Nearly optimal edge estimation with independent set queries

Xi Chen* Amit Levi† Erik Waingarten‡

November 6, 2019

Abstract

We study the problem of estimating the number of edges of an unknown, undirected graph $G = ([n], E)$ with access to an independent set oracle. When queried about a subset $S \subseteq [n]$ of vertices, the independent set oracle answers whether S is an independent set in G or not. Our first main result is an algorithm that computes a $(1 + \varepsilon)$-approximation of the number of edges m of the graph using $\min(\sqrt{m}, n/\sqrt{m}) \cdot \text{poly}(\log n, 1/\varepsilon)$ independent set queries. This improves the upper bound of $\min(\sqrt{m}, n^2/m) \cdot \text{poly}(\log n, 1/\varepsilon)$ by Beame et al. [3]. Our second main result shows that $\min(\sqrt{m}, n/\sqrt{m})/\text{polylog}(n)$ independent set queries are necessary, thus establishing that our algorithm is optimal up to a factor of $\text{poly}(\log n, 1/\epsilon)$.

1 Introduction

We study the problem of estimating the number of edges of a simple undirected graph $G = ([n], E)$ in the context of sublinear-time graph algorithms. The goal is to design a highly-efficient randomized algorithm that, given a certain type of oracle access to an underlying graph G, outputs a number $\widetilde{m}$ that approximates the number of edges of G. The first result in this direction was by Feige [14], who studied this problem when the oracle is a *degree oracle*: the degree oracle answers queries of the form "what is the degree of a given vertex v?" The algorithm of Feige makes $O(n/\sqrt{m})$ queries to the degree oracle, where m denotes the number of edges of the input graph G, and outputs a $(2 + \varepsilon)$-approximation to m for any constant $\varepsilon > 0$. Moreover, Feige showed that the upper bound of $n/\sqrt{m}$ is tight for a $(2 + \varepsilon)$-approximation, and indeed $\Omega(n^2/m)$ degree queries are necessary for a $(2 - o(1))$-approximation. Soon thereafter, Goldreich and Ron [15] considered an oracle that, in addition to degree queries, can answer *neighbor* queries (i.e., given a vertex $v \in [n]$ and an index j, the oracle returns the jth neighbor of v according to some fixed ordering). Their algorithm uses $\widetilde{O}(n/\sqrt{m})$ [1] queries and outputs a $(1+\varepsilon)$-approximation to m for any constant $\varepsilon > 0$; they further showed that the upper bound is tight up to a $\text{polylog}(n)$ factor.

Since then, sublinear-time algorithms have been developed for a variety of graph problems, including estimating the number of stars [16, 1], triangles [11], k-cliques [12], and arbitrary small subgraphs [2], finding forbidden graph minors [19, 20], sampling edges almost uniformly [13], approximating the minimum weight spanning tree [4, 8, 7], maximum matching [24, 29], and minimum vertex cover [26, 22, 24, 29, 17, 25]. As noted in a recent work of Beame, Har-Peled, Ramamoorthy, Rashtchian, and Sinha [3], all these algorithms interact with oracles that provide only *local* information about the underlying graph (such as degree, neighbor, and *edge existence* queries where an algorithm can ask "is vertex u connected to vertex v?")[2]. They suggested that *non-local* oracle models may be natural in certain scenarios of graph parameter estimation and their non-locality may enable more efficient graph algorithms.

Along this line of investigation, [3] introduced both the *independent set* oracle and the *bipartite independent set* oracle and studied the problem of estimating the number of edges under these two query models. The independent set oracle for a graph $G = ([n], E)$ can be queried with a set $S \subseteq [n]$ of vertices and outputs whether or not S is an independent set in G, i.e. whether or not there exist vertices $u, v \in S$ with $(u, v) \in E$. The bipartite independent set oracle, on the other hand, can be queried with a pair of disjoint sets $S, T \subset [n]$ and outputs whether or not (S, T) is a bipartite independent set in G, i.e. whether or not there exist $u \in S$ and $v \in T$ with $(u, v) \in E$.[3]

*Columbia University, email: xichen@cs.columbia.edu.

†University of Waterloo, email: amit.levi@uwaterloo.ca.

‡Columbia University, email: eaw@cs.columbia.edu.

[1] We use $\widetilde{O}(f(n))$ and $\widetilde{\Omega}(f(n))$ to surpress $\text{polylog}(f(n))$ factors.

[2] One exception is that [2] also uses uniform edge sampling in addition to the above specified queries.

[3] We remark that the bipartite independent set oracle is at least as powerful, up to poly-logarithmic factors, as the independent set oracle. Consider a graph $G = ([n], E)$, a set $S \subseteq [n]$ of vertices, and the question of whether or not S is an independent set. Letting $(\mathbf{S}_1, \mathbf{S}_2)$ be a uniformly random partition of S, we may query the bipartite independent set oracle with $\mathbf{S}_1$ and $\mathbf{S}_2$. If S is an independent set, then $(\mathbf{S}_1, \mathbf{S}_2)$ will be a bipartite independent set; if S is not an independent set, then $(\mathbf{S}_1, \mathbf{S}_2)$ will not be a bipartite independent set with probability at least 1/2. Thus, $O(\log(1/\delta))$ bipartite independent set queries can simulate an independent set query with probability at least $1 - \delta$.

The problem of edge estimation using (bipartite) independent set queries shares resemblance to the classical problem of *group testing*, which dates back to 1943 [9] and has found many recent applications in computer science [28, 5, 10, 23, 21, 6, 18]. In group testing one needs to recover an unknown subset S of a known universe U by making *subset queries*: an algorithm can pick a subset T of U and ask whether T contains any element from S. The graph setting of the current paper is a natural generalization of group testing by considering the unknown object as a binary relation over a known universe U. The goal of estimating the number of edges, on the other hand, is a relaxation of group testing because it suffices to obtain an approximation of the size of the unknown binary relation, instead of recovering the relation itself exactly. The same relaxation on the original group testing setting (i.e., using subset queries to estimate the size of an unknown subset $S \subseteq U$) was studied by Ron and Tsur [27]. Besides group testing, edge estimation using independent set queries is motivated by connections to problems that arise in computational geometry and counting complexity, which we refer the interested reader to [3].

Perhaps surprisingly, [3] gave an algorithm that returns a $(1+\varepsilon)$-approximation to the number of edges by making only $\text{poly}(\log n, 1/\varepsilon)$ queries to the *bipartite* independent set oracle. So in this setting, the non-locality indeed brings down the query complexity significantly for the edge estimation problem (compared to [14] and [15], both of which use local queries only). For the independent set oracle, [3] obtained an algorithm for a $(1+\varepsilon)$-approximation of the number m of edges with query complexity $\min(\sqrt{m}, n^2/m) \cdot \text{poly}(\log n, 1/\varepsilon)$. It was left as an open problem in [3] to improve current understanding of edge estimation under independent set queries.

1.1 Our results

Theorem 1.1. (Upper bound) *There is a randomized algorithm that takes as input (1) an accuracy parameter $\varepsilon > 0$, (2) a positive integer n as the number of vertices and (3) access to the independent set oracle of an undirected graph $G = ([n], E)$ with $m = |E| \geq 1$.*[4] *With probability at least $1 - o(1)$, the algorithm makes no more than $\min(\sqrt{m}, n/\sqrt{m}) \cdot \text{poly}(\log n, 1/\varepsilon)$ many independent set queries and outputs a number $\widetilde{m}$ that satisfies $(1-\varepsilon)m \leq \widetilde{m} \leq (1+\varepsilon)m$.*

The improvement over the upper bound of [3] is due to a new algorithm for edge estimation that uses $(n/\sqrt{m}) \cdot \text{poly}(\log n, 1/\varepsilon)$ independent set queries (Theorem 3.1). Note that the query complexity achieved by the algorithm underlying Theorem 1.1 is essentially the same as [15]; however, the two algorithms access the graph with very different ways (independent set oracle versus degree and neighbor oracles). The proof of Theorem 1.1 requires new ideas and algorithmic techniques that are developed for independent set queries. See further discussion in Section 1.2.

Theorem 1.2. (Lower bound) *Let n and m be two positive integers with $m \leq \binom{n}{2}$. Any randomized algorithm with access to the independent set oracle of an undirected graph $G = ([n], E)$ must make at least $\min(\sqrt{m}, n/\sqrt{m})/\text{polylog}(n)$ queries in order to determine whether $|E| \leq m/2$ or $|E| \geq m$ with probability at least $2/3$.*

Theorems 1.1 and 1.2 essentially settle the query complexity of edge estimation with independent set queries at $\min(\sqrt{m}, n/\sqrt{m})$. Theorem 1.1 brings down the overall complexity of the problem from $n^{2/3}$ [3] to $\sqrt{n}$; the worst case is when the number of edges m is linear in n. Theorem 1.2, on the other hand, shows that no algorithm with independent set queries can achieve sub-polynomial query complexity. This gives an exponential separation between the power of the bipartite independent set oracle and the independent set oracle for the task of edge estimation.

1.2 Overview of techniques We first give a high-level overview of the lower bound because some key ideas from the lower bound will be helpful in understanding the main algorithm later. For convenience we will slightly abuse the notation $\widetilde{O}$ and $\widetilde{\Omega}$ to hide factors of $\text{poly}(\log n, 1/\varepsilon)$ in the discussion below. Outside of Section 1.2 they follow the convention described in footnote 1.

1.2.1 Lower bound We describe our construction for the case when $m \geq n$, where we seek a lower bound of $\widetilde{\Omega}(n/\sqrt{m})$. The complement case follows from a reduction to this case.

The plan is to follow Yao's principle. We construct two distributions $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{no}$ over graphs with vertices $[n]$ so that $\mathbf{G} \sim \mathcal{D}_{\text{yes}}$ has no more than $m/2$ edges with probability at least $1 - o(1)$ and $\mathbf{G} \sim \mathcal{D}_{\text{no}}$ has at least m edges with probability at least $1 - o(1)$. We then show that no deterministic algorithm with access to an independent set oracle can distinguish these two distributions.

A graph $\mathbf{G} \sim \mathcal{D}_{\text{yes}}$ is generated by first sampling a uniformly random partition of vertices into $(\mathbf{A}, \overline{\mathbf{A}})$ and

[4]The assumption of $m \geq 1$ is merely for convenience; it avoids the issue that the query complexity upper bound claimed would be 0 when $m = 0$. We note that whether a graph is empty or not can be determined by a single independent set query.

then forming the bipartite graph by including each pair (i,j) with $i \in \mathbf{A}$ and $j \in \overline{\mathbf{A}}$ as an edge independently with probability d/n, where $d \stackrel{\text{def}}{=} m/n$. In expectation $\mathbf{G} \sim \mathcal{D}_{\text{yes}}$ has about $m/4$ edges and thus, has no more than $m/2$ edges with probability $1-o(1)$. On the other hand, a graph $\mathbf{G} \sim \mathcal{D}_{\text{no}}$ is generated by sampling a uniformly random partition $(\mathbf{A},\overline{\mathbf{A}})$ of $[n]$, as well as a subset $\mathbf{B} \subseteq \mathbf{A}$ by including each vertex of $\mathbf{A}$ independently with probability $d \log n/n$. Similar to $\mathcal{D}_{\text{yes}}$, a pair (i,j) where $i \in \mathbf{A} \setminus \mathbf{B}$ and $j \in \overline{\mathbf{A}}$ is included as an edge independently with probability d/n. The main difference compared to $\mathcal{D}_{\text{yes}}$ is that every pair (i,j), where $i \in \mathbf{B}$ and $j \in \overline{\mathbf{A}}$, is included as an edge (so $(\mathbf{B},\overline{\mathbf{A}})$ form a complete bipartite graph). Given that $|\overline{\mathbf{A}}| = \Omega(n)$ and $|\mathbf{B}| = \Omega(d \log n)$ with high probability, the number of edges in the graph is $\Omega(dn \log n) = \Omega(m \log n) \geq m$ with probability at least $1-o(1)$.

We make the following two observations. The first is that a graph $\mathbf{G} \sim \mathcal{D}_{\text{no}}$ can be generated by first drawing a graph $\mathbf{G}' \sim \mathcal{D}_{\text{yes}}$ with partition $(\mathbf{A},\overline{\mathbf{A}})$, then sampling $\mathbf{B} \subseteq \mathbf{A}$ by including each vertex in $\mathbf{A}$ independently with probability $d \log n/n$, and finally adding all pairs between $\mathbf{B}$ and $\overline{\mathbf{A}}$ as edges in $\mathbf{G}$. This suggests that, in order for an algorithm to distinguish $\mathcal{D}_{\text{no}}$ from $\mathcal{D}_{\text{yes}}$, a (seemingly quite weak) necessary condition is for one of its queries to overlap with $\mathbf{B}$ when it runs on $\mathbf{G} \sim \mathcal{D}_{\text{no}}$.

