \sum_{v \in V} Q(\vec{e},v,\pi) = |\mathcal{Q}(\vec{e},\pi)|$  (the latter equality follows since  $\mathcal{Q}(\vec{e},v,\pi)$  and  $\mathcal{Q}(\vec{e},u,\pi)$  are disjoint for  $u \neq v$ ).

**Observation 3.9.** *Let  $e = \{a,b\}$ ,  $\vec{e} = (a,b)$ , and  $\bar{e} = (b,a)$ . We have  $Q(e,\pi) = Q(\vec{e},\pi) + Q(\bar{e},\pi)$ .*

*Proof.* Recall from the definition that  $Q(e,v,\pi)$  is the number of times that edge oracle  $\text{EO}(e,\cdot,\pi)$  is called during the execution of  $\text{VO}(v,\pi)$ . Every time that we call  $\text{EO}(e,\cdot,\pi)$ , we add  $e$  to the stack  $S_{v,\pi}$  and thus get exactly one  $(v,\pi)$ -query-path that ends at  $e$  either in the direction of  $\vec{e}$  or  $\bar{e}$ . This implies that  $Q(e,v,\pi) = Q(\vec{e},v,\pi) + Q(\bar{e},v,\pi)$ . The observation follows since by definition  $Q(\vec{e},\pi) = \sum_v Q(\vec{e},v,\pi)$ ,  $Q(e,\pi) = \sum_v Q(e,v,\pi)$ , and  $Q(\bar{e},\pi) = \sum_v Q(\bar{e},v,\pi)$ .  $\blacksquare$

In what follows, we prove the following lemma which easily implies Lemma 3.8.

**Lemma 3.10.** *For any arbitrarily directed edge  $\vec{e} = (a,b)$ ,  $\mathbf{E}_\pi[Q(\vec{e},\pi)] = O(\log n)$ .*

*Proof of Lemma 3.8 via Lemma 3.10.* By Observation 3.9, Lemma 3.10 implies Lemma 3.8 since for any edge  $e = \{u,v\}$ ,  $\mathbf{E}_\pi[Q(e,\pi)] = \mathbf{E}_\pi[Q(\vec{e},\pi)] + \mathbf{E}_\pi[Q(\bar{e},\pi)] = O(\log n) + O(\log n) = O(\log n)$ .  $\blacksquare$

We now turn to prove Lemma 3.10 for edge  $\vec{e}$  that is fixed for the rest of the proof.

First, given a permutation  $\pi \in \Pi$  and a directed path  $\vec{P} = (\vec{e}_1, \dots, \vec{e}_k)$ , we define  $\phi(\pi, \vec{P}) \in \Pi$  to be another permutation  $\sigma \in \Pi$  constructed as:

$$\begin{aligned} (\sigma(e_1), \dots, \sigma(e_{k-1}), \sigma(e_k)) &:= (\pi(e_2), \dots, \pi(e_k), \pi(e_1)), \text{ and} \\ \sigma(e') &:= \pi(e') \quad \forall e' \notin \vec{P}. \end{aligned}$$

That is,  $\phi(\pi, \vec{P})$  is obtained by rotating the ranks of  $\pi$  along  $\vec{P}$  in the reverse direction.

Now we construct a bipartite graph  $H = H(\vec{e})$  with two parts  $A$  and  $B$  such that  $|A| = |B| = |\Pi|$ . Each permutation  $\pi \in \Pi$  over the edge-set of  $G$  has one corresponding vertex  $\pi_A$  in  $A$  and one corresponding vertex  $\pi_B$  in  $B$ . Furthermore, for any permutation  $\pi \in \Pi$  and any query-path  $\vec{P} = (\vec{e}_1, \dots, \vec{e}_k = \vec{e}) \in \mathcal{Q}(\vec{e},\pi)$ , we connect vertex  $\pi_A \in A$  to vertex  $\sigma_B \in B$  corresponding to permutation  $\sigma := \phi(\pi, \vec{P})$ .

Graph  $H$  is particularly constructed such that for each permutation  $\pi \in \Pi$  the degree of vertex  $\pi_A \in A$  equals  $Q(\vec{e},\pi)$ . Namely:

**Observation 3.11.** For any permutation  $\pi \in \Pi$ ,  $\deg_H(\pi_A) = Q(\vec{e}, \pi)$ .

*Proof.* By construction there is a one-to-one mapping between the edges of  $\pi_A$  and query paths  $\vec{P} \in Q(\vec{e}, \pi)$ . Thus  $\deg_H(\pi_A) = |Q(\vec{e}, \pi)| = Q(\vec{e}, \pi)$ . ■

Therefore to prove Lemma 3.10 that  $\mathbf{E}_\pi[Q(\vec{e}, \pi)] = O(\log n)$ , it suffices to bound the average degree of graph  $H$  by  $O(\log n)$ . In other words, it suffices to show that for a vertex  $\pi_A \in A$  chosen uniformly at random,  $\mathbf{E}_{\pi_A \sim A}[\deg_H(\pi_A)] = O(\log n)$ . This is our plan for the rest of the proof.

For some large enough constant  $c \geq 1$  and parameter  $\beta = c \log n$ , we partition  $\Pi$  into two subsets of *likely* permutations  $L$  and *unlikely* permutations  $U$  as follows:

$$L := \left\{ \pi \in \Pi \mid \max_{P \in Q(\vec{e}, \pi)} |P| \leq \beta \right\}, \quad U := \Pi \setminus L. \quad (2)$$

In words, if all query-paths in  $Q(\vec{e}, \pi)$  have length  $\leq \beta$  then  $\pi \in L$ , and otherwise  $\pi \in U$ . We use  $A_L$  (resp.  $A_U$ ) to denote the set of vertices in part  $A$  of graph  $H$  that correspond to permutations in  $L$  (resp.  $U$ ).

We use the next two lemmas to bound the number of edges connected to  $A_U$  and  $A_L$  respectively.

**Lemma 3.12.** Any vertex  $\sigma_B \in B$  has at most  $\beta$  neighbors in  $A_L$ .

**Lemma 3.13.** If constant  $c$  is large enough,  $|A_U| \leq m!/n^2$ .

Let us first see how Lemmas 3.12 and 3.13 prove Lemma 3.10 (and thus Theorem 3.5 as discussed before). We then turn to prove these two claims.

*Proof of Lemma 3.10 via Lemmas 3.12 and 3.13.* We first upper bound the size of the edge-set  $E(H)$  of  $H$  by  $O(m! \log n)$ . Since each vertex  $\pi_A \in A$  has degree  $O(n^2)$  by Observation 3.7, and that by Lemma 3.13,  $|A_U| \leq m!/n^2$ , the number of edges connected to  $A_U$  is  $\frac{m!}{n^2} \cdot O(n^2) = O(m!)$ . Moreover, by Lemma 3.12, each vertex  $\sigma_B \in B$  has at most  $\beta$  neighbors in  $A_L$ . Since  $H$  is bipartite and every edge of  $A_L$  goes to  $B$ , the total number of edges connected to  $A_L$  can be upper bounded by  $|B| \cdot \beta = m! \cdot c \log n = O(m! \log n)$ . Since  $A_L$  and  $A_U$  partition  $A$  and that every edge of  $H$  has one endpoint in  $A$ , we get  $|E(H)| = O(m! \log n) + O(m!) = O(m! \log n)$ .

Now if we pick a vertex  $\pi_A$  from  $A$ , it has expected degree  $\frac{|E(H)|}{|A|} = \frac{O(m! \log n)}{m!} = O(\log n)$ . By Observation 3.11, this implies  $\mathbf{E}_{\pi \in \Pi}[Q(\vec{e}, \pi)] = \mathbf{E}_{\pi_A \sim A}[\deg_H(\pi_A)] = O(\log n)$ . ■

### 3.1 Proof of Lemma 3.12

We prove the following statement which we show suffices to prove Lemma 3.12.

**Claim 3.14.** Let  $\pi, \pi' \in \Pi$  with  $\pi \neq \pi'$ , let  $\vec{P}$  and  $\vec{P}'$  respectively be  $(v, \pi)$ - and  $(v', \pi')$ -query-paths both ending at  $\vec{e}$  for some  $v, v' \in V$ , and suppose that  $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}') = \sigma$ . Then  $|\vec{P}| \neq |\vec{P}'|$ .

Let us first see how Claim 3.14 suffices to prove Lemma 3.12:

*Proof of Lemma 3.12 via Claim 3.14.* Let us take an arbitrary vertex  $\sigma_B$  in part  $B$  of graph  $H$ . For any edge  $\{\pi_A, \sigma_B\}$  in  $H$ , by construction of  $H$  there must be a unique  $(v, \pi)$ -query-path  $\vec{P}$  such that  $\phi(\pi, \vec{P}) = \sigma$  where here  $\pi, \sigma \in \Pi$  are the permutations corresponding to  $\pi_A$  and  $\sigma_B$  respectively. Let us define the label  $\chi(\pi_A, \sigma_B)$  of edge  $\{\pi_A, \sigma_B\}$  to be the length  $|\vec{P}|$  of this query-path  $\vec{P}$ . Claim 3.14 essentially implies that all the edges of  $\sigma_B$  receive different labels. On the other hand, if  $\pi_A$  is a neighbor of  $\sigma_B$  and  $\pi_A \in A_L$ , then by definition (2) of set  $L$ ,  $\chi(\pi_A, \sigma_B) \leq \beta$ . The uniqueness of the integer labels and the upper bound of  $\beta$  imply together that  $\sigma_B$  can have at most  $\beta$  neighbors in  $A_L$ . The proof of Lemma 3.12 is thus complete. ■

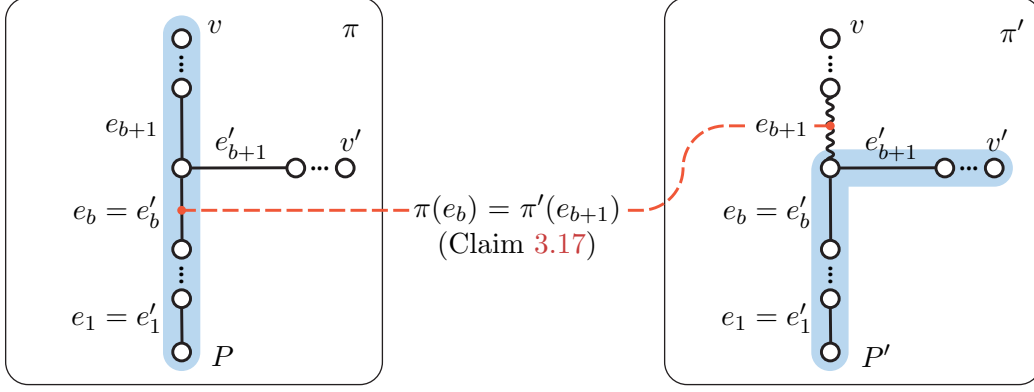


Figure 1: On the right hand side, we have permutation  $\pi'$  and query-path  $P'$  is highlighted. On the left hand side, we have permutation  $\pi$  and path  $P$  is highlighted. Our arguments essentially show that  $e_{b+1}$  must belong to matching  $\text{GMM}(G, \pi')$ . We also show that  $\pi'(e_{b+1}) < \pi'(e'_b)$ . Therefore,  $\text{EO}(e'_{b+1}, \cdot, \pi')$  should terminate (returning FALSE) before calling  $\text{EO}(e'_b, \cdot, \pi')$ . This means  $P'$  cannot be a valid query-path in  $\pi'$  and this is our desired contradiction which proves Claim 3.14.