For the second observation, we consider a query set $S \subseteq [n]$ of size larger than $(n/\sqrt{m}) \cdot \log n$. In both $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$, we have $|S \cap \mathbf{A}|, |S \cap \overline{\mathbf{A}}| \geq \Omega((n/\sqrt{m}) \cdot \log n)$ with high probability and when this happens, S is not an independent set with high probability, given that there are at least

$$\Omega\left((n^2/m) \cdot \log^2 n\right) = \Omega\left((n/d) \cdot \log^2 n\right)$$

pairs between $S \cap \mathbf{A}$ and $S \cap \overline{\mathbf{A}}$ and each is included in the graph with probability d/n. Since S is not an independent set in both $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$ with high probability, such a query conveys very little information in distinguishing the two distributions. Thus, a reasonable algorithm should only make queries of size smaller than $(n/\sqrt{m}) \cdot \log n$. This intuition, that algorithms should not make queries of size larger than $n/\sqrt{m}$, will be helpful in our discussion of the algorithm later, and we will frequently refer to the quantity $n/\sqrt{m}$ as the *critical threshold*. However, if all the queries an algorithm makes are smaller than $(n/\sqrt{m}) \cdot \log n$, then $\widetilde{\Omega}(n/\sqrt{m})$ queries are necessary for at least one of them to overlap with $\mathbf{B}$; otherwise, given that $|\mathbf{B}| = O(d \log n)$, the probability that one of the queries overlaps with $\mathbf{B}$ is negligible.

To formalize the above intuition and simplify the presentation of our lower bound proof, we introduce the notion of an *augmented* (independent set) oracle in Section 5.2. We first show that any algorithm with access to the standard independent set oracle can be simulated using an augmented oracle with the same query complexity. Then, we prove an $\widetilde{\Omega}(n/\sqrt{m})$ lower bound for algorithms that distinguish $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$ with access to an augmented oracle.

1.2.2 Upper Bound Our goal is to obtain a $(1+\varepsilon)$-approximation algorithm for edge estimation with $\widetilde{O}(n/\sqrt{m})$ independent set queries, where m denotes the number of edges of the input graph (Theorem 3.1). Theorem 1.1 follows by combining it with the algorithm of [3] by running both algorithms in parallel and outputting the result of whichever finishes first.

In the sketch of the algorithm below, we assume that a rough estimate $\overline{m}$ of the number of edges m is given, satisfying $m = \Theta(\overline{m})$. The goal is to refine it to obtain a $(1+\varepsilon)$-approximation $\widetilde{m}$ of m.

An Initial Plan: At a high level, we partition the vertex set $[n]$ into $O((\log n)/\varepsilon)$ many buckets according to their degrees: a vertex $u \in [n]$ belongs to the ith bucket B_i if $\deg(u)$ is between $(1+\varepsilon)^i$ and $(1+\varepsilon)^{i+1}$. We refer to $(1+\varepsilon)^i$ as the degree of bucket B_i for convenience. Our initial plan is to develop efficient algorithms for the following two tasks:

- **Task 1**: Develop a subroutine that, given a vertex u and an index i, checks if u belongs to B_i.[5]
- **Task 2**: Use the first subroutine to estimate the size of each bucket B_i.

We point out that this initial plan looks very similar to the framework of the algorithm of [15], where ideally one would like to estimate the size of each B_i by drawing enough random samples and running the subroutine in Task 1 on each sample to obtain an estimate of $|B_i|$. The similarity, however, stops here as we start discussing more details about how to implement the plan with an independent set oracle.

We consider Task 1 first (which is trivial with a degree oracle). Note that when $d \geq \sqrt{m}$, checking whether a vertex u has $\deg(u) \geq (1+\varepsilon)d$ or $\deg(u) \leq d$ requires $\widetilde{O}(1)$ independent set queries. As a result, it requires $\widetilde{O}(1)$ to tell if $u \in B_i$ when the degree of B_i is at least $\sqrt{m}$. The bad news is that the same task

[5]The goal of the subroutine as described above may not sound reasonable. If $\deg(u)$ lies very close to the boundary of two buckets B_i and B_{i+1}, determining which of the two buckets u lies in may be expensive with independent set queries. This is indeed one source of errors we need to handle. We focus on high-level ideas behind the algorithm and skip details such as errors most of time, and discuss briefly how we analyze the algorithm in the presence of errors at the end of the sketch.

becomes significantly more challenging as d goes down from $\sqrt{\overline{m}}$. This challenge leads to a major revision of our initial plan.

To gain some intuition we consider the task of distinguishing $\deg(u) \geq (1+\varepsilon)d$ and $\deg(u) \leq d$ when $d \gg \sqrt{\overline{m}}$.[6] Suppose we sample a set $\mathbf{T}$ from $[n] \setminus \{u\}$ by including each vertex with probability $1/d$ and then make two independent set queries on $\mathbf{T}$ and $\mathbf{T} \cup \{u\}$. Let $\mathcal{E}$ denote the event that $\mathbf{T}$ is an independent set but $\mathbf{T} \cup \{u\}$ is not (so $\mathbf{T}$ contains at least one neighbor of u). Then we claim that there is a significant gap in the probability of $\mathcal{E}$ when $\deg(u) \geq (1+\varepsilon)d$ versus $\deg(u) \leq d$. This gap in the probability of $\mathcal{E}$ is large enough so that one can repeat the experiment $\widetilde{O}(1)$ times (each time making two independent set queries) to distinguish the two cases with high probability.

Now we turn to the case when $d \ll \sqrt{\overline{m}}$. In this case, the algorithm is limited to query sets $\mathbf{T}$ of size much smaller than n/d. Therefore, we limit $\mathbf{T}$ to include each vertex with probability $1/\sqrt{\overline{m}}$ instead of $1/d$. Two issues arise. The first (minor) issue is that, given that the size of $\mathbf{T}$ is roughly $n/\sqrt{\overline{m}}$, even to hit a neighbor of u (with degree roughly d) one needs to draw $\mathbf{T}$ at least $\sqrt{\overline{m}}/d$ many times. This suggests that $\sqrt{\overline{m}}/d$ queries are needed for Task 1 when the degree d of the bucket we are interested is less than $\sqrt{\overline{m}}$.

There is, however, a more serious issue that is subtle but leads to a major revision of the initial plan. Consider the scenario where u has $(1+\varepsilon)d$ neighbors and every neighbor has degree $\gg \sqrt{\overline{m}}$. If we sample $\mathbf{T}$ by including each vertex with probability $1/\sqrt{\overline{m}}$, it is very unlikely that $\mathbf{T}$ contains a neighbor of u but $\mathbf{T}$ is at the same time independent (since when conditioning on $\mathbf{T}$ containing a neighbor v of u, most likely $\mathbf{T}$ also contains a neighbor of v given the large degree of v). Because of the second issue, we change the goal of the subroutine in Task 1 from finding the right bucket of u according to the degree of u to finding the right bucket according to the *number of neighbors of u with degree at most* $\sqrt{\overline{m}}$, when $\deg(u) < \sqrt{\overline{m}}$. For vertices with degree at least $\sqrt{\overline{m}}$, we still would like to partition them into buckets according to their degrees.

A Revised Plan: By the above, we arrived at the following revised plan:

Task 0: Develop a subroutine that, given a vertex u, decides[7] if $\deg(u) \geq \sqrt{\overline{m}}$ (which we refer to as high-degree vertices and denote the set by H) or $\deg(u) < \sqrt{\overline{m}}$ (which we refer to as low-degree vertices and denote the set by L). High-degree vertices are further partitioned into buckets H_i according to their degrees. Low-degree vertices, on the other hand, are partitioned into buckets L_i according to their degrees to low-degree vertices, denoted by $\deg(u, L)$ for a vertex u.

Task 1: Develop a subroutine that, given a vertex $u \in H$ (or $u \in L$) and an index i, decides if u belongs to the bucket H_i (or L_i).

Task 2: Use the two subroutines to obtain $(1+\varepsilon)$-estimations of the size of each L_i and H_i.

Looking ahead, with $(1+\varepsilon)$-approximations ℓ_i and h_i for $|L_i|$ and $|H_i|$, one can compute

$$\textstyle\sum_i \ell_i \cdot (1+\varepsilon)^i + \sum_i h_i \cdot (1+\varepsilon)^i$$

as roughly a 2-approximation of the number of edges m. The reason that we only get 2-approximation follows by the fact that in the sum, edges between vertices in L and edges between vertices in H are counted twice but edges between L and H are only counted once. We will discuss more about how to further revise the plan to obtain a $(1+\varepsilon)$-approximation; for now let us consider Task 2.

Note that Task 2 for buckets L_i is easy. Consider a low-degree bucket L_i with $d = (1+\varepsilon)^i \leq \sqrt{\overline{m}}$. Unless $|L_i| = \Omega(\overline{m}/d)$, L_i has negligible impact on the final estimate. When $|L_i| = \Omega(\overline{m}/d)$, it takes $\widetilde{O}(nd/\overline{m})$ samples to get a sufficient number of vertices in L_i. We can then get a good estimation of $|L_i|$ by running subroutines for Task 0 and 1 on these vertices. We pay $\widetilde{O}(\sqrt{\overline{m}}/d)$ queries for each vertex so the overall query complexity is

$$\widetilde{O}(nd/\overline{m}) \cdot \widetilde{O}(\sqrt{\overline{m}}/d) = \widetilde{O}(n/\sqrt{\overline{m}})$$

as desired. In contrast, uniformly sampling vertices and checking individually if each of them lies in H_i is too inefficient for high-degree buckets, given that $nd/\overline{m} \gg n/\sqrt{\overline{m}}$ when $d \gg \sqrt{\overline{m}}$.

Estimating the size of each high-degree bucket H_i is where we fully take advantage of the *non-locality* of independent set queries. To explain the intuition, let us consider the task of distinguishing $|H_i| \geq (1+\varepsilon)r$ versus $|H_i| \leq r$ for some parameter $r = \Theta(\overline{m}/d)$ where $d = (1+\varepsilon)^i \gg \sqrt{\overline{m}}$ denotes the degree of the bucket H_i. To this end, it suffices to have a procedure that can take a random set $\mathbf{S} \subseteq [n]$ of size $n/(\sqrt{\overline{m}} \log n)$ and answers the question "does there exist $u \in \mathbf{S}$ that belongs to H_i?" with $\widetilde{O}(1)$ queries. With such a procedure it suffices to draw $\mathbf{S}$ and run the procedure on $\mathbf{S}$ for

$$\widetilde{O}\left(\frac{n}{(n/(\sqrt{\overline{m}}\log n)) \cdot (\overline{m}/d)}\right) = \widetilde{O}\left(\frac{d}{\sqrt{\overline{m}}}\right) \leq \widetilde{O}\left(\frac{n}{\sqrt{\overline{m}}}\right)$$

[6] For convenience we consider the case of $d \gg \sqrt{m}$ in the sketch but the same idea works when $d \geq \sqrt{m}$.

[7] Again we need to handle errors when $\deg(u)$ is close to $\sqrt{\overline{m}}$.

many times in order to obtain a good estimation of $|H_i|$.

As discussed earlier, the revised plan ultimately leads to a $(2+\varepsilon)$-approximation algorithm with $\widetilde{O}(n/\sqrt{m})$ independent set queries. We achieve $(1+\varepsilon)$-approximation by revising the plan further. First we divide high-degree vertices u into buckets $H_{i,j}$ where i is related to the degree of u (as usual), but the second index j is related to the fraction of neighbors of u in L; see Definition 3.3 for details. Task 1 is updated to develop a subroutine that can decide whether u belongs to $H_{i,j}$ or not. Task 2 is updated to estimate the size of each $H_{i,j}$ (with similar ideas in the approximation of $|H_i|$ sketched above) and L_i. Together they lead to a $(1+\varepsilon)$-approximation of the number of edges between low-degree and high-degree vertices, and ultimately a $(1+\varepsilon)$-approximation of m.

Now extra care must be taken to handle errors when executing the above plan. As alerted in two footnotes, one cannot hope for a subroutine that returns the true bucket of a vertex u. To simplify the presentation of the algorithm and its analysis, we introduce the notion of $(\overline{m},\varepsilon)$-*degree oracles* (see Definition 3.5). An $(\overline{m},\varepsilon)$-degree oracle can answer questions listed in Tasks 0 and 1 consistently and accurately up to certain errors (as captured by the notion of an $(\overline{m},\varepsilon)$-*degree partition* in Definition 3.3 underlying each $(\overline{m},\varepsilon)$-degree oracle). We first present an algorithm in Section 3.3 that has query access to a $(\overline{m},\varepsilon)$-degree oracle. We finish the proof of Theorem 3.1 by giving an efficient implementation of a $(\overline{m},\varepsilon)$-degree oracle using an independent set oracle in Section 4.

2 Preliminaries

Given a positive integer n, we write $[n]$ to denote $\{1,\ldots,n\}$. Similarly, for two non-negative integers $i\le j$, we write $[i:j]$ to denote $\{i,\ldots,j\}$. All graphs considered in this paper are undirected and simple (meaning that there are no parallel edges or loops), and have $[n]$ as its vertex set.

Definition 2.1. (Independent set oracle) *Given an undirected graph $G=([n],E)$, its* independent set oracle *is a map $\mathsf{IS}_G\colon 2^{[n]}\to\{0,1\}$ which satisfies that for any set of vertices $U\subseteq[n]$, $\mathsf{IS}_G(U)=1$ if and only if U is an independent set of G (i.e., $(u,v)\notin E$ for all $u,v\in U$).*

We use $\deg_G(v)$ to denote the degree of a vertex $v\in[n]$. Given $v\in[n]$ and $U\subseteq[n]$, we let $\Gamma_G(v,U)=\big\{u\in U:(u,v)\in E\big\}$ and $\deg_G(v,U)\stackrel{\text{def}}{=}\big|\Gamma_G(v,U)\big|$. Note that v can lie in U, but since we only consider simple graphs, $\Gamma_G(v,U)=\Gamma_G(v,U\setminus\{v\})$. For the sake of brevity, we write $\Gamma_G(v)=\Gamma_G(v,[n])$. We usually skip the subscript in IS_G,Γ_G and $\deg_G$ when the underlying graph G is clear from the context.