We now turn to prove Claim 3.14.

Let  $\pi, \pi', \vec{P}, \vec{P}', v, v', \sigma$  be as defined in Claim 3.14 and assume for contradiction that  $|\vec{P}| = |\vec{P}'|$ . First observe that if in addition to their lengths, the edges traversed by the two paths  $\vec{P}, \vec{P}'$  are also the same i.e.,  $\vec{P} = \vec{P}'$ , then by definition of the mapping  $\phi$ , we have  $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}')$  iff  $\pi = \pi'$  which contradicts the assumption  $\pi \neq \pi'$  of Claim 3.14. So let us assume that  $\vec{P} \neq \vec{P}'$  but  $|\vec{P}| = |\vec{P}'|$ . This implies that the two paths must “branch” at least once.

It would be convenient to define  $\vec{P} = (\vec{e}_1, \vec{e}_2, \dots, \vec{e}_k)$  and  $\vec{P}' = (\vec{e}'_1, \vec{e}'_2, \dots, \vec{e}'_k)$  to be respectively the same as  $\vec{P} = (\vec{e}_k, \dots, \vec{e}_1)$  and  $\vec{P}' = (\vec{e}'_k, \dots, \vec{e}'_1)$  except that their edges are traversed in the reverse direction. Since both  $\vec{P}$  and  $\vec{P}'$  end at  $\vec{e}$  (by definition of query-paths), both  $\vec{P}$  and  $\vec{P}'$  must start from  $\vec{e} = \vec{e}_1 = \vec{e}'_1$ . Let  $(b+1)$  be the smallest index where  $\vec{e}_{b+1} \neq \vec{e}'_{b+1}$ . That is, we have  $e_1 = e'_1, \dots, e_b = e'_b$  and  $e_{b+1} \neq e'_{b+1}$  (see Figure 1). Observe that  $b+1 \geq 2$  since  $\vec{e}_1 = \vec{e}'_1 = \vec{e}$ .

Let us make the following assumption that comes without loss of generality as we have not distinguished  $\pi$  and  $\pi'$  in any other way up to this point of the analysis.

**Assumption 3.15.**  $\pi(e_b) \leq \pi'(e'_b)$ .

**Observation 3.16.** *It holds that  $\pi(e_1) < \pi(e_2) < \dots < \pi(e_k)$  and  $\pi'(e'_1) < \pi'(e'_2) < \dots < \pi'(e'_k)$ .*

*Proof.* This holds because  $\vec{P}$  is a query-path in  $\pi$  and  $\vec{P}'$  is a query-path in  $\pi'$ . Recall that query-paths correspond to recursions in the stack and Algorithm 2 only recurses on edges with lower rank than the current edge. Hence the edges along a query path must be decreasing in rank. ■

**Claim 3.17.** *It holds that  $\pi(e_b) = \pi'(e_{b+1})$ .*

*Proof.* First, since  $b+1 \geq 2$ , we have  $e_{b+1} \neq e_1$ . Therefore by definition of  $\phi(\pi, \vec{P})$ , which rotates the ranks of  $\pi$  in the reverse direction of  $\vec{P}$  (i.e., in direction of  $\vec{P}$ ),  $\phi(\pi, \vec{P})(e_{b+1}) = \pi(e_b)$ . On the other hand, note that  $e_{b+1} \notin P'$  since  $\vec{P}$  and  $\vec{P}'$  branch after edge  $e_b = e'_b$  and they are paths — this implies that  $\phi(\pi', \vec{P}')(e_{b+1}) = \pi'(e_{b+1})$ . Combined with our assumption that  $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}') = \sigma$ , this implies the desired equality  $\pi'(e_{b+1}) = \pi(e_b)$ . ■

**Claim 3.18.**  $\pi'(e_{b+1}) < \pi'(e'_b)$ .

*Proof.* From Claim 3.17 we know  $\pi'(e_{b+1}) = \pi(e_b)$ . Combined with Assumption 3.15 that  $\pi(e_b) \leq \pi'(e'_b)$ , this implies  $\pi'(e_{b+1}) \leq \pi'(e'_b)$ . The equality can be ruled out since  $\pi'$  is a permutation and distinct edges  $e_{b+1}$  and  $e'_b$  cannot be assigned the same rank. As such,  $\pi'(e_{b+1}) < \pi'(e'_b)$ . ■

**Claim 3.19.** *If  $\pi(f) \neq \pi'(f)$  for some edge  $f$ , then  $\pi(f) \geq \pi(e_b)$  and  $\pi'(f) \geq \pi(e_b)$ . In other words, the two permutations  $\pi$  and  $\pi'$  are identical on all ranks smaller than  $\pi(e_b)$ .*

*Proof.* If  $f \notin P \cup P'$ , then clearly  $\pi'(f) = \pi(f) = \sigma(f)$  since  $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}') = \sigma$  and  $\phi$  only changes the ranks of edges along the query-path that it is given. So we can assume  $f \in P \cup P'$ ; there are therefore four possible scenarios:

**Case (1)**  $f \in \{e_1, \dots, e_{b-1}\}$ : For any  $i \in \{2, \dots, b\}$ , to have  $\phi(\pi, \vec{P})(e_i) = \phi(\pi', \vec{P}')(e_i)$ , it is necessary that  $\pi(e_{i-1}) = \pi'(e_{i-1})$ . Thus, in this case  $\pi(f) = \pi'(f)$ .

**Case (2)**  $f = e_b$ : We have  $\pi(f) = \pi(e_b)$  since  $f = e_b$  and we have  $\pi'(f) = \pi'(e_b) \geq \pi(e_b)$  by Assumption 3.15. So the claim holds in this case.

**Case (3)**  $f = e_i$  for some  $e_i \in \{e_{b+1}, \dots, e_k\}$ : Observation 3.16 already implies  $\pi(f) > \pi(e_b)$  in this case; it remains to prove  $\pi'(f) \geq \pi(e_b)$ . First,  $\phi(\pi, \vec{P})(f) = \pi(e_{i-1}) \geq \pi(e_b)$  by construction of  $\phi$  (as  $i \neq 1$ ) and Observation 3.16 (as  $i - 1 \geq b$ ). Therefore, from  $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}')$  (as assumed in Claim 3.14) we get  $\phi(\pi', \vec{P}')(f) \geq \pi(e_b)$ . Given that  $\phi(\pi', \vec{P}')(e) \leq \pi'(e)$  for every edge  $e \neq e_1$  by construction of  $\phi$  and monotonicity of  $\pi'$  along  $P'$  (Observation 3.16) this means  $\pi'(f) \geq \pi(e_b)$ .

**Case (4)**  $f = e'_i$  for some  $e'_i \in \{e'_{b+1}, \dots, e'_k\}$ : Observation 3.16 and Assumption 3.15 together imply  $\pi'(f) > \pi'(e'_b) \geq \pi(e_b)$  in this case; it remains to prove  $\pi(f) \geq \pi(e_b)$ . First,  $\phi(\pi', \vec{P}')(f) = \pi'(e'_{i-1}) \geq \pi'(e'_b)$  by construction of  $\phi$  (as  $i \neq 1$ ) and Observation 3.16 (as  $i - 1 \geq b$ ). Thus, from  $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}')$  (as assumed in Claim 3.14) we get  $\phi(\pi, \vec{P})(f) \geq \pi'(e'_b)$ . Given that  $\phi(\pi, \vec{P})(e) \leq \pi(e)$  for every edge  $e \neq e_1$  by construction of  $\phi$  and monotonicity of  $\pi$  along  $P$  (Observation 3.16) this means  $\pi(f) \geq \pi'(e'_b) \geq \pi(e_b)$  where the last inequality follows from Assumption 3.15. ■

**Claim 3.20.**  $e_{b+1} \in \text{GMM}(G, \pi')$ .

*Proof.* Assume for the sake of contradiction that  $e_{b+1} \notin \text{GMM}(G, \pi')$ . This means that  $e_{b+1}$  must be incident to an edge  $f \in \text{GMM}(G, \pi')$  with  $\pi'(f) < \pi'(e_{b+1}) = \pi(e_b)$  (where recall the last equality follows from Claim 3.17). Since  $\pi$  and  $\pi'$  are identical for ranks smaller than  $\pi(e_b)$  by Claim 3.19,  $f \in \text{GMM}(G, \pi')$  implies  $f \in \text{GMM}(G, \pi)$  as well. Using this and combined with  $\pi(f) = \pi'(f) < \pi(e_b)$ , we show that  $\vec{P}$  is not a valid query-path in permutation  $\pi$  which is a contradiction. To see this, note that while running the edge oracle  $\text{EO}(e_{b+1}, \cdot, \pi)$ , we should call  $\text{EO}(f, \cdot, \pi)$  before  $\text{EO}(e_b, \cdot, \pi)$  because  $\pi(f) < \pi(e_b)$  and that  $f$  and  $e_b$  share the same endpoint with  $e_{b+1}$ ; moreover, since  $f \in \text{GMM}(G, \pi)$ ,  $\text{EO}(e_{b+1}, \cdot, \pi)$  will immediately terminate and return FALSE without calling  $\text{EO}(e_b, \cdot, \pi)$ . ■

We are now ready to finalize the proof of Claim 3.14 by showing that the assumptions above lead to a contradiction. The contradiction that we prove is that  $\vec{P}'$  cannot be a valid query-path in permutation  $\pi'$ . To see this, recall that during the execution of  $\text{EO}(e'_{b+1}, \cdot, \pi')$ , we have to call  $\text{EO}(e_{b+1}, \cdot, \pi')$  before  $\text{EO}(e'_b, \cdot, \pi')$  since by Claim 3.18  $\pi'(e_{b+1}) < \pi'(e'_b)$ . On the other hand, by Claim 3.20 the answer to  $\text{EO}(e_{b+1}, \cdot, \pi')$  is TRUE and so  $\text{EO}(e'_{b+1}, \cdot, \pi')$  should terminate immediately and return FALSE without calling  $\text{EO}(e'_b, \cdot, \pi')$ . Therefore,  $\vec{P}'$  is not a valid query-path in  $\pi'$  contradicting our assumption that  $|\vec{P}| = |\vec{P}'|$  is possible. The proof of Claim 3.14 is thus complete. As discussed at the start of Section 3.1, this also completes the proof of Lemma 3.12.