Subroutine `Binary-Search`(n,G,T,δ)

Input: A positive integer n, access to the independent set oracle of a graph $G=([n],E)$, a set $T\subseteq[n]$ with a promise that T is not an independent set of G, and an error parameter $\delta>0$.
Output: An edge $(u,v)\in E$ with $u,v\in T$, or "fail."

1. Let $\mathbf{A}\leftarrow T$.
2. Repeat the following for $t=O(\log n+\log(1/\delta))$ iterations:
   (a) If $|\mathbf{A}|=2$, output the two vertices in $\mathbf{A}$
   (b) Randomly partition $\mathbf{A}$ into $\mathbf{A}_1\cup\mathbf{A}_2$ where $|\mathbf{A}_1|$ and $|\mathbf{A}_2|$ differ by at most 1. Query $\mathsf{IS}_G(\mathbf{A}_1)$ and $\mathsf{IS}_G(\mathbf{A}_2)$ to see if one of them is not an independent set. If $\mathbf{A}_b$ is not an independent set for some $b\in\{1,2\}$, set $\mathbf{A}\leftarrow\mathbf{A}_b$.
3. Output "fail".

Figure 1: Description of the `Binary-Search` subroutine.

The following simple lemma will be used multiple times.

Lemma 2.2. *Let $G=([n],E)$ be an undirected graph, $S\subseteq[n]$ be a set of vertices, and $r\in\mathbb{N}$ be an upper bound on the number of edges in the subgraph induced by S. Let $\mathbf{T}\subseteq S$ be a random subset given by independently including each vertex of S with probability p. Then,*

$$\Pr_{\mathbf{T}\subseteq S}\big[\mathbf{T}\text{ is an independent set of }G\,\big]\ge 1-rp^2.$$

Proof: The expected number of edges where both vertices lie in $\mathbf{T}$ is at most rp^2. By Markov's inequality the probability that $\mathbf{T}$ contains at least one edge is at most rp^2. ■

2.1 Binary search using the independent set oracle

We present a subroutine based on binary search for finding an edge using independent set queries:

Lemma 2.3. *There is a randomized algorithm,* `Binary-Search`(n,G,T,δ)*, that takes as input (1) a positive integer n, (2) access to the independent set oracle IS_G of an undirected graph $G=([n],E)$,* *(3) a set $T \subseteq [n]$ of vertices such that T is not an independent set of G, and (4) an error parameter $\delta > 0$.* `Binary-Search` *makes $O(\log n + \log(1/\delta))$ queries to IS_G and outputs $u, v \in T$ with $(u, v) \in E$ with probability at least $1 - \delta$.*

REMARK 2.4. *We will always invoke* `Binary-Search` *with the parameter $\delta = 1/\text{poly}(n)$.*[8] *The subroutine will always make $O(\log n)$ queries, and will fail with probability at most $1/\text{poly}(n)$.*

3 Upper bound

In this section we prove the following upper bound:

THEOREM 3.1. *There is a randomized algorithm* `Estimate-Edges`(ε, n, G) *that takes as input (1) an accuracy parameter $\varepsilon \in (0, 1)$, (2) a positive integer n, and (3) access to the independent set oracle of a graph $G = ([n], E)$ with $m = |E| \geq 1$. With probability at least $1 - o(1)$,* `Estimate-Edges` *makes $(n/\sqrt{m}) \cdot \text{poly}(\log n, 1/\varepsilon)$ queries and outputs a number $\widetilde{m}$ satisfying $(1 - \varepsilon)m \leq \widetilde{m} \leq (1 + \varepsilon)m$.*

We recall the following lemma from [3].

LEMMA 3.1. (LEMMA 5.6 FROM [3]) *There is a randomized algorithm that takes as input (1) an accuracy parameter $\varepsilon \in (0, 1)$, (2) a positive integer n, and (3) access to the independent set oracle of a graph $G = ([n], E)$ with $m = |E| \geq 1$. With probability at least $1 - o(1)$, the algorithm makes $\sqrt{m} \cdot \text{poly}(\log n, 1/\varepsilon)$ queries and outputs a number $\widetilde{m}$ satisfying $(1 - \varepsilon)m \leq \widetilde{m} \leq (1 + \varepsilon)m$.*

The upper bound claimed in Theorem 1.1 of $\min\{\sqrt{m}, n/\sqrt{m}\} \cdot \text{poly}(\log n, 1/\varepsilon)$ follows by running the algorithm of Theorem 3.1 and the algorithm of Lemma 3.1 in parallel. Specifically, we alternate queries between the two algorithms until one of them terminates. Once one terminates with an estimate $\widetilde{m}$ to m, we output $\widetilde{m}$.

3.1 Reduction to edge estimation with advice

We prove Theorem 3.1 using the following lemma stated next. We will provide an algorithm, which we call `Estimate-With-Advice`, for estimating $|E|$ given an extra parameter $\overline{m}$ which is promised to be an upper bound for $|E|$.

LEMMA 3.2. (ESTIMATION WITH ADVICE) *There is a randomized algorithm,* `Estimate-With-Advice`*, that takes four inputs: (1) an accuracy parameter $\varepsilon \in (0, 1)$, (2) two positive integers $n, \overline{m}$, and (3) access to an independent set oracle of $G = ([n], E)$ with $1 \leq m = |E| \leq \overline{m}$.* `Estimate-With-Advice` *makes $(n/\sqrt{\overline{m}}) \cdot \text{poly}(\log n, 1/\varepsilon)$ queries and with probability at least $1 - 1/n$ outputs $\hat{m}$ that satisfies*

$$(3.1) \qquad (1 - 5\varepsilon)m - O\left(\frac{\varepsilon \overline{m}}{\log n}\right) \leq \hat{m} \leq (1 + \varepsilon)m.$$

[8] For example, setting $\delta = 1/n^{10}$ will suffice for our purposes.

Algorithm `Estimate-Edges` (ε, n, G)

Input: An accuracy parameter $\varepsilon \in (0, 1)$, a positive integer n, and access to the independent set oracle of an undirected graph $G = ([n], E)$.
Output: A number $\widetilde{m}$ as an estimation of $m = |E|$.

1. Set $\overline{m} = \binom{n}{2}$.
2. While $\overline{m} \geq 1$:
   (a) Invoke `Estimate-With-Advice`$(\epsilon/11, n, \overline{m}, G)$.
   (b) Let $\hat{m}$ denote the output. If $4\hat{m} \geq \overline{m}$ **return** $\hat{m}$ as $\widetilde{m}$; otherwise set $\overline{m}$ to be $\lfloor \overline{m}/2 \rfloor$.
3. **Return** 0 as $\widetilde{m}$ (this line is reached with low probability).

Figure 2: Description of the `Estimate-Edges` algorithm.

Before proving Lemma 3.2, we show that it implies Theorem 3.1.

Proof of Theorem 3.1 Assuming Lemma 3.2: We present `Estimate-Edges` in Figure 2.

Note that at the end of each iteration of step 2 in Figure 2, either the algorithm terminates or $\overline{m}$ is halved. Since $\overline{m}$ is initially $\binom{n}{2}$, the maximum number of iterations of the step 2 (before $\overline{m} < 1$) in `Estimate-Edges` is $O(\log n)$. It follows from Lemma 3.2 and a union bound that, with probability at least $1 - o(1)$, every execution of `Estimate-With-Advice` in step 2(a) of `Estimate-Edges` returns a *correct* value (meaning that if $\overline{m}$ of this run indeed satisfies $\overline{m} \geq m = |E|$, then its output $\hat{m}$ satisfies (3.1) but with ε set to $\varepsilon/11$). We show that the following holds when this is the case:

> $(*)$: `Estimate-Edges` terminates in the while loop (instead of going to line 3)
> with the final value of $\overline{m}$ satisfying $m \leq \overline{m} \leq 5m$.

Assume that $(*)$ holds, and let $\widetilde{m}$ be the output of `Estimate-Edges`(ε, n, G). Since $m \leq \overline{m}$ in every iteration of step 2 and the final iteration also satisfies

$\overline{m} \le 5m$, Theorem 3.1 would follow from two observations.(i) The query complexity of `Estimate-Edges` can be bounded using $\overline{m} \ge m$, and (ii) since the final run of `Estimate-With-Advice` is correct, we have (using $\overline{m} \le 5m$)

$$(1-\varepsilon/2)m < (1-5\varepsilon/11)m - O\left(\frac{\varepsilon m}{\log n}\right) \le \widetilde{m} \le (1+\varepsilon/11)m < (1+\varepsilon)m. \tag{3.2}$$

It suffices to show that (∗) holds when every run of `Estimate-With-Advice` returns a correct value.

Assuming for contradiction of (∗) that the final value of $\overline{m}$ is smaller than m. This implies that $m \le \overline{m} \le 2m$ in one of the runs of `Estimate-With-Advice` in `Estimate-Edges`. Since it returns a correct value $\hat{m}$ (and note that for this run we still have $m \le \overline{m}$), the same calculation in (3.2) implies that $\hat{m} \ge (1-\varepsilon/2)m$ and thus, $4\hat{m} \ge 4 \cdot 0.5 \cdot m = 2m \ge \overline{m}$ and the algorithm should have terminated at the end of this run, a contradiction. On the other hand, assume for a contradiction of (∗) that the final value of $\overline{m}$ is larger than $5m$. Since the final run returns a correct value $\hat{m}$, $\hat{m} \le (1+\varepsilon/11)m \le 1.1m$ and thus, $4\hat{m} < 5m < \overline{m}$; however, step 2(b) should have terminated if $4\hat{m} \ge \overline{m}$, a contradiction. This finishes the proof of the theorem. ■

We prove Lemma 3.2 in the rest of the section. From now on, let $\varepsilon \in (0,1)$ be the accuracy parameter, $\overline{m} \le \binom{n}{2}$ be a positive integer, and $G = ([n], E)$ be a graph with $1 \le m = |E| \le \overline{m}$ as in the statement of Lemma 3.2. Let $\alpha = 1+\varepsilon$ and let s be the unique positive integer such that

$$\alpha^{s-1} \le \sqrt{\overline{m}} < \alpha^s. \tag{3.3}$$

We also write $\beta = \Theta((\log n)/\varepsilon)$ to denote the smallest integer such that $\alpha^\beta \ge n$, and τ to denote the smallest integer such that $\alpha^\tau \ge \log n/\varepsilon$ (so $\alpha^\tau = \Theta(\log n/\varepsilon)$). It may be helpful to the reader to consider the case when $\overline{m}$ is only a constant factor larger than $|E|$, so the algorithm's task is to refine an approximation to the number of edges given a crude approximation; however, the proof of Lemma 3.2 assumes just the upper bound $\overline{m} \ge |E|$.

3.2 Degree oracles and the high-level plan

To simplify the presentation and analysis of our algorithm, `Estimate-With-Advice`, we introduce the notion of $(\overline{m},\varepsilon)$-*degree partitions* and $(\overline{m},\varepsilon)$-*degree oracles.* Roughly speaking, an $(\overline{m},\varepsilon)$-degree partition $P = (L_i, H_{k,\ell} : i \in [0:s], k \in [s+1:\beta] \text{ and } \ell \in [0:\tau])$ of an undirected graph $G = ([n], E)$ is a partition of $[n]$ (so L_i's and $H_{k,\ell}$'s are pairwise disjoint subsets of $[n]$ whose union is $[n]$) such that the placement of a vertex v reveals important degree information of v (see Definition 3.3 for details). An $(\overline{m},\varepsilon)$-degree oracle, on the other hand, contains an underlying $(\overline{m},\varepsilon)$-degree partition and the latter can be accessed via queries such as "does v belong to L_i" or "does v belong to $H_{k,\ell}$." There is also a cost associated with each such query (see Definition 3.5 for details).

With the definition of degree partitions and degree oracles, our proof of Lemma 3.2 proceeds in the following two steps. First we present in Lemma 3.6 an algorithm `Estimate-With-Advice`* that achieves the same goal as `Estimate-With-Advice`, namely (3.1) in Lemma 3.2 with high probability. The difference, however, is that `Estimate-With-Advice`* is given access to not only an independent set oracle but also an $(\overline{m},\varepsilon)$-degree oracle. Next, we show in Lemma 3.7 that an $(\overline{m},\varepsilon)$-degree oracle can be implemented efficiently using access to the independent set oracle. This allows us to convert `Estimate-With-Advice`* into `Estimate-With-Advice` with a similar performance guarantee, and Lemma 3.2 follows directly from Lemma 3.6 and Lemma 3.7.