### 3.2 Proof of Lemma 3.13

To prove Lemma 3.13 we first recall a parallel implementation of the randomized greedy maximal matching algorithm and the bounds known for its *round-complexity*. For more details see [5, 16].

**Parallel Randomized Greedy MM:** Given a graph  $G$  and a permutation  $\pi$  over its edge-set, we repeat the following until  $G$  becomes empty: in parallel add any “local minimum” edge  $e$  to the matching and remove its endpoints from the graph. An edge is local minimum if its rank is smaller than that of all of its neighboring edges that remain in the graph.

It can be easily confirmed that the output of the algorithm above is exactly  $\text{GMM}(G, \pi)$ . We use  $\rho(G, \pi)$  to denote the round-complexity of the algorithm, i.e., the number of iterations that it takes until the graph becomes empty.

It was shown in [5] that  $\rho(G, \pi)$  can be bounded for a random permutation by  $O(\log^2 n)$  with high probability. This was improved to  $O(\log n)$  in [16]:

**Lemma 3.21** ([16]). *Let  $\pi$  be a permutation chosen uniformly at random over the edge-set of an  $n$ -vertex graph  $G$ . With probability at least  $1 - n^{-2}$ ,  $\rho(G, \pi) = O(\log n)$ .*

We prove the following:

**Claim 3.22.** *Let  $P$  be any query-path in  $G$  for permutation  $\pi$ , then  $\rho(G, \pi) \geq \lfloor \frac{|P|}{2} \rfloor$ .*

Claim 3.22 suffices to prove Lemma 3.13 as proved next.

*Proof of Lemma 3.13 via Claim 3.22.* For any permutation  $\pi \in U$ , by definition (2) there is at least one query-path of length at least  $\beta + 1$  in permutation  $\pi$ . This implies by Claim 3.22 that  $\rho(G, \pi) \geq \lfloor \frac{\beta+1}{2} \rfloor$  for any  $\pi \in U$ . If we set the constant  $c$  in  $\beta = c \log n$  to be sufficiently large, we can use Lemma 3.21 to bound the probability of this event by  $\leq 1/n^2$ . Thus,  $|U|/|\Pi| \leq 1/n^2$  which implies  $|U| \leq |\Pi|/n^2 = m!/n^2$ . The lemma follows noting that  $|A_U| = |U|$  due to the one-to-one correspondence between vertices  $A_U$  and permutations in  $U$  that we discussed in Section 3.1. ■

*Proof of Claim 3.22.* Let  $P = (e_k, \dots, e_1)$  be our query-path and recall from Observation 3.16 that  $\pi(e_k) > \dots > \pi(e_1)$ . For any edge  $e$ , we use  $\rho(e)$  to denote the round of the parallel implementation in which edge  $e$  gets removed from the graph according to permutation  $\pi$ . We show that for any  $i \in \{2, \dots, k-1\}$ ,  $\rho(e_i) \geq \rho(e_{i-2}) + 1$ . By a simple induction, this implies  $\rho(e_{k-1}) \geq \lfloor k/2 \rfloor$  and so  $\rho(G, \pi) \geq \lfloor k/2 \rfloor$ .

Suppose for the sake of contradiction that  $\rho(e_i) < \rho(e_{i-2}) + 1$  (or equivalently  $\rho(e_i) \leq \rho(e_{i-2})$ ) for some  $2 \leq i \leq k-1$ . This means that at the start of round  $\rho(e_i)$  both  $e_i$  and  $e_{i-2}$  are still in the graph. Observe that if  $\rho(e_{i-1}) < \rho(e_i)$ , then during round  $\rho(e_{i-1})$  at least one of the endpoints of  $e_{i-1}$  must be matched and so either  $e_i$  or  $e_{i-2}$  (or both) should also be removed from the graph which contradicts  $\rho(e_{i-1}) < \rho(e_i) \leq \rho(e_{i-2})$ . Therefore  $\rho(e_{i-1}) \geq \rho(e_i)$  and so  $e_{i-1}$  is also still in the graph along with  $e_i$  and  $e_{i-2}$  at the start of round  $\rho(e_i)$ . This means that  $e_i$  is not a local minimum during round  $\rho(e_i)$  and so it is removed in this round because some other edge  $f \neq e_i$  joins the matching. Note also that  $f \neq e_{i-1}$  since  $e_{i-1}$  is incident to  $e_{i-2}$  and is not a local minimum.

Let  $x$  be the vertex incident to both  $e_{i-1}$  and  $e_i$  and let  $y$  be the other endpoint of  $e_i$  incident to  $e_{i+1}$  (note that since  $i \leq k-1$  edge  $e_{i+1}$  should exist). Since  $f$  is a neighbor of  $e_i$ , it is either connected to  $x$  or  $y$ . We show that both cases lead to contradictions.

**Case (1)**  $f$  connected to  $y$ : Noting that  $\pi(f) < \pi(e_i)$  since  $\pi(f)$  is a local minimum when  $e_i$  still is in the graph, we get that  $\text{EO}(e_{i+1}, \cdot, \pi)$  calls  $\text{EO}(f, \cdot, \pi)$  before  $\text{EO}(e_i, \cdot, \pi)$ . But because  $f \in \text{GMM}(G, \pi)$ ,  $\text{EO}(e_{i+1}, \cdot, \pi)$  would not continue to call  $\text{EO}(e_i, \cdot, \pi)$  and  $P$  cannot be a valid

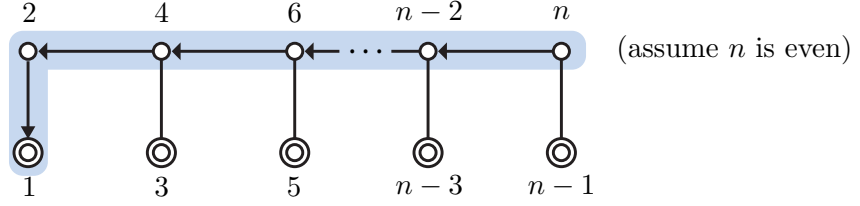


Figure 2: In this permutation, all the vertices with odd ranks join the MIS in round one and the whole graph becomes empty immediately, hence the parallel depth is 1. However, the query process for the vertex with rank  $n$  first goes through all even nodes, thus has length  $\geq n/2$ .

query-path.

**Case (2)**  $f$  connected to  $x$ : Since  $f$  is a local minimum when  $e_{i-1}$  still exists in the graph,  $\pi(f) < \pi(e_{i-1})$ . This implies that  $\text{EO}(e_i, \cdot, \pi)$  should call  $\text{EO}(f, \cdot, \pi)$  before  $\text{EO}(e_{i-1}, \cdot, \pi)$ . But because  $f \in \text{GMM}(G, \pi)$ ,  $\text{EO}(e_i, \cdot, \pi)$  would not continue to call  $\text{EO}(e_{i-1}, \cdot, \pi)$  and so, again,  $P$  cannot be a valid query-path.

The contradictions above rule out the possibility of our assumption that  $\rho(e_i) < \rho(e_{i-2}) + 1$  and so we indeed get  $\rho(e_i) \geq \rho(e_{i-2}) + 1$  for all  $2 \leq i \leq k-1$ . As discussed, this implies  $\rho(G, \pi) \geq \lfloor k/2 \rfloor$  completing the proof. ■

**Remark 3.23.** *Claim 3.22 shows that for any permutation  $\pi$ , the maximum query length in the random greedy maximal matching algorithm is asymptotically upper bounded by the parallel round-complexity of this algorithm for the same permutation. One may wonder if this also holds for the randomized greedy maximal independent set (MIS) algorithm which processes the vertices in a random order and adds each encountered feasible vertex to the independent set greedily (see [26, 5]). Interestingly, the answer turns out to be negative. See Figure 2.*

## 4 The Final Algorithms for the Adjacency List Query Model

In this section, we show how the oracle analysis of Section 3 can lead to our claimed bounds of Theorems 1.2 and 1.3 in the adjacency list query model.

As before, let  $G = (V, E)$  be an arbitrary graph with  $n$  vertices,  $m$  edges, maximum degree  $\Delta$ , and average degree  $\bar{d}$ . Having defined and analyzed the oracle calls of the GMM algorithm in Section 3, we now employ the standard recipe of the literature in estimating the size of MCM or MVC. We sample a number of random vertices and simulate the vertex oracle of Algorithm 1 on each. If our sample size is sufficiently large, the fraction of matched sampled vertices is a good estimate of the fraction of vertices in the graph that are matched by GMM. To formalize this, we first show how to simulate the vertex and edge oracles of Section 3 using adjacency list queries.