We start with the definition of $(\overline{m},\varepsilon)$-degree partitions:

Definition 3.3. *Let $G = ([n], E)$ be a graph. An* $(\overline{m},\varepsilon)$-degree partition *of G is a partition*

$$P = \Big(L_i, H_{k,\ell} : i \in [0:s], k \in [s+1:\beta] \text{ and } \ell \in [0:\tau]\Big)$$

of its vertex set $[n]$ (so the sets in P are disjoint and their union is $[n]$) such that

1. *Let $L = \cup_i L_i$ and $H = \cup_{k,\ell} H_{k,\ell}$ (so we have $L \cup H = [n]$). Every vertex $u \in L$ satisfies $\deg(u) \le \alpha^{s+1}$ and every vertex $u \in H$ satisfies $\deg(u) \ge \alpha^s$.*
2. *Every vertex $u \in L_0$ satisfies $\deg(u, L) = 0$ and every vertex $u \in L_i$, $i \in [s]$, satisfies*
$$\alpha^{i-1} \le \deg(u, L) \le \alpha^{i+1}. \tag{3.4}$$
3. *Let $H_k = \cup_\ell H_{k,\ell}$ for each $k \in [s+1:\beta]$. Then every vertex $u \in H_k$ satisfies*
$$\alpha^{k-1} \le \deg(u) \le \alpha^{k+1}. \tag{3.5}$$
Moreover, every vertex $u \in H_{k,\ell}$ for some $\ell \in [0:\tau-1]$ satisfies
$$\alpha^{k-\ell-1} \le \deg(u, L) \le \alpha^{k-\ell+1} \tag{3.6}$$
and every $u \in H_{k,\tau}$ satisfies $\deg(u, L) \le \alpha^{k-\tau+1}$.

REMARK 3.4. *It is worth pointing out that intervals used in* (3.4)*,* (3.5)*, and* (3.6) *are not disjoint* (*and so are the conditions on* $\deg(u)$ *in the first item*)*. As a result, such partitions are not unique for a given* G *in general. For example, a vertex with degree between* α^s *and* α^{s+1} *can lie in either* L *or* H*.*

Next we define $(\overline{m}, \varepsilon)$-degree oracles:

DEFINITION 3.5. *Let* $G = ([n], E)$ *be an undirected graph. An* $(\overline{m}, \varepsilon)$-degree oracle $D = (D_{low}, D_{high})$ *of* G *contains an underlying* $(\overline{m}, \varepsilon)$*-degree partition* $P = (L_i, H_{k,\ell} : i, k, \ell)$ *of* G *and can be accessed via two maps* $D_{high} : [n] \times [s+1 : \beta] \times [0 : \tau] \to \{0, 1\}$ *and* $D_{low} : [n] \times [0 : s] \to \{0, 1\}$*, where*

1. *For every vertex* $u \in [n]$*,* $D_{high}(u, k, \ell) = 1$ *if* $u \in H_{k,\ell}$ *and* $D_{high}(u, k, \ell) = 0$ *otherwise.*
2. *For every vertex* $u \in [n]$*,* $D_{low}(u, i) = 1$ *if* $u \in L_i$ *and* 0 *otherwise.*

The cost *of each query on* D_{high} *is* 1 *and the cost of each query* $D_{low}(u, i)$ *is* α^{s-i}*.*

We will be interested in algorithms that have access to both the independent set oracle IS_G and an $(\overline{m}, \varepsilon)$-degree oracle D of a graph $G = ([n], E)$. For such an algorithm `Alg`* (for clarity we always use $*$ to mark algorithms that have access to such a pair of oracles), we are interested in its *total cost*. The cost of each query on the independent set oracle is 1, and the cost of each query on the degree oracle is specified in Definition 3.5. The total cost of an algorithm is the sum of the costs of individual queries.

We are ready to state Lemma 3.6 and Lemma 3.7 which together imply Lemma 3.2.

LEMMA 3.6. (ESTIMATION WITH DEGREE ORACLES) *There is a randomized algorithm,* `Estimate-With--Advice`$^*(\epsilon, n, \overline{m}, G)$*, that takes four inputs: an accuracy parameter* $\varepsilon \in (0, 1)$*, two positive integers* n *and* $\overline{m}$*, and access to both the independent set oracle* IS_G *and an* $(\overline{m}, \varepsilon)$*-degree oracle* D *of a graph* $G = ([n], E)$ *with* $1 \le m = |E| \le \overline{m}$*. Its worst-case total cost is* $(n/\sqrt{\overline{m}}) \cdot \mathrm{poly}(\log n, 1/\varepsilon)$ *and with probability at least* $1 - 1/n^2$*, it returns* $\hat{m}$ *satisfying*

$$(3.7)\qquad (1 - 5\varepsilon)m - O\left(\frac{\varepsilon\overline{m}}{\log n}\right) \le \hat{m} \le (1 + \varepsilon)m.$$

We point out that, because $(\overline{m}, \varepsilon)$-degree partitions are *not* unique, `Estimate-With-Advice`* in Lemma 3.6 needs to work with an $(\overline{m}, \varepsilon)$-degree oracle with *any* underlying $(\overline{m}, \varepsilon)$-degree partition (as long as it satisfies Definition 3.3). Lemma 3.7 below says that one can simulate a degree oracle efficiently using the independent set oracle.

LEMMA 3.7. (SIMULATION OF DEGREE ORACLES) *Let* $\varepsilon \in (0, 1)$ *and* $n, \overline{m}$ *be positive integers. There are a positive integer* $q = q(\varepsilon, n, \overline{m})$ *and a pair of* deterministic *algorithms* $\texttt{Sim-D}_{\texttt{low}}$ *and* $\texttt{Sim-D}_{\texttt{high}}$*, where* $\texttt{Sim-D}_{\texttt{low}}(v, i, G, r)$ *takes as input a vertex* $v \in [n]$*,* $i \in [0 : s]$*, access to the independent set oracle of a graph* $G = ([n], E)$ *with* $1 \le |E| \le \overline{m}$*, and a string* $r \in \{0, 1\}^q$*;* $\texttt{Sim-D}_{\texttt{high}}(v, k, \ell, G, r)$ *takes the same inputs but has* i *replaced by* $k \in [s+1 : \beta]$ *and* $\ell \in [0 : \tau]$*. Both algorithms output a value in* $\{0, 1\}$ *and together have the following performance guarantee:*

1. $\texttt{Sim-D}_{\texttt{low}}(v, i, G, r)$ *makes* $\alpha^{s-i} \cdot \mathrm{poly}(\log n, 1/\varepsilon)$ *queries to* IS_G *and* $\texttt{Sim-D}_{\texttt{high}}(v, k, \ell, G, r)$ *makes* $\mathrm{poly}(\log n, 1/\varepsilon)$ *queries to* IS_G*.*
2. *Given any graph* G *with* $1 \le |E| \le \overline{m}$*, when* $\mathbf{r} \sim \{0, 1\}^q$ *is drawn uniformly at random,* $\texttt{Sim-D}_{\texttt{low}}(v, i, G, \mathbf{r})$ *viewed as a map from* $[n] \times [0 : s] \to \{0, 1\}$ *and* $\texttt{Sim-D}_{\texttt{high}}(v, k, \ell, G, r)$ *viewed as a map from* $[n] \times [s+1 : \beta] \times [0 : \tau] \to \{0, 1\}$ *together form an* $(\overline{m}, \varepsilon)$*-degree oracle of* G *with probability at least* $1 - 1/n^2$ (*over the randomness of* $\mathbf{r}$)*.*

We use Lemma 3.6 and 3.7 to prove Lemma 3.2.

Proof of Lemma 3.2 Assuming Lemma 3.6 and 3.7: The algorithm `Estimate-With-Advice` $(\varepsilon, n, \overline{m}, G)$ draws a string $\mathbf{r} \sim \{0, 1\}^q$ uniformly at random, where $q = q(\varepsilon, n, \overline{m})$ as in Lemma 3.7, and simulates `Estimate-With-Advice`*. When the latter makes a query on its given degree oracle, `Estimate-With-Advice` runs either $\texttt{Sim-D}_{\texttt{low}}$ or $\texttt{Sim-D}_{\texttt{high}}$ using $\mathbf{r}$ and uses its output to continue the simulation of `Estimate-With-Advice`. The query complexity of `Estimate-With-Advice` can be bounded using the total cost of `Estimate-With-Advice`* and complexity of $\texttt{Sim-D}_{\texttt{low}}$ and $\texttt{Sim-D}_{\texttt{high}}$. The error probability of `Estimate-With-Advice` is at most $1/n^2$ (for the probability that $\mathbf{r}$ fails to produce an $(\overline{m}, \varepsilon)$-degree oracle) plus $1/n^2$ (for the error probability of `Estimate-With-Advice`*), which is smaller than $1/n$. This finishes the proof of Lemma 3.2. ■

We prove Lemma 3.6 in the rest of this section and then prove Lemma 3.7 in Section 4.

3.3 Estimation of $|L_i|$ and $|H_{k,\ell}|$.

Let $G = ([n], E)$ be the input graph with $1 \le m = |E| \le \overline{m}$. We are given access to the independent set oracle IS_G and an $(\overline{m}, \varepsilon)$-degree oracle $\mathsf{D} = (\mathsf{D}_{\text{low}}, \mathsf{D}_{\text{high}})$ of G, where we use $P = (L_i, H_{k,\ell} : i, k, \ell)$ to denote the degree partition underlying the degree oracle D. To obtain a good estimation of $|E|$, it suffices to obtain good estimations of cardinalities of L_i's and $H_{k,\ell}$'s (the latter would also lead to good estimations of $|H_k|$; recall that $H_k = \cup_\ell H_{k,\ell}$). Roughly speaking, estimations of $|L_i|$'s allow us to approximately count the number of edges in the subgraph induced by L; estimations of $|H_k|$'s allow us to approximately count the total degree of vertices in H; estimations of $|H_{k,\ell}|$'s allow us to approximately count the number of edges between L and H.

We describe two subroutines for estimating $|L_i|$ and $|H_{k,\ell}|$ in Lemma 3.8 and 3.9, respectively, and then use them to prove Lemma 3.6.

LEMMA 3.8. (ESTIMATION OF $|L_i|$) *Let $\varepsilon \in (0,1)$ and $\overline{m}$ be a positive integer. There is a randomized algorithm that runs on graphs $G = ([n], E)$ with $1 \le |E| \le \overline{m}$ via access to the independent set oracle and an $(\overline{m}, \varepsilon)$-degree oracle of G with an underlying $(\overline{m}, \varepsilon)$-degree partition $P = (L_i, H_{k,\ell} : i, k, \ell)$. It has total cost $(n/\sqrt{\overline{m}}) \cdot \mathrm{poly}(\log n, 1/\varepsilon)$ and returns a number κ_i for each $i \in [0 : s]$ satisfying*

$$|L_i| - \frac{\varepsilon^2 \overline{m}}{\alpha^i \log^2 n} \le \kappa_i \le |L_i| \tag{3.8}$$

with probability at least $1 - 1/n^3$.

Proof: Fix an $i \in [0 : s]$ and let $c_i = |L_i|/n$. We show how to compute κ_i. If

$$\frac{\varepsilon^2 \overline{m}}{\alpha^i \log^2 n} \ge n, \tag{3.9}$$

then we can set $\kappa_i = 0$ and it satisfies (3.8) trivially. So we assume below that the inequality above does not hold. To estimate c_i we draw (the equation uses the assumption that (3.9) does not hold)

$$\left\lceil \frac{n\alpha^i \log^5 n}{\varepsilon^5 \overline{m}} \right\rceil = O\left(\frac{n\alpha^i \log^5 n}{\varepsilon^5 \overline{m}} \right)$$

vertices uniformly at random from $[n]$ (with replacements). For each vertex sampled, we query the degree oracle with a cost of $\alpha^{s-i} = \Theta(\sqrt{\overline{m}}/\alpha^i)$ to tell if it belongs to L_i. The fraction of times that a vertex sampled belongs to L_i gives us an empirical estimate $\hat{c}_i$ of c_i and it follows from Chernoff bound (using $c_i n \cdot \alpha^{i-1} = |L_i| \cdot \alpha^{i-1} \le 2|E| \le 2\overline{m}$) that

$$|\hat{c}_i - c_i| \le \frac{\varepsilon^2 \overline{m}}{2\alpha^i n \log^2 n}$$

with probability at least $1 - \varepsilon/n^4$. Setting κ_i to be

$$\kappa_i = \left(\hat{c}_i - \frac{\varepsilon^2 \overline{m}}{2\alpha^i n \log^2 n} \right) n$$

would satisfy (3.8). The total cost for obtaining κ_i is $(n/\sqrt{\overline{m}}) \cdot \mathrm{poly}(\log n, 1/\varepsilon)$. The algorithm works on each i and succeeds with probability at least $1-(s+1)\varepsilon/n^4 \ge 1 - 1/n^3$ by a union bound. ■

LEMMA 3.9. (ESTIMATION OF $|H_{k,\ell}|$) *Let $\varepsilon \in (0,1)$ and $\overline{m}$ be a positive integer. There is a randomized algorithm that runs on $G = ([n], E)$ with $1 \le |E| \le \overline{m}$ via access to the independent set oracle and an $(\overline{m}, \varepsilon)$-degree oracle of G with an underlying degree partition $P = (L_i, H_{k,\ell} : i, k, \ell)$. It has total cost $(n/\sqrt{\overline{m}}) \cdot \mathrm{poly}(\log n, 1/\varepsilon)$ and returns $\gamma_{k,\ell}$ for each $k \in [s+1 : \beta]$ and $\ell \in [0 : \tau]$ satisfying*

$$\frac{|H_{k,\ell}|}{(1+\varepsilon)^4} - O\left(\frac{\varepsilon^4 \overline{m}}{\alpha^k \log^3 n} \right) \le \gamma_{k,\ell} \le |H_{k,\ell}| \tag{3.10}$$

with probability at least $1 - 1/n^3$.