First, to generate the random permutation  $\pi$ , one can for each edge  $e$  sample an independent rank  $\sigma(e)$  which is a real in  $[0, 1]$  chosen uniformly at random, and then obtain  $\pi$  by sorting the edges in the increasing order of their ranks. This way we can expose the random permutation “on the fly” only where it is needed, avoiding the  $\Omega(m)$  time needed for generating it for all the edges. Another challenge remains though. A trivial simulation of the edge oracle  $\text{EO}(e, \pi)$  (Algorithm 2) is to generate the rank  $\sigma(e')$  of all edges  $e'$  incident to  $e$  upon calling  $\text{EO}(e, \pi)$ . The problem with this approach is that the total number of queries in answering  $\text{VO}(v, \pi)$  can be as large as  $O(T(v, \pi)\Delta)$  whereas we need a bound of  $\tilde{O}(T(v, \pi))$  for our final results. Here  $T(v, \pi)$  as defined in Section 3 is

the number of edges on which the edge oracle is recursively called during the execution of  $\text{VO}(v, \pi)$  and the  $O(\Delta)$  factor comes from querying up to  $O(\Delta)$  neighbors of each such edge. To get rid of this  $\Delta$  factor, the idea is to expose the neighbors of each edge in “batches,” only when they are needed. This leads to the following bound, a variant of which was first proved by [24]:

**Lemma 4.1** ([24]). *Let  $v$  be an arbitrary vertex in a graph  $G = (V, E)$ . There is an algorithm that draws a random permutation  $\pi$  over  $E$ , and determines whether  $v$  is matched in  $\text{GMM}(G, \pi)$  in time  $\tilde{O}(T(v, \pi) + 1)$  having query access to the adjacency lists. The algorithm succeeds w.h.p.*

We note that Lemma 4.1 is slightly stronger than its variant proved in [24] where the produced answers were only approximately close, in total variation distance, to the actual distribution. We observe that this can be turned to an exact guarantee by using the exact sublinear time binomial samplers of [14, 6]. For completeness, we provide the full proof of Lemma 4.1 in Appendix A.

#### 4.1 Proof of Theorem 1.2: Multiplicative Approximation

We assume, w.l.o.g., in this section that the graph has no singleton vertices, and that the average degree  $\bar{d}$  and maximum degree  $\Delta$  are given. Note that we can simply query the degree of every vertex in the graph, discard all the singleton vertices, and compute  $\bar{d}$  and  $\Delta$  for the rest of the vertices in  $O(n)$  time and queries as allowed by Theorem 1.2. Hence, the assumption comes w.l.o.g. Having this assumption, the rest of the algorithm of this section runs in  $\tilde{O}(\Delta/\varepsilon^2)$  time.

As discussed, the general idea is to take  $k$  random vertices and run the greedy oracle of Lemma 4.1 on them to see what fraction of them get matched. For the guarantee of Theorem 1.2, it turns out that setting  $k = \tilde{\Theta}(\Delta/\bar{d}\varepsilon^2)$  suffices to see sufficiently many matched vertices. The following simple claim plays a crucial role in arguing that these many samples suffice for our purpose:

**Claim 4.2.** *For any  $n$ -vertex graph  $G$  of maximum degree  $\Delta$  and average degree  $\bar{d}$ ,  $\mu(G) \geq \frac{n\bar{d}}{4\Delta}$ .*

*Proof.* By Vizing’s theorem, any graph  $G$  of maximum degree  $\Delta$  has a proper  $(\Delta + 1)$ -edge-coloring. Since the  $m$  edges are colored only via  $(\Delta + 1)$  colors, there must be a color that is assigned to  $\geq m/(\Delta + 1)$  edges. Since the edges of any color form a matching, the graph must have a matching of size  $\geq m/(\Delta + 1)$ . Noting that  $\bar{d} = 2m/n$ , we get:

$$\mu(G) \geq \frac{m}{\Delta + 1} = \frac{n\bar{d}/2}{\Delta + 1} \geq \frac{n\bar{d}}{4\Delta}. \quad \blacksquare$$

Our starting point is the following Algorithm 3:

---

**Algorithm 3:** An algorithm used for Theorem 1.2, given parameter  $\varepsilon > 0$ .

---

- 1  $k \leftarrow 128 \cdot 24(\Delta \ln n)/(\varepsilon^2 \bar{d})$ . // Note that  $\bar{d}$  and  $\Delta$  are known to the algorithm.
  - 2 Sample  $k$  vertices  $v_1, \dots, v_k$  (with replacement) independently and uniformly from  $V$ .
  - 3 For each  $i \in [k]$  run the algorithm of Lemma 4.1 on vertex  $v_i$ . For each  $i \in [k]$  let  $X_i$  be the indicator of the event that  $v_i$  is matched once we run Lemma 4.1 for  $v_i$ .
  - 4 Let  $X \leftarrow \sum_{i=1}^k X_i$  and let  $f \leftarrow X/k$  be the fraction of vertices  $v_1, \dots, v_k$  that get matched.
  - 5 Let  $\tilde{\mu} \leftarrow (1 - \frac{\varepsilon}{2})fn/2$  and let  $\tilde{\nu} \leftarrow (1 + \frac{\varepsilon}{2})fn$ .
  - 6 **return**  $\tilde{\mu}$  as the estimate for  $\mu(G)$  and **return**  $\tilde{\nu}$  as the estimate for  $\nu(G)$ .
- 

We start by analyzing the approximation ratio of Algorithm 3:

**Lemma 4.3.** *Let  $\tilde{\mu}$  and  $\tilde{\nu}$  be the outputs of Algorithm 3. With probability  $1 - 2n^{-4}$ ,*

$$(1 - \varepsilon)\frac{1}{2}\mu(G) \leq \tilde{\mu} \leq \mu(G) \quad \& \quad \nu(G) \leq \tilde{\nu} \leq (1 + \varepsilon)2\nu(G).$$



*Proof.* Let us now measure the expected value of our estimates. From our definition,  $X_i = 1$  if and only if a random vertex  $v_i$  for a random permutation  $\pi$  is matched in  $\text{GMM}(G, \pi)$ . Since the number of vertices matched in a matching is twice the size of the matching, this implies that

$$\mathbf{E}[X_i] = \Pr_{v_i, \pi}[X_i = 1] = \frac{2 \mathbf{E}_\pi |\text{GMM}(G, \pi)|}{n}.$$

As such,

$$\mathbf{E}[X] = \mathbf{E}[X_1 + \dots + X_k] = \frac{2k \mathbf{E}_\pi |\text{GMM}(G, \pi)|}{n}. \quad (3)$$

Since  $X$  is sum of independent Bernoulli random variables, by the Chernoff bound (Proposition 2.1):

$$\Pr \left[ |X - \mathbf{E}[X]| \geq \sqrt{12 \mathbf{E}[X] \ln n} \right] \leq 2 \exp \left( -\frac{12 \mathbf{E}[X] \ln n}{3 \mathbf{E}[X]} \right) = 2/n^4. \quad (4)$$

Noting from Algorithm 3 that  $f \cdot n = Xn/k$ , inequality (4) implies that with probability  $1 - 2/n^4$ ,

$$\begin{aligned} f \cdot n &\in \frac{(\mathbf{E}[X] \pm \sqrt{12 \mathbf{E}[X] \ln n})n}{k} \\ &= \frac{\mathbf{E}[X]n}{k} \pm \sqrt{12 \mathbf{E}[X]n^2k^{-2} \ln n} \\ &= 2 \mathbf{E}_\pi |\text{GMM}(G, \pi)| \pm \sqrt{24 \mathbf{E}_\pi |\text{GMM}(G, \pi)|nk^{-1} \ln n} && \text{(By (3).)} \\ &= 2 \mathbf{E}_\pi |\text{GMM}(G, \pi)| \pm \sqrt{\frac{\mathbf{E}_\pi |\text{GMM}(G, \pi)|\varepsilon^2 n \bar{d}}{128\Delta}}. && \text{(Since } k = 128 \cdot 24 \frac{\Delta \ln n}{\varepsilon^2 \bar{d}} \text{.)} \end{aligned}$$

Note that  $\mu(G) \leq 2 \mathbf{E}_\pi |\text{GMM}(G, \pi)|$  and also recall from Claim 4.2 that  $\mu(G) \geq \frac{n\bar{d}}{4\Delta}$ . As such, we have  $\frac{n\bar{d}}{4\Delta} \leq 2 \mathbf{E}_\pi |\text{GMM}(G, \pi)|$ . Combined with the range above, with probability  $1 - 2/n^4$  we have

$$f \cdot n \in 2 \mathbf{E}_\pi |\text{GMM}(G, \pi)| \pm \sqrt{\frac{(\mathbf{E}_\pi |\text{GMM}(G, \pi)|)^2 \varepsilon^2}{16}} = \left(2 \pm \frac{\varepsilon}{4}\right) \mathbf{E}_\pi |\text{GMM}(G, \pi)|.$$

Since  $\tilde{\mu} = (1 - \frac{\varepsilon}{2})f \cdot n/2$  and  $\tilde{\nu} = (1 + \frac{\varepsilon}{2})f \cdot n$ , this means

$$(1 - \varepsilon) \mathbf{E}_\pi |\text{GMM}(G, \pi)| \leq \tilde{\mu} \leq \mathbf{E}_\pi |\text{GMM}(G, \pi)|, \quad (5)$$

$$2 \mathbf{E}_\pi |\text{GMM}(G, \pi)| \leq \tilde{\nu} \leq (1 + \varepsilon)2 \mathbf{E}_\pi |\text{GMM}(G, \pi)|. \quad (6)$$

Next, observe that  $\frac{1}{2}\mu(G) \leq \mathbf{E}_\pi |\text{GMM}(G, \pi)| \leq \mu(G)$  since a maximal matching is a 2-approximate maximum matching, and  $\nu(G) \leq 2 \mathbf{E}_\pi |\text{GMM}(G, \pi)| \leq 2\nu(G)$  since the set of vertices of a maximal matching is a 2-approximate minimum vertex cover. These, plugged into (5) and (6) give the desired inequalities of the lemma, completing the proof.  $\blacksquare$

We are now ready to prove Theorem 1.2.

*Proof of Theorem 1.2.* By Theorem 3.5, for a random vertex  $v \in V$ ,  $\mathbf{E}_{v, \pi}[T(v, \pi)] = O(\bar{d} \cdot \log n)$ . As such, for each vertex  $v_i$  of Algorithm 3, the algorithm of Lemma 4.1 takes  $\tilde{O}(\bar{d} + 1)$  expected time to determine whether it is matched in a random permutation. Since we run this algorithm for  $k = \tilde{O}(\Delta/\varepsilon^2 \bar{d})$  vertices, the expected running time of Algorithm 3 is  $\tilde{O}(\Delta/\varepsilon^2 \bar{d}) \cdot \tilde{O}(\bar{d} + 1) = \tilde{O}(\Delta/\varepsilon^2)$  where the latter equality crucially uses our assumption of the start of the section that there are no singleton vertices in the graph, which implies  $1/\bar{d} = O(1)$ .

To achieve the high probability bound on the time-complexity, we run  $\Theta(\log n)$  instances of Algorithm 3 in parallel and return the output of the instance that terminates first. By Markov's inequality, each instance terminates in time  $\tilde{O}(\Delta/\varepsilon^2)$  with a constant probability. As such, at least one of the instances terminates in  $\tilde{O}(\Delta/\varepsilon^2)$  time with probability  $1 - 1/\text{poly}(n)$ . Recall also that we spent  $O(n)$  time at the start of the section to throw away singleton vertices and compute  $\bar{d}$  and  $\Delta$ . As such, the total time complexity is, w.h.p.,  $O(n) + \tilde{O}(\Delta/\varepsilon^2)$ . On the other hand, since the approximation ratio guarantee of Lemma 4.3 holds with probability  $1 - 1/\text{poly}(n)$  for each instance, all  $O(\log n)$  instances (including the one that terminates first) achieve the claimed approximation with a high probability of  $1 - 1/\text{poly}(n)$ . ■

## 4.2 Proof of Theorem 1.3: Multiplicative-Additive Approximation

The algorithm we use for Theorem 1.3 is similar to Algorithm 3 for Theorem 1.2 except that the additive  $\varepsilon n$  error of Theorem 1.3 allows us to take  $\tilde{O}(1/\varepsilon^2)$  sample vertices instead of  $\tilde{O}(\Delta/\varepsilon^2 \bar{d})$  as in Algorithm 3. Formally, we use the following Algorithm 4:

---

**Algorithm 4:** An algorithm used for Theorem 1.3, given parameter  $\varepsilon > 0$ .

---

- 1  $k \leftarrow 16 \cdot 24 \ln n / \varepsilon^2$ .
  - 2 Sample  $k$  vertices  $v_1, \dots, v_k$  (with replacement) independently and uniformly from  $V$ .
  - 3 For each  $i \in [k]$  run the algorithm of Lemma 4.1 on vertex  $v_i$ . For each  $i \in [k]$  let  $X_i$  be the indicator of the event that  $v_i$  is matched once we run Lemma 4.1 for  $v_i$ .
  - 4 Let  $X \leftarrow \sum_{i=1}^k X_i$  and let  $f \leftarrow X/k$  be the fraction of vertices  $v_1, \dots, v_k$  that get matched.
  - 5 Let  $\tilde{\mu} \leftarrow \frac{fn}{2} - \frac{\varepsilon}{2}n$  and let  $\tilde{\nu} \leftarrow fn + \frac{\varepsilon}{4}n$ .
  - 6 **return**  $\tilde{\mu}$  as the estimate for  $\mu(G)$  and **return**  $\tilde{\nu}$  as the estimate for  $\nu(G)$ .
- 

**Lemma 4.4.** Let  $\tilde{\mu}$  and  $\tilde{\nu}$  be the outputs of Algorithm 4. With probability  $1 - 2n^{-4}$ ,

$$\frac{1}{2}\mu(G) - \varepsilon n \leq \tilde{\mu} \leq \mu(G) \quad \& \quad \nu(G) \leq \tilde{\nu} \leq 2\nu(G) + \varepsilon n.$$

*Proof.* Observe that inequalities (3) and (4) that we proved for Algorithm 3 hold for Algorithm 4 too for exactly the same reasons. Particularly, inequality (4) implies that with probability  $1 - 2/n^4$ ,

$$\begin{aligned} f \cdot n &\in \frac{(\mathbf{E}[X] \pm \sqrt{12 \mathbf{E}[X] \ln n})n}{k} \\ &= \frac{\mathbf{E}[X]n}{k} \pm \sqrt{12 \mathbf{E}[X]n^2 k^{-2} \ln n} \\ &= 2 \mathbf{E}_\pi |\text{GMM}(G, \pi)| \pm \sqrt{24 \mathbf{E}_\pi |\text{GMM}(G, \pi)|nk^{-1} \ln n} && \text{(By (3).)} \\ &= 2 \mathbf{E}_\pi |\text{GMM}(G, \pi)| \pm \sqrt{\mathbf{E}_\pi |\text{GMM}(G, \pi)|\varepsilon^2 n / 16}. && \text{(Since } k = 16 \cdot 24 \frac{\ln n}{\varepsilon^2} \text{.)} \\ &\in 2 \mathbf{E}_\pi |\text{GMM}(G, \pi)| \pm \varepsilon n / 4. && \text{(Since } \mathbf{E}_\pi |\text{GMM}(G, \pi)| \leq n \text{.)} \end{aligned}$$

Combined with  $\tilde{\mu} = \frac{fn}{2} - \frac{\varepsilon}{2}n$  and  $\tilde{\nu} = fn + \frac{\varepsilon}{4}n$ , this implies that, w.h.p.,

$$\mathbf{E}_\pi |\text{GMM}(G, \pi)| - \varepsilon n \leq \tilde{\mu} \leq \mathbf{E}_\pi |\text{GMM}(G, \pi)|, \tag{7}$$

$$2 \mathbf{E}_\pi |\text{GMM}(G, \pi)| \leq \tilde{\nu} \leq 2 \mathbf{E}_\pi |\text{GMM}(G, \pi)| + \varepsilon n. \tag{8}$$

Plugging  $\frac{1}{2}\mu(G) \leq \mathbf{E}_\pi |\text{GMM}(G, \pi)| \leq \mu(G)$  and  $\nu(G) \leq 2 \mathbf{E}_\pi |\text{GMM}(G, \pi)| \leq 2\nu(G)$  into (7) and (8) completes the proof. ■

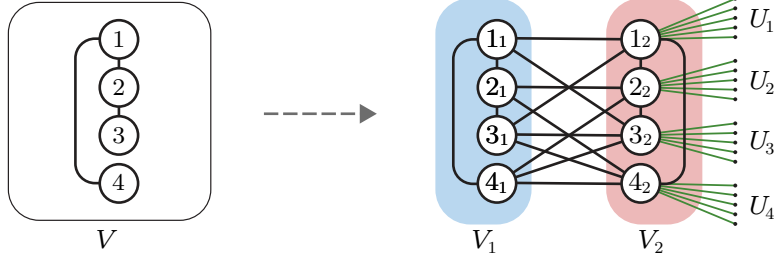


Figure 3: An example of our reduction.

*Proof of Theorem 1.3.* As discussed in the proof of Theorem 1.2, each call to Lemma 4.1 for a randomly chosen vertex takes  $\tilde{O}(\bar{d} + 1)$  expected time. Since  $k = \tilde{O}(1/\varepsilon^2)$ , Algorithm 4 takes  $\tilde{O}((\bar{d} + 1)/\varepsilon^2)$  expected time in total.

To achieve the high probability bound on the time-complexity, as in the proof of Theorem 1.2, we run  $\Theta(\log n)$  instances of Algorithm 4 in parallel and return the output of the instance that terminates first. By Markov's inequality, each instance terminates in time  $\tilde{O}((\bar{d} + 1)/\varepsilon^2)$  with a constant probability. As such, at least one of the instances terminates in  $\tilde{O}((\bar{d} + 1)/\varepsilon^2)$  time with probability  $1 - 1/\text{poly}(n)$ . On the other hand, since the approximation guarantee of Lemma 4.4 holds with probability  $1 - 1/\text{poly}(n)$  for each instance, *all*  $O(\log n)$  instances (including the one that terminates first) achieve a  $(2, \varepsilon n)$ -approximation with probability  $1 - 1/\text{poly}(n)$ . ■

## 5 The Final Algorithm for the Adjacency Matrix Query Model

In this section, we prove Theorem 1.4. The first challenge is that Lemma 4.1 is based on adjacency list queries. As such, we need a way of implementing these adjacency list queries in the adjacency matrix model. To do this, we transform the input graph  $G$  to another graph  $H$  where the adjacency list queries to  $H$  can be implemented efficiently via adjacency matrix queries to  $G$  and at the same time the oracle calls on  $H$  suffice to approximate the size of MCM and MVC for  $G$ .

We note that another such reduction was given before by [24]. However, the reduction of [24] is not applicable in our case since it, crucially, adds parallel edges and self-loops to  $H$  while Theorem 3.5 is proved for simple graphs.

We first make a mild assumption that  $\varepsilon$  in Theorem 1.4 is at least  $1/n$ , noting that otherwise  $\tilde{O}(n/\varepsilon^3)$  is large enough to query the whole graph, making Theorem 1.4 trivial.

Let us now formalize the construction of graph  $H = (V_H, E_H)$  from the input graph  $G = (V, E)$  (see Figure 3 for an illustration). Throughout this section, we continue to use  $n$  to denote the number of vertices in the original graph  $G$ . Letting  $s := 10n/\varepsilon$ , the construction is as follows:

- As for the vertex-set  $V_H$  we have  $V_H = V_1 \cup V_2 \cup U_1 \cup \dots \cup U_n$  where:
  - $V_1 := \{1_1, 2_1, \dots, n_1\}$  and  $V_2 := \{1_2, 2_2, \dots, n_2\}$  are two copies of  $V$  both with size  $n$ .
  - For each  $i \in [n]$ , subset  $U_i = \{1'_i, 2'_i, \dots, s'_i\}$  has size  $s$ .

Note in particular that  $H$  has  $2n + ns = \Theta(n^2/\varepsilon)$  vertices.

- We define the edge-set  $E_H$  by specifying the adjacency lists of the vertices in  $V_H$  while being careful that each edge  $(u, v) \in E_H$  appears in the adjacency lists of both  $u$  and  $v$ . See Figure 3 for reference.

- Let  $v \in [n]$ . Vertex  $v_1 \in V_1$  has degree exactly  $n$  in  $H$ . For any  $i \in [n]$ , if  $(v, i) \in E$  then the  $i$ -th neighbor of  $v_1$  is vertex  $i_1 \in V_1$ , otherwise it is vertex  $i_2 \in V_2$ .
- Let  $v \in [n]$ . Vertex  $v_2 \in V_2$  has degree exactly  $n + s$ . For any  $i \in [n]$ , if  $(v, i) \in E$  then the  $i$ -th neighbor of  $v_2$  is vertex  $i_2 \in V_2$ , otherwise it is vertex  $i_1 \in V_1$ . The last  $s$  vertices in the adjacency list of  $v_2$  are the vertices in  $U_v$ .
- Each vertex  $u \in U_v$  for any  $v \in [n]$  has exactly one neighbor  $v_2 \in V_2$ .

Note that  $H[V_1]$  and  $H[V_2]$  are both isomorphic to  $G$ .