We delay the proof of Lemma 3.9 to Section 3.4 but first use it to prove Lemma 3.6

Proof of Lemma 3.6 assuming Lemma 3.9: Given G and P, we let m_1, m_2 and m_3 denote

$$m_1 = \sum_{u \in L} \deg(u, L), \quad m_2 = \sum_{u \in H} \deg(u)$$
$$m_3 = \sum_{u \in H} \deg(u, L).$$

Then we have $m = |E| = (m_1 + m_2 + m_3)/2$. The algorithm `Estimate-With-Advice*` simply runs the subroutines described in Lemma 3.8 and 3.9 to obtain κ_i's and $\gamma_{k,\ell}$'s. Letting $\gamma_k = \sum_{\ell \in [0:\tau]} \gamma_{k,\ell}$, it then outputs $\hat{m} = (\hat{m}_1 + \hat{m}_2 + \hat{m}_3)/2$, where

$$\hat{m}_1 = \sum_{i \in [s]} \kappa_i \cdot \alpha^i, \hat{m}_2 = \sum_{k \in [s+1:\beta]} \gamma_k \cdot \alpha^k$$
$$\hat{m}_3 = \sum_{\substack{k \in [s+1:\beta] \\ \ell \in [0:\tau-1]}} \gamma_{k,\ell} \cdot \alpha^{k-\ell}.$$

Assuming that κ_i's satisfy (3.8) and $\gamma_{k,\ell}$'s satisfy (3.10) (which hold with probability at least $1 - 2/n^3$ by Lemma 3.8 and Lemma 3.9), we show in the rest of the proof that $\hat{m}$ satisfies (3.7). This finishes the proof of the lemma since the worst-case total cost of `Estimate-With-Advice*` can be bounded using Lemma 3.8 and Lemma 3.9.

First for m_1, we have from (3.8) and the definition of $(\overline{m}, \varepsilon)$-degree partitions that

$$\frac{m_1}{1+\varepsilon} - O\left(\frac{\varepsilon \overline{m}}{\log n} \right) \le \sum_{i \in [s]} |L_i| \cdot \alpha^i - O\left(\frac{\varepsilon^2 \overline{m}}{\alpha^i \log^2 n} \right) s\alpha^i$$
$$\le \hat{m}_1 \le \sum_{i \in [s]} |L_i| \cdot \alpha^i \le (1+\varepsilon) m_1.$$

Next, from (3.10) combined with the fact that $|H_k| = \sum_{\ell\in[0:\tau]} |H_{k,\ell}|$ and $\tau = \Theta(\log(\log n/\varepsilon)/\varepsilon) = O(\log\log n/\varepsilon^2)$ we have that

$$\frac{|H_k|}{(1+\varepsilon)^4} - O\left(\frac{\varepsilon^2\overline{m}}{\alpha^k\log^2 n}\right) \le \frac{|H_k|}{(1+\varepsilon)^4} - O\left(\tau\cdot\frac{\varepsilon^4\overline{m}}{\alpha^k\log^3 n}\right) \le \gamma_k = \sum_{\ell\in[0:\tau]}\gamma_{k,\ell} \le |H_k|.$$

As a result, we have from the definition of $(\overline{m},\varepsilon)$-degree partitions that

$$\frac{m_2}{(1+\varepsilon)^5} - O\left(\frac{\varepsilon\overline{m}}{\log n}\right) \le \sum_{k\in[s+1:\beta]}\frac{|H_k|}{(1+\varepsilon)^4}\cdot\alpha^k - O\left(\frac{\varepsilon^2\overline{m}}{\alpha^k\log^2 n}\right)(\beta-s-1)\alpha^k \le \hat{m}_2 \le \sum_{k\in[s+1:\beta]}|H_k|\cdot\alpha^k \le (1+\varepsilon)m_2.$$

Finally the following upper bound for $\hat{m}_3$ follows from (3.10):

$$\hat{m}_3 \le \sum_{\substack{k\in[s+1:\beta]\\ \ell\in[0:\tau-1]}}|H_{k,\ell}|\cdot\alpha^{k-\ell} \le (1+\varepsilon)m_3.$$

For a lower bound note that $\sum_{k\in[s+1:\beta]}\alpha^{k-1}|H_k| \le 2|E| \le 2\overline{m}$. Together with $\alpha^\tau = \Theta(\log n/\varepsilon)$ we have

$$m_3 \le \sum_{\substack{k\in[s+1:\beta]\\ \ell\in[0:\tau-1]}}|H_{k,\ell}|\cdot\alpha^{k-\ell+1} + \sum_{k\in[s+1:\beta]}|H_k|\cdot\alpha^{k-\tau+1} \le \sum_{\substack{k\in[s+1:\beta]\\ \ell\in[0:\tau-1]}}|H_{k,\ell}|\cdot\alpha^{k-\ell+1} + O\left(\frac{\varepsilon\overline{m}}{\log n}\right).$$

As a result, we have from (3.10) that

$$\hat{m}_3 \ge \sum_{\substack{k\in[s+1:\beta]\\ \ell\in[0:\tau-1]}}\frac{|H_{k,\ell}|}{(1+\varepsilon)^4}\cdot\alpha^{k-\ell} - O\left(\frac{\varepsilon\overline{m}}{\log n}\right) \ge \frac{1}{(1+\varepsilon)^5}\cdot\left(m_3 - O\left(\frac{\varepsilon\overline{m}}{\log n}\right)\right) - O\left(\frac{\varepsilon\overline{m}}{\log n}\right) \ge \frac{m_3}{(1+\varepsilon)^5} - O\left(\frac{\varepsilon\overline{m}}{\log n}\right).$$

It follows that

$$(1-5\varepsilon)m - O\left(\frac{\varepsilon\overline{m}}{\log n}\right) \le \frac{m}{(1+\varepsilon)^5} - O\left(\frac{\varepsilon\overline{m}}{\log n}\right) \le \hat{m} \le (1+\varepsilon)m.$$

This finishes the proof of the lemma. ∎

Subroutine `High-Degree-Event`$^*(k,\ell,\eta,G)$

Input: Integers $k\in[s+1:\beta]$ and $\ell\in[0:\tau]$, a parameter $\eta\in[0,1]$ satisfying (3.11), and access to both the independent set oracle IS_G and an $(\overline{m},\varepsilon)$-degree oracle D (with underlying degree partition $P=(L_i,H_{k,\ell}: i,k,\ell))$ of a graph $G=([n],E)$ with $1\le m=|E|\le\overline{m}$.
Output: Either "few" or "many."

1. Initialize a counter $c\leftarrow 0$, and repeat the following N times:
   (a) Sample an $\mathbf{S}\subseteq[n]$ where each vertex is included with probability p independently.
   (b) Sample an $\mathbf{T}\subseteq[n]$ where each vertex is included with probability q independently.
   (c) If $\mathbf{T}$ is an independent set and $\mathbf{S}\cup\mathbf{T}$ is not an independent set (via IS_G)
      i. Run `Binary-Search`$(n,G,\mathbf{S}\cup\mathbf{T},\varepsilon^7/n^2)$ to find an edge (u,v) in $\mathbf{S}\cup\mathbf{T}$.
      ii. Query $\mathsf{D}_{\text{high}}(u,k,\ell)$ and $\mathsf{D}_{\text{high}}(v,k,\ell)$.
      iii. If $u^*\in\{u,v\}$ lies in $\mathbf{S}$ and $H_{k,\ell}$, and $\{u^*\}\cup\mathbf{T}$ is not an independent set (via IS_G), let $c\leftarrow c+1$.
2. If $c\ge h$, **return** "many;" otherwise **return** "few."

Figure 3: Description of the `High-Degree-Event`* subroutine.

3.4 Proof of Lemma 3.9

In this subsection we will prove Lemma 3.9. Specifically, fixing any $k\in[s+1:\beta]$ and $\ell\in[0:\tau]$ we will design a procedure to approximate the size of $H_{k,\ell}$. Our procedure `High-Degree-Bucket`* for this purpose uses a subroutine called `High-Degree-Event`*. Its performance guarantee is proved in the following lemma:

LEMMA 3.10. *There is a randomized algorithm* `High-Degree-Event`$^*(k,\ell,\eta,G)$ *that takes the following inputs*[9]*: integers $k\in[s+1:\beta]$ and $\ell\in[0:\tau]$, a*

[9]For convenience we skip $n,\overline{m}$ and ε as inputs of `High-Degree-Event`* and `High-Degree-Bucket`*.

parameter $\eta \in [0,1]$ satisfying[10]

$$\frac{\varepsilon^4 \overline{m}}{\alpha^k n \log^3 n} \le \eta \le 1, \tag{3.11}$$

and access to the independent set oracle IS_G and an $(\overline{m}, \varepsilon)$-degree oracle D of $G = ([n], E)$ satisfying $1 \le m = |E| \le \overline{m}$. The algorithm `High-Degree-Event*` *has a total cost of $(n/\sqrt{\overline{m}}) \cdot \mathrm{poly}(\log n, 1/\varepsilon)$ and has the following performance guarantee. Let $P = (L_i, H_{k,\ell} : i, k, \ell)$ denote the degree partition of the given degree oracle D of G. Then*

1. *If $|H_{k,\ell}| \le \eta n$, then the algorithm outputs "few" with probability at least $1 - \varepsilon^4/n^4$;*
2. *If $|H_{k,\ell}| \ge \alpha^3 \eta n$, then the algorithm outputs "many" with probability at least $1 - \varepsilon^4/n^4$.*

Proof: We describe `High-Degree-Event*` in Figure 3 with the following four parameters (one can check that $p < 1$ using the condition on $\eta \in [0,1]$ in (3.11)):

(3.12)

$$N = \frac{n \log^7 n}{\epsilon^9 \sqrt{\overline{m}}},\ h = \left(1 + \frac{\varepsilon}{4}\right) \frac{\log^2 n}{\varepsilon^3},\ p = \frac{\varepsilon^5 \sqrt{\overline{m}}}{\eta n^2 \log^4 n} \text{ and } q = \frac{\varepsilon}{\alpha^{k+1} \log n}.$$

Suppose that $|H_{k,\ell}| \le \eta n$, and consider the probability that the counter is incremented at any specific iteration of `High-Degree-Event*`. Note that a necessary condition for this to happen is that there is a vertex $u^* \in H_{k,\ell}$ that is included in $\mathbf{S}$ and u^* has a neighbor in $\mathbf{T}$ so that step 1(c)iii increments the counter c. Thus we have

$$\Pr_{\mathbf{S},\mathbf{T}}\big[c \text{ is incremented}\big] \le \sum_{u \in H_{k,\ell}} \Pr_{\mathbf{S}}\big[u \in \mathbf{S}\big] \cdot \Pr_{\mathbf{T}}\big[\mathbf{T} \text{ contains a neighbor of } u\big].$$

Given that every vertex $u \in H_{k,\ell}$ has degree at most α^{k+1}. We have

$$\Pr_{\mathbf{T}}\big[\mathbf{T} \text{ contains a neighbor of } u\big] \le \alpha^{k+1} q = \frac{\varepsilon}{\log n}.$$

As a result, for the case when $|H_{k,\ell}| \le \eta n$ we have

$$\Pr_{\mathbf{S},\mathbf{T}}\big[c \text{ is incremented}\big] \le \frac{\eta n p \varepsilon}{\log n}.$$

Next we consider the case of $|H_{k,\ell}| \ge \alpha^3 \eta n$. A *sufficient* condition for the counter to increment is that there is a vertex $u^* \in H_{k,\ell}$ such that (1) $u^* \in \mathbf{S}$, (2) one of the neighbors of u^* lies in $\mathbf{T}$, (3) $(\mathbf{S} \setminus \{u^*\}) \cup \mathbf{T}$ is an independent set, and (4) `Binary-Search` does not fail. Suppose these occur for a sample of $\mathbf{S}$ and $\mathbf{T}$ in step 1(a) and 1(b). Then, $\mathbf{T} \subset (\mathbf{S} \setminus \{u^*\}) \cup \mathbf{T}$ must be an independent set by (3), and $\mathbf{S} \cup \mathbf{T}$ is not an independent set by (1) and (2). This means step 1(c) enters lines (i), (ii) and (iii). By (3) and (4), `Binary-Search`$(m, G, \mathbf{S} \cup \mathbf{T}, \varepsilon^7/n^2)$ outputs an edge (u^*, v) since all edges in $\mathbf{S} \cup \mathbf{T}$ are adjacent to u^*; hence, (ii) executes $\mathsf{D}_{\text{high}}(u^*, k, \ell)$ and notices u^* lies in $\mathbf{S}$ and $H_{k,\ell}$. Finally by (2), $\{u^*\} \cup \mathbf{T}$ is not an independent set in (iii) and the counter is incremented.