The following observation follows immediately from the construction above and the way neighbors of each vertex are ordered in their adjacency lists:

**Observation 5.1.** *For any vertex  $v \in V_H$ ,  $\deg_H(v)$  does not depend on the edges in  $G$  and is, therefore, known a priori with zero queries to  $G$ . Furthermore, for any  $v \in V_H$  and any  $i \in [\deg_H(v)]$  one can determine the  $i$ -th edge of  $v$  in  $H$  by making at most one adjacency matrix query to  $G$ .*

Let us use  $T_H(v, \pi)$  instead of  $T(v, \pi)$  (defined in Section 3 and used in Lemma 4.1) to emphasize that we run RGMM on graph  $H$  and not  $G$  in this section.

**Claim 5.2.** *Let  $\pi$  be a random permutation over the edge-set  $E_H$  of  $H$ . For a vertex  $v$  chosen uniformly at random from  $V_1$  and independently from  $\pi$ ,*

$$\mathbf{E}_{v \sim V_1, \pi} [T_H(v, \pi)] = \tilde{O}(n/\varepsilon).$$

*Proof.* By the construction, graph  $H$  has  $|E_H| = O(n^2 + ns) = \tilde{\Theta}(n^2/\varepsilon)$  edges and  $|V_H| = 2n + ns = \Theta(n^2/\varepsilon)$  vertices. Applying Theorem 3.5, for a vertex  $v \in V_H$  chosen uniformly at random, we get

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

Instead of querying a random vertex  $v \in V_H$ , suppose that we sum over all of them. This gives:

$$\sum_{v \in V_H} \mathbf{E}_{\pi} [T_H(v, \pi)] = |V_H| \cdot \mathbf{E}_{v \sim V_H, \pi} [T_H(v, \pi)] = O(|E_H| \log |V_H|) = \tilde{O}(n^2/\varepsilon). \quad (9)$$

Finally, since  $|V_1| = n$ , for a vertex  $v$  chosen uniformly at random from  $V_1$ , we have

$$\mathbf{E}_{v \sim V_1, \pi} [T_H(v, \pi)] \leq \left( \sum_{v \in V_H} \mathbf{E}_{\pi} [T_H(v, \pi)] \right) / |V_1| \stackrel{(9)}{=} \tilde{O}(n^2/\varepsilon)/n = \tilde{O}(n/\varepsilon). \quad \blacksquare$$

Now that we know queries starting from a random vertex in  $V_1$  can be implemented efficiently in  $\tilde{O}(n/\varepsilon)$  expected time, the question is how can we use these queries to get our estimators for the size of maximum matching and minimum vertex cover of the original graph  $G$ .

For any permutation  $\pi$  over  $E_H$ , let us define

$$M_1(\pi) := \text{GMM}(H, \pi) \cap (V_1 \times V_1)$$

to be the set of edges in matching  $\text{GMM}(H, \pi)$  with both endpoints in  $V_1$ . The following Claim 5.3 shows that the expected value of  $M_1(\pi)$  for a random permutation  $\pi$  can be used for approximating both maximum matching and the minimum vertex cover of  $G$ .

The idea behind Claim 5.3 is as follows. Note that by construction,  $H[V_1]$  is isomorphic to  $G$  and so for any  $\pi$ , there is a matching in  $G$  that corresponds to  $M_1(\pi)$ . However, the vertices in  $V_1$  can also be matched to the vertices in  $V_2$  in graph  $H$ , and so  $M_1(\pi)$  is not necessarily a *maximal* matching of  $H[V_1]$ . The key insight, however, is that the vast majority of vertices  $v_2 \in V_2$  will actually be matched to their neighbors in  $U_v$  in a RGMM of  $H$ . Hence,  $M_1(\pi)$  for a random permutation  $\pi$  is indeed close to a maximal matching of  $H[V_1]$  and its size can be used to approximate both the size of MCM and that of the MVC of  $G$ . Formally:

**Claim 5.3.** *It holds that*

$$\frac{1}{2}\mu(G) - \frac{\varepsilon}{20}n \leq \mathbf{E}_\pi |M_1(\pi)| \leq \mu(G) \quad \& \quad \nu(G) - \frac{\varepsilon}{10}n \leq 2\mathbf{E}_\pi |M_1(\pi)| \leq 2\nu(G).$$

*Proof.* Let us use  $B_1(\pi)$  to denote the set of vertices in  $V_1$  that are matched to  $V_2$  in  $\text{GMM}(H, \pi)$ . We first show that for every permutation  $\pi$  over  $E_H$ , it holds that

$$\frac{1}{2}(\mu(G) - |B_1(\pi)|) \leq |M_1(\pi)| \leq \mu(G) \quad \& \quad \nu(G) - |B_1(\pi)| \leq 2|M_1(\pi)| \leq 2\nu(G). \quad (10)$$

For the first inequality, observe that  $M_1(\pi)$  is a matching of  $H[V_1]$  and  $H[V_1]$  is isomorphic to  $G$  by the construction; thus, clearly  $|M_1(\pi)| \leq \mu(G)$ .

For the second inequality, observe that  $M_1(\pi)$  is a maximal matching of  $H[V_1 \setminus B_1(\pi)]$  which is isomorphic to  $G[V \setminus B_1(\pi)]$ . Since a maximal matching is at least half the size of a maximum matching, we get  $|M_1(\pi)| \geq \frac{1}{2}\mu(G[V \setminus B_1(\pi)]) \geq \frac{1}{2}(\mu(G) - |B_1(\pi)|)$ .

For the third inequality, note that since  $M_1(\pi)$  is a maximal matching of  $H[V_1 \setminus B_1(\pi)]$ , the set of vertices matched by it covers all edges in  $H[V_1 \setminus B_1(\pi)]$ . Adding the vertices in  $B_1(\pi)$  to this set, we cover all edges in  $H[V_1]$ . Hence  $\nu(H[V_1]) \leq 2|M_1(\pi)| + |B_1(\pi)|$ . The inequality then follows from  $H[V_1]$  being isomorphic to  $G$ .

For the fourth inequality, note that since  $M_1(\pi)$  is a matching in  $H[V_1]$  and each vertex can cover at most one edge of it, we have  $|M_1(\pi)| \leq \nu(H[V_1])$ . Given that  $H[V_1]$  is isomorphic to  $G$ , we thus get  $|M_1(\pi)| \leq \nu(G)$ . Multiplying through by a factor of 2 and adding  $|B_1(\pi)|$  to both sides, we get  $2|M_1(\pi)| + |B_1(\pi)| \leq 2\nu(G) + |B_1(\pi)|$ .

Now, observe that in any permutation  $\pi$ , if the lowest rank neighbor of a vertex  $v_2 \in V_2$  is connected to  $U_v$ , then this edge is in matching  $\text{GMM}(\pi)$  since the vertices in  $U_v$  have degree exactly one. Moreover, since each vertex  $v_2 \in V_2$  has degree exactly  $n+s$  and  $s$  of these neighbors are to  $U_v$ , the minimum rank edge of  $v_2$  is indeed connected to  $U_v$  with probability  $\frac{s}{n+s} = \frac{10n/\varepsilon}{n+10n/\varepsilon} \geq 1 - \varepsilon/10$ . As such the number of vertices in  $V_2$  that are matched to  $V_1$  is at most  $\frac{\varepsilon}{10}|V_2| = \frac{\varepsilon}{10}n$  in expectation taken over  $\pi$ . This, in turn, implies that  $\mathbf{E}_\pi |B_1(\pi)| \leq \frac{\varepsilon}{10}n$ . Taking expectation over a random  $\pi$  in (10) and plugging this upper bound for  $\mathbf{E}_\pi |B_1(\pi)|$  proves the claim.  $\blacksquare$

Finally, the algorithm we use for Theorem 1.4 is formalized below as Algorithm 5.

Let us analyze the approximation ratio of Algorithm 5.

**Lemma 5.4.** *Let  $\tilde{\mu}$  and  $\tilde{\nu}$  be the outputs of Algorithm 5. With probability  $1 - 2n^{-4}$ ,*

$$\frac{1}{2}\mu(G) - \varepsilon n \leq \tilde{\mu} \leq \mu(G) \quad \& \quad \nu(G) \leq \tilde{\nu} \leq 2\nu(G) + \varepsilon n.$$

*Proof.* Let us now measure the expected value of our estimates. From our definition,  $X_i = 1$  if and only if a random vertex  $v^i \in V_1$  for a random permutation  $\pi$  is matched to another vertex of

---

**Algorithm 5:** An algorithm used for Theorem 1.4, given parameter  $\varepsilon > 0$ .

---

- 1 Let  $H = (V_H, E_H)$  be the graph described above (we do not explicitly construct  $H$  here).
  - 2  $k \leftarrow 16 \cdot 24 \ln n / \varepsilon^2$ .
  - 3 Sample  $k$  vertices  $v^1, \dots, v^k$  (with replacement) independently and uniformly from  $V_1$ .
  - 4 For each  $i \in [k]$  run the algorithm of Lemma 4.1 on vertex  $v^i$ , for graph  $H$ . Let  $X_i$  be the indicator of the event that  $v^i$  is matched to another vertex in  $V_1$ .
  - 5 Let  $X \leftarrow \sum_{i=1}^k X_i$  and let  $f \leftarrow X/k$  be the fraction of  $v_1, \dots, v_k$  that get matched to  $V_1$ .
  - 6 Let  $\tilde{\mu} \leftarrow \frac{fn}{2} - \frac{\varepsilon}{2}n$  and  $\tilde{\nu} \leftarrow fn + \frac{\varepsilon}{2}n$ .
  - 7 **return**  $\tilde{\mu}$  as the estimate for  $\mu(G)$  and **return**  $\tilde{\nu}$  as the estimate for  $\nu(G)$ .
- 

$V_1$  in  $\text{GMM}(H, \pi)$ . Recall that we defined  $M_1(\pi)$  to be the set of edges in  $\text{GMM}(H, \pi) \cap (V_1 \times V_1)$ . Combined with the fact that the number of vertices matched in a matching is twice the size of the matching, this implies that

$$\mathbf{E}[X_i] = \Pr_{v^i, \pi}[X_i = 1] = \frac{2 \mathbf{E}_\pi |M_1(\pi)|}{|V_1|} = \frac{2 \mathbf{E}_\pi |M_1(\pi)|}{n}.$$

As such,

$$\mathbf{E}[X] = \mathbf{E}[X_1 + \dots + X_k] = \frac{2k \mathbf{E}_\pi |M_1(\pi)|}{n}. \quad (11)$$

Since  $X$  is sum of independent Bernoulli random variables, by the Chernoff bound (Proposition 2.1):

$$\Pr \left[ |X - \mathbf{E}[X]| \geq \sqrt{12 \mathbf{E}[X] \ln n} \right] \leq 2 \exp \left( -\frac{12 \mathbf{E}[X] \ln n}{3 \mathbf{E}[X]} \right) = 2/n^4. \quad (12)$$

Noting from Algorithm 5 that  $f \cdot n = Xn/k$ , inequality (12) implies that with probability  $1 - 2/n^4$ ,

$$\begin{aligned} f \cdot n &\in \frac{(\mathbf{E}[X] \pm \sqrt{12 \mathbf{E}[X] \ln n})n}{k} \\ &= \frac{\mathbf{E}[X]n}{k} \pm \sqrt{12 \mathbf{E}[X]n^2k^{-2} \ln n} \\ &= 2 \mathbf{E}_\pi |M_1(\pi)| \pm \sqrt{24 \mathbf{E}_\pi |M_1(\pi)|nk^{-1} \ln n} && \text{(By (11).)} \\ &= 2 \mathbf{E}_\pi |M_1(\pi)| \pm \sqrt{\mathbf{E}_\pi |M_1(\pi)|\varepsilon^2n/16} && \text{(Since } k = 16 \cdot 24 \frac{\ln n}{\varepsilon^2}\text{.)} \\ &\in 2 \mathbf{E}_\pi |M_1(\pi)| \pm \frac{\varepsilon n}{4}. && \text{(Since } \mathbf{E}_\pi |M_1(\pi)| \leq |V_1| = n\text{.)} \end{aligned}$$

Combined with  $\frac{1}{2}\mu(G) - \frac{\varepsilon}{20}n \leq \mathbf{E}_\pi |M_1(\pi)| \leq \mu(G)$  from Claim 5.3, and  $\tilde{\mu} = \frac{fn}{2} - \frac{\varepsilon}{2}n$ , the range above for  $fn$  implies  $\frac{1}{2}\mu(G) - \varepsilon n < \tilde{\mu} < \mu(G)$ .

Combined with  $\nu(G) - \frac{\varepsilon}{10}n \leq 2 \mathbf{E}_\pi |M_1(\pi)| \leq 2\nu(G)$  from Claim 5.3 and  $\tilde{\nu} = fn + \frac{\varepsilon}{2}n$ , the range above for  $fn$  implies  $\nu(G) < \tilde{\nu} < 2\nu(G) + \varepsilon n$ .  $\blacksquare$

*Proof of Theorem 1.4.* By Claim 5.2 each call to Lemma 4.1 for a random vertex from  $V_1$  leads to  $\tilde{O}(n/\varepsilon)$  adjacency list queries to  $H$ . By Observation 5.1, each of these adjacency list queries to  $H$ , can be implement with one adjacency matrix query to  $G$  in  $O(1)$  time. The total expected number of adjacency matrix queries to  $G$  and the time-complexity of the algorithm, is therefore,  $k \cdot \tilde{O}(n/\varepsilon) = \tilde{O}(n/\varepsilon^3)$ .

To achieve the high probability bound on the time-complexity, as in the proofs of Theorems 1.2 and 1.3, we run  $\Theta(\log n)$  instances of Algorithm 5 in parallel and return the output of the instance that terminates first. By Markov’s inequality, each instance terminates in time  $\tilde{O}(n/\varepsilon^3)$  with a constant probability. As such, at least one of the instances terminates in  $\tilde{O}(n/\varepsilon^3)$  time with probability  $1 - 1/\text{poly}(n)$ . On the other hand, since the approximation guarantee of Lemma 5.4 holds with probability  $1 - 1/\text{poly}(n)$  for each instance, *all*  $O(\log n)$  instances (including the one that terminates first) achieve a  $(2, \varepsilon n)$ -approximation with probability  $1 - 1/\text{poly}(n)$ . ■

## 6 Acknowledgements

I thank Sanjeev Khanna for many helpful discussions, and additionally thank Yu Chen, Sanjeev Khanna, and Zihan Tan for insightful comments.

## References

- [1] Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-Efficient Local Computation Algorithms. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1132–1139. 4
- [2] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear Algorithms for  $(\Delta + 1)$  Vertex Coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 767–786. SIAM, 2019. 1
- [3] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, Vahab S. Mirrokni, and Warren Schudy. Massively Parallel Computation via Remote Memory Access. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 59–68. ACM, 2019. 4
- [4] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, Vahab S. Mirrokni, and Warren Schudy. Parallel Graph Algorithms in Constant Adaptive Rounds: Theory meets Practice. *Proc. VLDB Endow.*, 13(13):3588–3602, 2020. 4
- [5] Guy E. Blelloch, Jeremy T. Fineman, and Julian Shun. Greedy Sequential Maximal Independent Set and Matching are Parallel on Average. In *24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA ’12, Pittsburgh, PA, USA, June 25-27, 2012*, pages 308–317, 2012. 5, 12, 13
- [6] Karl Bringmann, Fabian Kuhn, Konstantinos Panagiotou, Ueli Peter, and Henning Thomas. Internal DLA: Efficient Simulation of a Physical Growth Model - (Extended Abstract). In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 247–258, 2014. 14, 25
- [7] Moses Charikar, Weiyun Ma, and Li-Yang Tan. Unconditional Lower Bounds for Adaptive Massively Parallel Computation. In *SPAA ’20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 141–151. ACM, 2020. 4
- [8] Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the Minimum Spanning Tree Weight in Sublinear Time. *SIAM J. Comput.*, 34(6):1370–1379, 2005. 1

- [9] 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. [1](#), [3](#)
- [10] Artur Czumaj and Christian Sohler. Estimating the Weight of Metric Minimum Spanning Trees in Sublinear Time. *SIAM J. Comput.*, 39(3):904–922, 2009. [1](#)
- [11] Artur Czumaj and Christian Sohler. Sublinear-time Algorithms. In *Property Testing - Current Research and Surveys*, volume 6390 of *Lecture Notes in Computer Science*, pages 41–64. Springer, 2010. [1](#)
- [12] Artur Czumaj and Christian Sohler. Sublinear Time Approximation of the Cost of a Metric  $k$ -Nearest Neighbor Graph. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2973–2992. SIAM, 2020. [1](#)
- [13] Hossein Esfandiari and Michael Mitzenmacher. Metric Sublinear Algorithms via Linear Sampling. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 11–22. IEEE Computer Society, 2018. [1](#)
- [14] Martin Farach-Colton and Meng-Tsung Tsai. Exact Sublinear Binomial Sampling. *Algorithmica*, 73(4):637–651, 2015. [14](#), [25](#)
- [15] Uriel Feige. On Sums of Independent Random Variables with Unbounded Variance and Estimating the Average Degree in a Graph. *SIAM J. Comput.*, 35(4):964–984, 2006. [1](#)
- [16] Manuela Fischer and Andreas Noever. Tight Analysis of Parallel Randomized Greedy MIS. *ACM Trans. Algorithms*, 16(1):6:1–6:13, 2020. [5](#), [12](#)
- [17] Mohsen Ghaffari and Jara Uitto. Sparsifying Distributed Algorithms with Ramifications in Massively Parallel Computation and Centralized Local Computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1636–1653. SIAM, 2019. [4](#)
- [18] Oded Goldreich and Dana Ron. Approximating Average Parameters of Graphs. *Random Struct. Algorithms*, 32(4):473–493, 2008. [1](#)
- [19] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 734–751. SIAM, 2014. [1](#)
- [20] 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. [2](#), [3](#)
- [21] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948. SIAM, 2010. [4](#)



- [22] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local Computation: Lower and Upper Bounds. *J. ACM*, 63(2):17:1–17:44, 2016. [2](#)
- [23] 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. [1](#), [2](#), [4](#), [6](#)
- [24] 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. [1](#), [2](#), [3](#), [6](#), [14](#), [17](#), [24](#)
- [25] 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. [1](#), [2](#), [3](#)
- [26] Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 225–234, 2009. [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [13](#)

## A Implementation Details

Here we prove Lemma 4.1 using similar ideas to [24, Section 4]. Our modifications to the proof of [24] are straightforward and we claim no novelty in this section. As already discussed in Section 4, the first (standard) idea is to generate the random permutation  $\pi$  locally by sampling an independent rank  $\sigma_e$  for each edge  $e$  which is a real in  $[0, 1]$  chosen uniformly at random, and forming  $\pi$  by sorting the edges in the increasing order of their ranks.

Now suppose w.l.o.g. that  $\Delta$  is a power of 2 and consider the following sub-intervals of  $[0, 1]$ :<sup>5</sup>

$$I_0 := [0, 1/\Delta), \quad I_1 := [1/\Delta, 2/\Delta), \dots, I_i := [2^{i-1}/\Delta, 2^i/\Delta), \dots, I_{\log_2 \Delta} := [0.5, 1),$$

and define  $s_i$  such that  $I_i = [s_i, s_{i+1})$ . For any vertex  $v$ , we maintain the following data structures:

- **exposed**( $v$ ): This is a subset of the neighbor set  $N(v)$  of  $v$  in graph  $G$  that is initially empty. For any  $u \in \mathbf{exposed}(v)$ , vertex  $v$  knows the rank  $\sigma_{(v,u)}$  of the edge between them (but  $v$  may not know the location of  $u$  in its adjacency list as this edge may have been notified to  $v$  by  $u$ ). We store **exposed**( $v$ ) in two binary search trees, one indexed by the vertex IDs and the other indexed by the rank of the edges.
- $k(v)$ : This is an integer in  $\{0, \dots, (\log_2 \Delta) + 1\}$  which is initially 0 for every vertex  $v \in V$ . For any  $v \in V$  and any edge  $e = (v, u)$  of  $v$  with rank  $\sigma_e \in I_0 \cup \dots \cup I_{k(v)-1}$ , it will hold that  $u \in \mathbf{exposed}(v)$  and so  $v$  is “aware” of this edge  $e$  and its rank. Since initially  $k(v) = 0$  and  $s_0 = 0$  for all  $v \in V$ , the vertices do not need to be aware of any of their edges initially.
- **count**( $v, i$ ): The number of neighbors  $u$  of  $v$  in **exposed**( $v$ ) such that  $\sigma_{(v,u)} \in I_i$ .

Armed with these data structures, we will describe a procedure **lowest**( $v, i$ ) which returns a vertex  $w$  where  $(v, w)$  is the  $i$ -th lowest rank edge of  $v$ . We first formalize how we can implement the oracles using this procedure and then formalize it.