We first show that the events (1), (2), and (3) are disjoint for different vertices $u \in H_{k,\ell}$. Suppose for contradiction that $u_1, u_2 \in H_{k,\ell}$ satisfy events (1), (2), and (3). Then, by (3), $(\mathbf{S} \setminus \{u_1\}) \cup \mathbf{T}$ and $(\mathbf{S} \setminus \{u_2\}) \cup \mathbf{T}$ are independent sets, which means that (u_1, u_2) is the only edge in $\mathbf{S} \cup \mathbf{T}$. This implies by applying (2) to u_1 that $u_2 \in \mathbf{T}$, and similarly $u_1 \in \mathbf{T}$ by applying (2) to u_2. Thus, there is an edge in $(\mathbf{S} \setminus \{u_1\}) \cup \mathbf{T}$, a contradiction. Thus, the probability for c to increment is at least (the last term accounts for `Binary-Search`)

$$\sum_{u \in H_{k,\ell}} \Pr_{\mathbf{S},\mathbf{T}}\big[u \in \mathbf{S}, \mathbf{T} \text{ contains a neighbor of } u \text{ and } (\mathbf{S} \setminus \{u\}) \cup \mathbf{T} \text{ is an IS}\big] - (\varepsilon^7/n^2).$$

Let $\mathbf{S}'$ be the set drawn by including each vertex in $[n] \setminus \{u\}$ with probability p independently, and let $\mathbf{T}'$ be the set drawn similarly from $[n] \setminus \{u\}$ using q. Then the probability in the sum above can be written as

$$p(1-q) \cdot \mathbf{Pr}_{\mathbf{S}',\mathbf{T}'}\big[\mathbf{T}' \text{ contains a neighbor of } u \text{ and } \mathbf{S}' \cup \mathbf{T}' \text{ is an IS}\big].$$

On the one hand, the probability that $\mathbf{T}'$ contains a neighbor of u is at least

$$\alpha^{k-1} q - \binom{\alpha^{k-1}}{2} q^2 \ge \left(\frac{1}{\alpha^2} - O\left(\frac{\varepsilon}{\log n}\right)\right) \cdot \frac{\varepsilon}{\log n}$$

as $\alpha^{k+1} q = \varepsilon / \log n$. On the other hand, it follows from Lemma 2.2, (3.11) and $\alpha^k = O(n)$ that

$$\mathbf{Pr}_{\mathbf{S}',\mathbf{T}'}\big[\mathbf{S}' \cup \mathbf{T}' \text{ is not an IS}\big] \le \overline{m} \cdot (p+q)^2 \le \overline{m} \cdot 2(p^2 + q^2) = O\left(\frac{\varepsilon^2}{\log^2 n}\right).$$

As a result, we have

$$\mathbf{Pr}_{\mathbf{S}',\mathbf{T}'}\big[\mathbf{T}' \text{ has a neighbor of } u \text{ and } \mathbf{S}' \cup \mathbf{T}' \text{ is an IS}\big] \ge \left(\frac{1}{\alpha^2} - O\left(\frac{\varepsilon}{\log n}\right)\right) \cdot \frac{\varepsilon}{\log n}.$$

[10] Note that the left hand side of (3.11) is smaller than 1 given that $\alpha^k > \alpha^s = \Theta(\sqrt{\overline{m}})$ and $\overline{m} \le \binom{n}{2}$.

Procedure `High-Degree-Bucket`$^*(k, \ell, G)$

Input: Integers $k \in [s+1 : \beta]$ and $\ell \in [0 : \tau]$, and access to both the independent set oracle IS_G and an $(\overline{m}, \varepsilon)$-degree oracle D (with underlying degree partition $P = (L_i, H_{k,\ell} : i, k, \ell)$) of an undirected graph $G = ([n], E)$ with $1 \leq m = |E| \leq \overline{m}$.
Output: An estimation $\gamma_{k,\ell}$ of $|H_{k,\ell}|$.

1. Let $\eta = 1$
2. While $\eta \geq \varepsilon^4 \overline{m}/(\alpha^k n \log^3 n)$, perform the following:
   (a) Run `High-Degree-Event`$^*(k, \ell, \eta, G)$
   (b) If it outputs "many," **return** ηn as $\gamma_{k,\ell}$; Otherwise, set η to be η/α.
3. **Return** 0

Figure 4: Description of the `High-Degree-Bucket`* procedure.

So for the case when $|H_{k,\ell}| \geq \alpha^3 \eta n$, we have

$$\Pr_{\mathbf{S},\mathbf{T}}\left[c \text{ is incremented}\right] \geq \alpha^3 \eta n \cdot p(1-q) \cdot \left(\frac{1}{\alpha^2} - O\left(\frac{\varepsilon}{\log n}\right)\right) \cdot \frac{\varepsilon}{\log n} - \frac{\varepsilon^7}{n^2} \geq (1+\varepsilon/2) \cdot \frac{\eta n p \varepsilon}{\log n}.$$

Plugging in the choices of p and N, we have that

$$N = \frac{\log n}{\eta n p \varepsilon} \cdot \frac{\log^2 n}{\varepsilon^4}.$$

By a Chernoff bound the counter will distinguish the two cases with probability $1 - \varepsilon^3/n^4$. ■

Using the above lemma, we can estimate the sizes of the high degree buckets.

Proof of Lemma 3.9: The algorithm simply runs `High-Degree-Bucket`$^*(k, \ell, G)$ for each $k \in [s+1 : \beta]$ and $\ell \in [0 : \tau]$ to obtain $\gamma_{k,\ell}$. Its total cost can be bounded easily given that `High-Degree-Bucket`* only invokes `High-Degree-Event`* at most $O(\log n/\varepsilon)$ many times, and both β and τ are $\widetilde{O}(\log n/\varepsilon)$.

Below we assume that every call to `High-Degree-Event`$^*(k, \ell, \eta, G)$ satisfies the two conditions in Lemma 3.10, which happens with probability at least

$$1 - \frac{\varepsilon^4}{n^4} \cdot \widetilde{O}\left(\frac{\log^3 n}{\varepsilon^3}\right) > 1 - \frac{1}{n^3}.$$

We show that every $\gamma_{k,\ell}$ satisfies (3.10) and the lemma follows.

Let $\gamma_{k,\ell}$ be the output of `High-Degree-Bucket`$^*(k, \ell, G)$. Considered two cases. First suppose line 3 is reached so $\gamma_{k,\ell} = 0$. Let $\hat{\eta}$ be the value of η in the last call to `High-Degree-Event`*. Then

$$\hat{\eta} \leq \frac{\varepsilon^4 \overline{m}}{\alpha^{k-1} n \log^3 n}$$

and because every call to `High-Degree-Event`* returns a correct answer ("few" in this case),

$$|H_{k,\ell}| \leq \alpha^3 \hat{\eta} n = O\left(\frac{\varepsilon^4 \overline{m}}{\alpha^k \log^3 n}\right)$$

so (3.10) holds trivially with $\gamma_{k,\ell} = 0$.

Next suppose that $\gamma_{k,\ell} = \hat{\eta} n$ since `High-Degree-Event`$^*(k, \ell, \hat{\eta}, G)$ outputted "many", and the previous `High-Degree-Event`$^*(k, \ell, \alpha\hat{\eta}, G)$ outputted "few." Given the assumption that both invocations return correct answers, we get that $\gamma_{k,\ell} = \hat{\eta} n \leq |H_{k,\ell}|$ and $|H_{k,\ell}| \leq \alpha^4 \hat{\eta} n = \alpha^4 \gamma_{k,\ell}$, so (3.10) follows. ■

4 Simulation of Oracles

We prove Lemma 3.7 in this section. We show how to simulate access to an $(\overline{m}, \varepsilon)$-degree oracle by giving implementations of $\texttt{Sim-D}_{\texttt{high}}$ and $\texttt{Sim-D}_{\texttt{low}}$, which assume access to an independent set oracle. To simplify the presentation, we break the simulation into two steps. In the first step, we introduce the notion of a *high-low partition* and a *high-low oracle* in Section 4.1 and show how to simulate a high-low oracle using access to an independent set oracle. In the second step, we show how to simulate an $(\overline{m}, \varepsilon)$-degree oracle with access to both an independent set oracle and a high-low oracle.

Throughout the section, let $\varepsilon \in (0, 1)$ be an accuracy parameter, $1 \leq \overline{m} \leq \binom{n}{2}$ and $G = ([n], E)$ be a graph where $1 \leq m = |E| \leq \overline{m}$. Recall $\alpha = 1 + \varepsilon$, s is set according to (3.3), $\beta = \Theta((\log n)/\varepsilon)$ is the smallest integer such that $\alpha^\beta \geq n$, and τ is the smallest integer such that $\alpha^\tau \geq \log^2 n/\varepsilon$. For convenience we will fix ε and $\overline{m}$ and skip them as inputs of algorithms presented in this section.

4.1 High-low partitions and oracles We start with the definition of high-low partitions and oracles.

DEFINITION 4.1. *An* $(\overline{m}, \varepsilon)$-high-low partition *of* $G = ([n], E)$ *is a partition* (H, L) *of* $[n]$ *such that every vertex* $u \in L$ *satisfies* $\deg(u) \leq \alpha^{s+1}$ *and every vertex* $u \in H$ *satisfies* $\deg(v) \geq \alpha^s$.

An $(\overline{m}, \varepsilon)$-high-low oracle *contains an* $(\overline{m}, \varepsilon)$*-high-low partition* (H, L) *of* G*, and can be accessed via a map*

Subroutine $\texttt{Check-High-Degree}\,(u, d, G)$

Input: A vertex $u \in [n]$, a parameter $d \geq \alpha^s$, and access to an independent set oracle IS_G of an undirected graph $G = ([n], E)$ with $1 \leq m = |E| \leq \overline{m}$.
Output: Either "low" or "high."

1. Let c be a counter, initially set to 0. Repeat $t = \text{poly}(\log n, 1/\varepsilon)$ many iterations:
   - Sample $\mathbf{T} \subseteq [n] \setminus \{u\}$ by including each vertex independently with probability $\varepsilon/(d \log n)$. Increment c if $\mathbf{T}$ is an independent set but $\mathbf{T} \cup \{u\}$ in not.
2. If $c > (1 + \varepsilon/4)t\varepsilon/\log n$, output "high;" otherwise, output "low."

Figure 5: Description of the Check-High-Degree subroutine.

$\mathsf{D}_{\mathsf{HL}} \colon [n] \to \{0, 1\}$ *such that* $\mathsf{D}_{\mathsf{HL}}(u) = 1$ *if* $u \in H$ *and* $\mathsf{D}_{\mathsf{HL}}(u) = 0$ *if* $u \in L$.

We remark (similarly to the case of $(\overline{m}, \varepsilon)$-degree partitions in Definition 3.3) that $(\overline{m}, \varepsilon)$-high-low partitions are not unique; in fact, a vertex v with $\alpha^s \leq \deg(v) \leq \alpha^{s+1}$ may belong to either H or L in an $(\overline{m}, \varepsilon)$-high-low partition (H, L). We show in the next lemma that query access to an $(\overline{m}, \varepsilon)$-high-low oracle D_{HL} can be simulated very efficiently using an independent set oracle.

Lemma 4.2. *There is a positive integer $q_{\mathsf{HL}} = q_{\mathsf{HL}}(\varepsilon, n, \overline{m})$ and a deterministic algorithm* $\texttt{High-Low}$ *with the following performance guarantee.* $\texttt{High-Low}(u, G, r)$ *takes three inputs: a vertex $u \in [n]$, access to an independent set oracle IS_G of $G = ([n], E)$ with $1 \leq m = |E| \leq \overline{m}$, and $r \in \{0, 1\}^{q_{\mathsf{HL}}}$. The algorithm makes at most* $\text{poly}(\log n, 1/\varepsilon)$ *queries to IS_G and outputs a value in $\{0, 1\}$. With probability at least $1 - 1/n^3$ over the draw of $\boldsymbol{r} \sim \{0, 1\}^{q_{\mathsf{HL}}}$, the function* $\texttt{High-Low}(\cdot, G, \boldsymbol{r}) \colon [n] \to \{0, 1\}$, *is an $(\overline{m}, \varepsilon)$-high-low oracle of G.*

Before giving the proof of Lemma 4.2, we introduce the main subroutine, Check-High-Degree, which will be used for High-Low as well as for later parts of this section.