---

**Algorithm 6:** Implementation of the vertex oracle  $\mathbf{VO}(v)$ .

---

```

1  $j \leftarrow 1$ .
2  $w \leftarrow \mathbf{lowest}(v, j)$ .
3 while  $j \leq \deg(v)$  do
4   if  $\mathbf{EO}((v, w), w) = \mathbf{TRUE}$  then return  $\mathbf{TRUE}$ 
5    $j \leftarrow j + 1$ .
6    $w \leftarrow \mathbf{lowest}(v, j)$ .
7 return  $\mathbf{FALSE}$ 
```

---



---

**Algorithm 7:** Implementation of the edge oracle  $\mathbf{EO}(e = (u, v), u)$ .

---

```

1 if we have already computed  $\mathbf{EO}(e, u)$  then return the computed answer.
2  $j \leftarrow 1$ .
3  $w \leftarrow \mathbf{lowest}(u, j)$ .
4 while  $w \neq v$  do
5   if  $\mathbf{EO}((u, w), w) = \mathbf{TRUE}$  then return  $\mathbf{FALSE}$ .
6    $j \leftarrow j + 1$ ,
7    $w \leftarrow \mathbf{lowest}(u, j)$ .
8 return  $\mathbf{TRUE}$ 
```

---

<sup>5</sup>If  $\Delta$  is not a power of two, just take the smallest power of two that is larger than  $\Delta$ .

Next, we turn to formalize how we implement function `lowest`( $v, i$ ). To do this, we need a procedure `expose_next`( $v$ ) which increases  $k(v)$  by one and ensures that the invariants of the data structures continue to hold. That is, to increase  $k(v)$  by one, we first make sure that  $v$  is made aware of all of its edges in interval  $I_{k(v)}$  by having them stored in `exposed`( $v$ ). Once we have `expose_next`( $v$ ), procedure `lowest`( $v, i$ ) can be implemented as follows:

---

**Algorithm 8:** `lowest`( $v, i$ )

---

- 1 **if**  $i \geq \text{deg}(v)$  **then return**  $\emptyset$ .
  - 2 **while**  $\text{count}(v, 1) + \dots + \text{count}(v, k(v) - 1) < i$  **do**
  - 3     | `expose_next`( $v$ ).
  - 4 Let  $(v, w)$  be the  $i$ -th lowest-rank edge of  $v$  in `exposed`( $v$ ). This can be found in  $O(\log \Delta)$  time using the BST that `exposed`( $v$ ) is stored in which is indexed by the ranks.
  - 5 **return**  $w$ .
- 

**Procedure `expose_next`( $v$ ):** To expose all the edges of  $v$  in interval  $I_{k(v)}$ , we first take a subsample  $S(v, k(v))$  of  $[\text{deg}(v)]$  by including each element independently with prob.  $p := |I_{k(v)}|/(1 - s_{k(v)})$ . Note that  $p$  is the probability that the rank of an edge is in  $I_{k(v)}$  conditioned on that its rank is in  $I_{k(v)} \cup \dots \cup I_{\log_2 \Delta}$ . The goal is to assign a uniform rank from  $I_{k(v)}$  to any edge of  $v$  whose index in  $v$ 's adjacency list is sampled in  $S(v, k(v))$ , but we have to be careful that these ranks do not contradict the previously drawn ranks. To do this, for each sampled index  $i \in S(v, k(v))$ , we query the  $i$ -th neighbor  $u$  of  $v$ . If  $u \in \text{exposed}(v)$ , we do not reveal a new rank for  $(u, v)$  as  $\sigma((u, v))$  is already drawn. Otherwise, we consider the two cases  $k(u) \leq k(v)$  and  $k(u) \geq k(v) + 1$  individually. In the former case, we go ahead and draw a rank  $\sigma((u, v)) \in I_{k(v)}$  uniformly at random, add  $u$  to `exposed`( $v$ ), increase  $\text{count}(v, k(v))$  by one, add  $v$  to `exposed`( $u$ ), and increase  $\text{count}(u, k(v))$  by one. In the latter case, however, the fact that  $k(u) \geq k(v) + 1$  implies that  $u$  is already aware of all of its edges with ranks in  $I_{k(v)}$  which combined with  $u \notin \text{exposed}(v)$  implies that  $(u, v)$  should not have a rank in  $I_{k(v)}$ . Hence, in this case we do not reveal a new rank. At the end, we increase  $k(v)$  by one. It can be confirmed that with this process, the rank  $\sigma(e)$  of any edge  $e$  in the graph is drawn independently and uniformly at random from  $[0, 1]$ .

To implement `expose_next`( $v$ ) efficiently, instead of forming the subsample  $S(v, k(v))$  of  $[\text{deg}(v)]$  by going over the elements in  $[\text{deg}(v)]$  one by one and sampling each independently with probability  $p$  which takes  $\Theta(\text{deg}(v))$  time, we first draw its size  $|S(v, k(v))|$  from the binomial distribution  $B(\text{deg}(v), p)$ . This can be done in polylog  $n$  time both in expectation and with probability  $1 - 1/\text{poly}(n)$ , using the sublinear time *exact* binomial samplers of [14, 6].<sup>6</sup> Then, we pick an  $|S(v, k(v))|$ -size subset  $S(v, k(v))$  of  $[\text{deg}(v)]$  uniformly at random which can be done in  $\tilde{O}(|S(v, k(v))|)$  time.

We now turn to analyze the time-complexity of the algorithm. We start with two claims.

**Claim A.1.** *Consider a vertex  $v$  and suppose that we run `expose_next`( $v$ ) when  $k(v)$  equals some number  $k$ ; then procedure `expose_next`( $v$ ) takes only  $\tilde{O}(\text{deg}_{k-1}(v) + 1)$  time where  $\text{deg}_i(v)$  is the number of edges of  $v$  that end up receiving ranks in  $I_i$  (in case  $k = 0$  define  $\text{deg}_{-1}(v) = 0$ ). This guarantee holds, w.h.p., throughout the algorithm for all calls to `expose_next`( $v$ ) and all  $v \in V$ .*

*Proof.* As discussed, `expose_next`( $v$ ) runs in time  $\tilde{O}(|S(v, k)|)$ . Moreover,

$$\mathbf{E} |S(v, k)| = \frac{|I_k|}{1 - s_k} \text{deg}(v) \leq 2|I_k| \text{deg}(v),$$

---

<sup>6</sup>Particularly, see Theorem 2 of [14] and the paragraph next to it which refers to [6].

where the inequality follows from  $s_{k(v)} \leq 1/2$  which itself follows from  $k(v) \leq \log_2 \Delta$ , noting that if  $k(v) = \log_2 \Delta + 1$  we do not call `expose_next(v)`. On the other hand, we have

$$\mathbf{E}[\deg_{k-1}(v)] = |I_{k-1}| \deg(v) \geq (|I_k| \deg(v) - 1)/2.$$

Since  $S(v, k)$  and  $\deg_k(v)$  are both sums of independent Bernoulli random variables, Chernoff bound asserts they are highly concentrated around their expectations. A union bound over all  $n$  vertices and the  $O(\log \Delta)$  times that we may call `expose_next(\cdot)` for each finishes the proof.  $\blacksquare$

**Claim A.2.** *Consider a vertex  $v$  and suppose that we call procedures `lowest(v, 1), \dots, lowest(v, j)` for some  $1 \leq j \leq \deg(v)$ . The total time spent implementing all these  $j$  calls is upper bounded by  $\tilde{O}(j)$ . The guarantee holds, w.h.p., for all  $v \in V$  and all  $j$  throughout the algorithm.*

*Proof.* Let  $K$  be the final value of  $k(v)$  after running `lowest(v, 1), \dots, lowest(v, j)`. The condition of the while-loop in Algorithm 8 throughout all these  $j$  calls to `lowest(v, \cdot)` is evaluated at most  $K + j$  times since at most  $K$  turn out to be true and exactly  $j$  are false. Each evaluation takes  $O(K)$  time. As such, the total running time spent directly in `lowest(v, 1), \dots, lowest(v, j)` (i.e., ignoring the time spent in `expose_next(v)`) is  $O((K + j)K) = \tilde{O}(j)$  since  $K \leq \log_2 \Delta + 1$ .

By Claim A.1, the  $K$  calls to `expose_next(v)` take  $\tilde{O}(\deg_{-1}(v) + \dots + \deg_{K-2}(v))$  time. On the other hand,  $\deg_{-1}(v) + \dots + \deg_{K-2}(v) < j$  or otherwise, after the first  $K-1$  calls to `expose_next(v)`, we get `count(v, 0) + \dots + count(v, K-2) \geq j` and so we do not call `expose_next(v)` another time. Hence, the overall time-complexity is indeed  $\tilde{O}(j)$ .  $\blacksquare$

Now we turn to analyze the total time-complexity of the algorithm. Consider a call to the edge oracle  $\text{EO}(e = (u, v), u)$  of Algorithm 7 and let  $j$  be the final value of  $j$  in Algorithm 7. Observe that the time spent directly in  $\text{EO}(e, u)$  is  $O(j)$ . We also call `lowest(u, 1), \dots, lowest(u, j-1)` and generate  $O(j)$  recursive calls to Algorithm 7. As we showed in Claim A.2, the total time needed to execute the calls to `lowest(u, 1), \dots, lowest(u, j)` is  $\tilde{O}(j)$ . Hence, the time spent directly in  $\text{EO}(e, u)$  as well as the cost of calling `lowest(\cdot, \cdot)` by it is near-linear in the number of new recursive calls to Algorithm 7 generated directly by  $\text{EO}(e, u)$ . Similarly, the time spent directly in the vertex oracle  $\text{VO}(v)$  and its calls to `lowest(\cdot, \cdot)` is near-linear in the number of edge-oracle calls generated directly by  $\text{VO}(v)$ . As such, the total time-complexity of the algorithm is near-linear in the total number of calls to the edge-oracle of Algorithm 7.

To finalize the proof, let  $\pi$  be the permutation corresponding to  $\sigma$  obtained by sorting the edges in the increasing order of their  $\sigma$ -ranks. Suppose that we run the vertex oracle  $\text{VO}(v, \sigma)$  in Algorithm 6 and also run the original vertex oracle  $\text{VO}(v)$  of Algorithm 1 with respect to  $\pi$ . Observe that the number of recursive calls to Algorithm 7 via Algorithm 6 is exactly equal to the number of recursive calls to Algorithm 2 via Algorithm 1 which is  $T(v, \pi)$ . By the discussion of the previous paragraph, this means that the algorithm has time complexity  $\tilde{O}(T(v, \pi))$ .