Lemma 4.3. *There is a randomized algorithm* $\texttt{Check-High-Degree}\,(u, d, G)$ *which takes three inputs: a vertex $u \in [n]$, a parameter $d \geq \alpha^s$, and access to an independent set oracle of $G = ([n], E)$ with $1 \leq |E| \leq \overline{m}$. The algorithm makes at most* $\text{poly}(\log n, 1/\varepsilon)$ *queries and satisfies the following two properties:*

- *If* $\deg(u) \geq (1 + \varepsilon)d$*, then* $\texttt{Check-High-Degree}(u, d, G)$ *outputs "high" with probability at least $1 - \varepsilon^2/n^5$.*
- *If* $\deg(u) \leq d$*, then* $\texttt{Check-High-Degree}(u, d, G)$ *outputs "low" with probability at least $1 - \varepsilon^2/n^5$.*

Proof: Suppose first $\deg(u) \geq (1 + \varepsilon)d$. Consider the probability over the draw of $\mathbf{T} \subseteq [n] \setminus \{u\}$ that the counter c is incremented at any particular iteration. We notice that if $\mathbf{T}$ is an independent set containing a neighbor of u, the counter is incremented. Therefore,

$$\Pr_{\mathbf{T}}\big[c \text{ is incremented}\big] \geq$$
$$\Pr_{\mathbf{T}}\big[\mathbf{T} \cap \Gamma(u) \neq \emptyset\big] - \Pr_{\mathbf{T}}\big[\mathbf{T} \text{ is not an independent set}\big]$$
$$\geq 1 - \left(1 - \frac{\varepsilon}{d \log n}\right)^{(1+\varepsilon)d} - O\left(\frac{\varepsilon^2}{\log^2 n}\right)$$
$$\geq \frac{\varepsilon(1 + \varepsilon)(1 - o(\varepsilon))}{\log n} \geq \frac{\varepsilon(1 + \varepsilon/2)}{\log n}.$$

where we used Lemma 2.2 to say that $\mathbf{T}$ is very likely to be an independent set. On the other hand when $\deg(u) \leq d$, the probability that the counter is incremented is at most the probability that any neighbor of u is included in $\mathbf{T}$, so at most $\varepsilon/\log n$. By a Chernoff bound, the counter c at the end will be able to distinguish the two cases with probability at least $1 - \varepsilon^2/n^5$. ∎

We now use Lemma 4.3 to prove Lemma 4.2:
Proof of Lemma 4.2: Let $q_{\mathsf{HL}} = q_{\mathsf{HL}}(\varepsilon, n, \overline{m})$ be a large enough integer so that $r \in \{0, 1\}^{q_{\mathsf{HL}}}$ can store the randomness of calls to $\texttt{Check-High-Degree}\,(u, \alpha^s, G)$ for every $u \in [n]$. More formally, if κ is the number of random bits needed for each call to $\texttt{Check-High-Degree}\,(u, \alpha^s, G)$, then q_{HL} is set to be $n \cdot \kappa$. By a union bound, with probability at least $1 - \varepsilon^2/n^4$ over the draw of $\boldsymbol{r} \sim \{0, 1\}^{q_{\mathsf{HL}}}$, all n calls to $\texttt{Check-High-Degree}(u, \alpha^s, G)$ return a correct answer (i.e. no property in Lemma 4.3 is violated). We will refer to such a string r as a *good* string.

We now describe the implementation of $\texttt{High-Low}(u, G, r)$ and show that for every good string r, $\texttt{High-Low}(\cdot, G, r)$ implements an $(\overline{m}, \varepsilon)$-high-low oracle. When calling $\texttt{High-Low}(u, G, r)$, it just calls $\texttt{Check-High-Degree}(u, \alpha^s, G)$ with randomness taken from bits of r allocated to this call. Then $\texttt{High-Low}(u, G, r)$ outputs 1 if it outputs "high," and 0 if it outputs "low." It follows from Lemma

4.3 that High-Low(u, G, r) makes $\text{poly}(\log n, 1/\varepsilon)$ independent set queries. Moreover, when r is a good string, High-Low$(u, G, r) = 1$ implies that $\deg(u) \geq \alpha^s$; High-Low$(u, G, r) = 0$ implies that $\deg(u) \leq \alpha^{s+1}$. This finishes the proof of Lemma 4.2. ■

4.2 Implementation of a degree oracle using a high-low oracle Lemma 3.7 follows from Lemma 4.2 and the next lemma which is almost identical to Lemma 3.7, except that the algorithms now have access to both an independent set oracle and a high-low oracle.

LEMMA 4.4. *There exists a positive integer $q_* = q_*(\varepsilon, n, \overline{m})$ and two* deterministic *algorithms* $\texttt{Sim-D}^*_{\texttt{low}}$ *and* $\texttt{Sim-D}^*_{\texttt{high}}$, *where* $\texttt{Sim-D}^*_{\texttt{low}}(u, i, G, r)$ *takes as input a vertex $u \in [n]$, an index $i \in [0 : s]$, access to both an independent set oracle IS_G and an $(\overline{m}, \varepsilon)$-high-low oracle $\mathsf{D_{HL}}$ of an undirected graph $G = ([n], E)$ with $1 \leq m = |E| \leq \overline{m}$, and an $r \in \{0,1\}^{q_*}$;* $\texttt{Sim-D}^*_{\texttt{high}}(u, k, \ell, G, r)$ *takes the same inputs but has the index i replaced by indices $k \in [s+1 : \beta]$ and $\ell \in [0 : \tau]$. Both algorithms output a value in $\{0,1\}$ and together have the following performance guarantee:*

1. $\texttt{Sim-D}^*_{\texttt{low}}(u, i, G, r)$ *makes* $\alpha^{s-i} \cdot \text{poly}(\log n, 1/\varepsilon)$ *queries and* $\texttt{Sim-D}^*_{\texttt{high}}(u, k, \ell, G, r)$ *makes* $\text{poly}(\log n, 1/\varepsilon)$ *queries to the two oracles* IS_G *and* $\mathsf{D_{HL}}$.
2. *With probability at least $1 - 1/n^3$ over* $\mathbf{r} \sim \{0,1\}^{q_*}$, $\texttt{Sim-D}^*_{\texttt{low}}(u, i, G, \mathbf{r})$ *viewed as a map from $[n] \times [0 : s] \to \{0,1\}$ and* $\texttt{Sim-D}^*_{\texttt{high}}(u, k, \ell, G, \mathbf{r})$ *viewed as a map from $[n] \times [s+1 : \beta] \times [0 : \tau] \to \{0,1\}$ form an $(\overline{m}, \varepsilon)$-degree oracle of G.*

To prove Lemma 4.4, we need two procedures with properties summarized in the following two lemmas. We delay their proofs but first use them to prove Lemma 4.4.

LEMMA 4.5. *There is a randomized algorithm* Check-H-L-Degree(u, k, ℓ, G) *which takes as input a vertex $u \in [n]$, two integers $k \in [s+1 : \beta]$ and $\ell \in [\tau]$, and access to both an independent set oracle and an $(\overline{m}, \varepsilon)$-high-low oracle $\mathsf{D_{HL}}$ with $(\overline{m}, \varepsilon)$-high-low partition (H, L) of $G = ([n], E)$ with $1 \leq |E| \leq \overline{m}$. The algorithm makes $\text{poly}(\log n, 1/\varepsilon)$ queries and has the following properties when $\alpha^{k-1} \leq \deg(u) \leq \alpha^{k+1}$:*

- *If $\deg(u, L) \leq \alpha^{k-\ell}$,* Check-H-L-Degree$(u, k, \ell, G)$ *outputs "low" with probability at least $1 - \varepsilon^2/n^5$.*
- *If $\deg(u, L) \geq \alpha^{k-\ell+1}$,* Check-H-L-Degree$(u, k, \ell, G)$ *outputs "high" with probability at least $1 - \varepsilon^2/n^5$.*

LEMMA 4.6. *There is a randomized algorithm* Check-Low-Degree(u, d, G) *which takes as input a vertex $u \in [n]$, a parameter $0 < d \leq \alpha^s$, and access to an independent set oracle and an $(\overline{m}, \varepsilon)$-high-low oracle of a graph $G = ([n], E)$ with $1 \leq |E| \leq \overline{m}$. The algorithm makes $(\alpha^s/d) \cdot \text{poly}(\log n, 1/\varepsilon)$ queries to the two oracles and satisfies the following two properties:*

- *If $\deg(u, L) \geq (1 + \varepsilon)d$, then* Check-Low-Degree$(u, d, G)$ *outputs "high" with probability at least $1 - \varepsilon^2/n^5$.*
- *If $\deg(u, L) \leq d$, then* Check-Low-Degree(u, d, G) *outputs "low" with probability at least $1 - \varepsilon^2/n^5$.*

Proof of Lemma 4.4 Assuming Lemma 4.5 and 4.6: Similar to the proof of Lemma 4.2, we let q_* be a large enough integer so that a string $r \in \{0,1\}^{q_*}$ can store randomness needed by calls to

1. Check-Low-Degree(u, α^{i-1}, G) for all $u \in [n]$ and $i \in [0 : s]$;
2. Check-High-Degree(u, α^k, G) for all $u \in [n]$ and $k \in [s+1 : \beta]$; and
3. Check-H-L-Degree$(u, k, \ell, G)$for all $u \in [n]$, $k \in [s+1 : \beta]$ and $\ell \in [\tau]$.

Then it follows from Lemma 4.3, 4.5 and 4.6 and a union bound that, when $\mathbf{r} \sim \{0,1\}^{q_*}$, all these calls return a correct answer (in the sense that no property as stated in Lemma 4.3, 4.5 and 4.6 is violated) with probability $1 - 1/n^3$. We will refer to such an $r \in \{0,1\}^{q_*}$ as a *good* string, and will show that given correct outputs to all calls listed above, $\texttt{Sim-D}^*_{\texttt{low}}$ and $\texttt{Sim-D}^*_{\texttt{high}}$ can implement an $(\overline{m}, \varepsilon)$-degree oracle for G. For the remainder of the proof, we consider any fixed good string $r \in \{0,1\}^{q_*}$.

Before describing the implementation details of $\texttt{Sim-D}^*_{\texttt{low}}$ and $\texttt{Sim-D}^*_{\texttt{high}}$, it is helpful to discuss results of running all these algorithms (1), (2) and (3) on a vertex u when r is good. We first consider a vertex u with $\mathsf{D_{HL}}(u) = 0$ and thus, $u \in L$ and we have $\deg(u) \leq \alpha^{s+1}$. In this case we consider the results of running Check-Low-Degree(u, α^{i-1}, G) for each $i \in [0 : s]$, and write $a_{i-1} \in \{\text{“low”}, \text{“high”}\}$ to denote the result; we set a_s = "low" by default. Then there are two cases. If $\deg(u, L) = 0$, then all a_i = "low"; if $1 \leq \deg(u, L) \leq \alpha^{s+1}$, we have a_{-1} = "high" and by Lemma 4.6, as well as the fact r is good, there is a unique $i \in [0 : s]$ such that a_{i-1} = "high" and a_i = "low", where i satisfies $\alpha^{i-1} < \deg(u, L) < \alpha^{i+1}$ (which intuitively means that we can place u in L_i).[11]

[11]More detailed, we note that a_{-1} = "high" and a_s = "low",

Next consider a vertex u with $\mathsf{D}_{\mathsf{HL}}(u) = 1$ and thus, $u \in H$ and $\deg(u) \geq \alpha^s$. We first consider $\texttt{Check-High-Degree}(u, \alpha^k, G)$ for each $k \in [s+1 : \beta-1]$ and use $b_k \in \{$"low", "high" $\}$ to denote the result; we also set $b_\beta =$ "low" and $b_s =$ "high" by default. By Lemma 4.3, as well as the fact r is good, there is a unique $k \in [s+1 : \beta]$ such that $b_{k-1} =$ "high" and $b_k =$ "low", which implies that $\alpha^{k-1} < \deg(u) < \alpha^{k+1}$ (so we can place u in H_k). Next for this particular k, we consider $\texttt{Check-H-L-Degree}(u, k, \ell, G)$ for each $\ell \in [\tau]$ and use c_ℓ to denote the result; we also set $c_0 =$ "low" by default. If all c_ℓ's are "low," then $\deg(u, L) < \alpha^{k-\tau+1}$ (which we can place in $H_{k,\tau}$). Otherwise there exists a unique $\ell \in [0 : \tau - 1]$ such that $c_\ell =$ "low" and $c_{\ell+1} =$ "high." In this case we have $\alpha^{k-\ell-1} < \deg(u, L) < \alpha^{k-\ell+1}$ (which we can place in $H_{k,\ell}$).

We now describe the implementation of $\texttt{Sim-D}^*_{\texttt{low}}$ and $\texttt{Sim-D}^*_{\texttt{high}}$ and show that for every good string r, they together become an $(\overline{m}, \varepsilon)$-degree oracle of the graph:

1. For $\texttt{Sim-D}^*_{\texttt{low}}(u, i, G, r)$, where $i \in [0 : s]$, we first check $\mathsf{D}_{\mathsf{HL}}(u)$ and return 0 if $\mathsf{D}_{\mathsf{HL}}(u) = 1$ (meaning that $u \in H$). There are two special cases: $i = 0$ and $i = 1$. If $i = 0$, we just run $a_{-1} = \texttt{Check-Low-Degree}(u, \alpha^{-1}, G)$ and if $a_{-1} =$ "low" return 1, and return 0 otherwise. If $i = 1$, run $a_{-1} = \texttt{Check-Low-Degree}(u, \alpha^{-1}, G)$ and $a_1 = \texttt{Check-Low-Degree}(u, \alpha, G)$ and if $a_{-1} =$ "high" and $a_1 =$ "low" return 1, and return 0 otherwise. For general $i \geq 2$, we run $a_i = \texttt{Check-Low-Degree}(u, \alpha^i, G)$ and $a_{i-1} = \texttt{Check-Low-Degree}(u, \alpha^{i-1}, G)$ but set a_i to be "low" by default if $i = s$. If $a_{i-1} =$ "high" and $a_i =$ "low", return 1; otherwise, return 0.

2. For $\texttt{Sim-D}^*_{\texttt{high}}(u, k, \ell, G, r)$, where $k \in [s+1 : \beta]$ and $\ell \in [0 : \tau]$, we first check $\mathsf{D}_{\mathsf{HL}}(u)$ and return 0 if $\mathsf{D}_{\mathsf{HL}}(u) = 0$ (meaning that $u \in L$). Next run $b_k = \texttt{Check-High-Degree}(u, \alpha^k, G)$, and $b_{k-1} = \texttt{Check-High-Degree}(u, \alpha^{k-1}, G)$ but set $b_{k-1} =$ "high" if $k = s+1$ by default and set $b_k =$ "low" if $k = \beta$ by default. If $b_{k-1} =$ "high" and $b_k =$ "low", we continue; otherwise we return 0 (meaning that u does not even belong to H_k). Finally we run $c_\ell = \texttt{Check-H-L-Degree}(u, k, \ell, G)$ and $c_{\ell+1} = \texttt{Check-H-L-Degree}(u, k, \ell+1, G)$ but set $c_\ell =$ "low" by default if $\ell = 0$. If $\ell = \tau$ and $c_\tau =$ "low", return 1; return 0 otherwise. If $\ell < \tau$, return 1 if $c_\ell =$ "low" and $c_{\ell+1} =$ "high", and return 0 otherwise.

Given results of these calls analyzed above, it can be verified that $\texttt{Sim-D}^*_{\texttt{low}}$ and $\texttt{Sim-D}^*_{\texttt{high}}$ together implement an $(\overline{m}, \varepsilon)$-degree oracle when r is a good string. This finishes the proof. ■

We now provide a proof of Lemma 3.7 by using Lemma 4.4 and Lemma 4.2.

Proof of Lemma 3.7: Let $q_{\mathsf{HL}} = q_{\mathsf{HL}}(\varepsilon, n, \overline{m})$ be the integer obtained from Lemma 4.2, and $q_* = q_*(\varepsilon, n, \overline{m})$ be the integer obtained from Lemma 4.4. We let $q = q_{\mathsf{HL}} + q_*$, and we consider a string $\boldsymbol{r} \sim \{0,1\}^q$ defined as the concatenation of $\boldsymbol{r}_1 \sim \{0,1\}^{q_{\mathsf{HL}}}$ and $\boldsymbol{r}_2 \sim \{0,1\}^{q_*}$.

If the function $\texttt{High-Low}(\cdot, G, r_1)\colon [n] \to \{0,1\}$ is an $(\overline{m}, \varepsilon)$-high-low oracle of G, we say that r_1 is a *good* string, and note that by Lemma 4.2 $\boldsymbol{r}_1 \sim \{0,1\}^{q_{\mathsf{HL}}}$ is a good string with probability at least $1 - 1/n^3$. Furthermore, for any fixed r_1 which is good, we let $r_2 \in \{0,1\}^*$ be a *good* string if the functions $\texttt{Sim-D}^*_{\texttt{low}}(\cdot, \cdot, G, r_2)\colon [n] \times [0 : s] \to \{0,1\}$ and $\texttt{Sim-D}^*_{\texttt{high}}(\cdot, \cdot, \cdot, G, r_2)\colon [n] \times [s+1 : \beta] \times [0 : \tau] \to \{0,1\}$, when run with access to the independent set oracle IS_G of G and the $(\overline{m}, \varepsilon)$-high-low oracle given by $\texttt{High-Low}(\cdot, G, r_1)$, form an $(\overline{m}, \varepsilon)$-degree oracle of G. Similarly, by Lemma 4.4, we have that $\boldsymbol{r}_2 \sim \{0,1\}^{q_*}$ is a good string with probability at least $1 - 1/n^3$.

As a result, for $\boldsymbol{r}_1$ which is good, and $\boldsymbol{r}_2$ is good (with respect to $\boldsymbol{r}_1$), which occurs with probability $1 - 2/n^3$, the functions $\texttt{Sim-D}_{\texttt{low}}(\cdot, \cdot, G, \boldsymbol{r})\colon [n] \times [0 : s] \to \{0,1\}$ and $\texttt{Sim-D}_{\texttt{high}}(\cdot, \cdot, \cdot, G, \boldsymbol{r})\colon [n] \times [s+1 : \beta] \times [0 : \tau] \to \{0,1\}$ are implemented by calling the functions $\texttt{Sim-D}^*_{\texttt{low}}$ and $\texttt{Sim-D}^*_{\texttt{high}}$.We note that these functions form an $(\overline{m}, \varepsilon)$-degree oracle of G which makes queries only to the independent set oracle IS_G of G.

Lastly, the upper bound on the query complexities to IS_G of $\texttt{Sim-D}_{\texttt{low}}$ and $\texttt{Sim-D}_{\texttt{high}}$ follows from the upper bounds on the query complexities of $\texttt{Sim-D}^*_{\texttt{low}}$ and $\texttt{Sim-D}^*_{\texttt{high}}$ to IS_G and $\texttt{High-Low}$, as well as the fact that $\texttt{High-Low}$ makes at most $\text{poly}(\log n, 1/\varepsilon)$ queries to IS_G. ■

4.3 Proof of Lemma 4.5: We describe $\texttt{Check-H-L-Degree}$ in Figure 6. The procedure shares resemblance with $\texttt{Check-High-Degree}$ and the main difference is that every time a set $\mathbf{T}$ is found such that $\mathbf{T}$ is an independent set but $\mathbf{T} \cup \{u\}$ is not, we continue to find an edge $(u, v) \in E$ and then use the high-low oracle to certify that $v \in L$. Note that we do not need to run the randomized binary search in order to find an edge $(u, v) \in E$. Given that $\mathbf{T}$ is an independent set but $\mathbf{T} \cup \{u\}$ is not, one can deterministically split $\mathbf{T}$ into two parts, query the two parts together with u separately, and continue with one

so that some index $i \in [s]$ satisfies $a_{i-1} =$ "high" and $a_i =$ "low". In order to see this index is unique, note that, if for $i' \neq i$, $a_{i'-1} =$ "high" and $a_{i'} =$ "low", then either $i'-1 > i$, or $i' < i-1$, and $\alpha^{i'-1} < \deg(u, L) < \alpha^{i'+1}$; however, this contradicts the fact $\alpha^{i-1} < \deg(u, L) < \alpha^{i+1}$.

Subroutine `Check-H-L-Degree` (u,k,ℓ,G)

Input: A vertex $u\in[n]$ satisfying $\alpha^{k-1}\le\deg(u)\le\alpha^{k+1}$, integers $k\in[s+1:\beta]$ and $\ell\in[0:\tau]$, and access to an independent set oracle and an $(\overline{m},\varepsilon)$-high-low oracle D_{HL} of $G=([n],E)$ with $1\le|E|\le\overline{m}$.
Output: Either "low" or "high."

1. Let c be a counter, initially set to 0. Repeat for $t=\text{poly}(\log n,1/\varepsilon)$ iterations:
   - Sample $\mathbf{T}\subseteq[n]\setminus\{u\}$ by including each element independently with probability $\varepsilon/(\alpha^k\log n)$. If $\mathbf{T}$ is an independent set but $\mathbf{T}\cup\{u\}$ in not (obtained by querying IS_G), run a deterministic binary search to find an edge $(u,v)\in E$.
   - Query $\mathsf{D}_{\mathsf{HL}}(v)$, and increment c if it outputs 0.
2. If $c>(1+\varepsilon/4)\varepsilon t/(\alpha^\ell\log n)$, output "high;" otherwise, output "low."

Figure 6: Description of the `Check-H-L-Degree` subroutine.

that is not independent.

Now we start to prove Lemma 4.5. Consider first the case of $\deg(u,L)\le\alpha^{k-\ell}$. We note that in any iteration of line 1, the probability c is incremented is at most the probability that a neighbor $v\in\Gamma(u,L)$ is included, and this occurs with probability at most $\frac{\varepsilon}{\alpha^k\log n}\cdot\alpha^{k-\ell}\le\frac{\varepsilon}{\alpha^\ell\log n}$.

Suppose, on the other hand, that $\deg(u,L)\ge\alpha^{k-\ell+1}$. A sufficient condition for the counter c to be incremented is (1) $\mathbf{T}$ is an independent set, (2) $\mathbf{T}$ contains a unique neighbor $v\in\Gamma(u,L)$, and (3) $\mathbf{T}$ avoids all vertices in $\Gamma(u,H)$. Representing $\mathbf{T}=\mathbf{T}_1\cup\mathbf{T}_2\cup\mathbf{T}_3$ where $\mathbf{T}_1\subseteq\Gamma(u,L)$, $\mathbf{T}_2\subseteq\Gamma(u,H)$, and $\mathbf{T}_3\subseteq[n]\setminus\Gamma(u)$, we have:

$$\mathop{\mathbf{Pr}}_{\mathbf{T}}[c\text{ is incremented}]\ge$$

(4.13)
$$\ge\mathop{\mathbf{Pr}}_{\mathbf{T}_2}[\mathbf{T}_2=\emptyset]\cdot\left(\sum_{v\in\Gamma(u,L)}\mathop{\mathbf{Pr}}_{\mathbf{T}_1,\mathbf{T}_3}\left[\begin{array}{c}\mathbf{T}_1=\{v\}\wedge\\ \mathbf{T}_1\cup\mathbf{T}_3\text{ is an IS}\end{array}\right]\right),$$

(4.14)
$$\mathop{\mathbf{Pr}}_{\mathbf{T}_2}[\mathbf{T}_2=\emptyset]\ge\left(1-\frac{\varepsilon}{\alpha^k\log n}\right)^{\alpha^{k+1}}\ge1-o(\varepsilon).$$

We note that since $\deg(u)\le\alpha^{k+1}$, for any $v\in\Gamma(u,L)$,

$$\begin{aligned}\mathop{\mathbf{Pr}}_{\mathbf{T}_1}[\mathbf{T}_1=\{v\}]&\ge\frac{\varepsilon}{\alpha^k\log n}\left(1-\frac{\varepsilon}{\alpha^k\log n}\right)^{\alpha^{k+1}-1}\\ &\ge\frac{\varepsilon(1-o(\varepsilon))}{\alpha^k\log n}.\end{aligned}\tag{4.15}$$

Finally, conditioning on $\mathbf{T}_1=\{v\}$, $\mathbf{T}_1\cup\mathbf{T}_3$ is an independent set if and only if $\mathbf{T}_3\cap\Gamma(v)=\emptyset$ and $\mathbf{T}_3$ (which is sampled from $[n]\setminus\Gamma(u)$ and avoids $\Gamma(v)$) is an independent set. Since $v\in L$, the probability of $\mathbf{T}_3\cap\Gamma(v)=\emptyset$ is at least $(1-\varepsilon/(\alpha^k\log n))^{\alpha^{s+1}}\ge1-o(\varepsilon)$. As a result, viewing $\mathbf{T}_3=\mathbf{T}_3^{(0)}\cup\mathbf{T}_3^{(1)}$ where $\mathbf{T}_3^{(0)}\subset\Gamma(v)\setminus\Gamma(u)$ and $\mathbf{T}_3^{(1)}\subset[n]\setminus(\Gamma(u)\cup\Gamma(v)$, we have that for any fixed $v\in\Gamma(u,L)$,

(4.16)
$$\begin{aligned}&\mathop{\mathbf{Pr}}_{\mathbf{T}_1,\mathbf{T}_3}[\{v\}\cup\mathbf{T}_3\text{ is an independent set}]\\ &\ge(1-o(\varepsilon))\mathop{\mathbf{Pr}}_{\mathbf{T}_3^{(1)}}\left[\mathbf{T}_3^{(1)}\text{ is an independent set}\right]\ge1-o(\varepsilon),\end{aligned}$$

where we used Lemma 2.2 to say $\mathbf{T}_3^{(1)}$ is an independent set with probability at least $1-o(\varepsilon)$. Plugging (4.14), (4.15) and (4.16) back into (4.13), and recalling that $|\Gamma(u,L)|=\deg(u,L)\ge\alpha^{k-l+1}$, the probability the counter c is incremented is at least

$$(1-o(\varepsilon))\cdot\alpha^{k-\ell+1}\cdot\frac{\varepsilon(1-o(\varepsilon))}{\alpha^k\log n}\cdot(1-o(\varepsilon))\ge\frac{\varepsilon(1+\varepsilon/2)}{\alpha^\ell\log n}.$$

Given that $\alpha^\ell\le\alpha^\tau=O(\log^2 n/\varepsilon)$, it follows from a Chernoff bound that $\text{poly}(\log n,1/\varepsilon)$ iterations are enough for the counter to distinguish these two cases with probability at least $1-\varepsilon^2/n^5$.

4.4 Proof of Lemma 4.6 We present the algorithm in Figure 7. The proof follows a similar path as that of Lemma 4.5 with a few parameters set differently.

Suppose $\deg(u,L)\le d$, the probability that the counter is incremented is at most the probability that a neighbor $v\in\Gamma(u,L)$ is included in $\mathbf{T}$, which occurs with probability at most $\varepsilon d/(\alpha^s\log n)$. Suppose $\deg(u,L)\ge(1+\varepsilon)d$, and consider the probability, over the draw of $\mathbf{T}\subseteq[n]\setminus\{u\}$ that the counter c is incremented at any particular round. Similarly to the proof of Lemma 4.5, we note that a sufficient condition for this to occur is when (1) $\mathbf{T}$ is an independent set, (2) $\mathbf{T}$ contains a unique neighbor $v\in\Gamma(u,L)$, and (3) $\mathbf{T}$ avoids all vertices in $\Gamma(u,H)$. Viewing $\mathbf{T}=\mathbf{T}_1\cup\mathbf{T}_2\cup\mathbf{T}_3$ where $\mathbf{T}_1\subseteq\Gamma(u,L)$, $\mathbf{T}_2\subseteq\Gamma(u,H)$, and $\mathbf{T}_3\subseteq[n]\setminus\Gamma(u)$, Subroutine `Check-Low-Degree` (u, d, G)

Input: A vertex $u \in L$, a parameter $0 \le d \le \alpha^{s+1}$, and query access to independent set oracle IS_G and an $(\overline{m}, \varepsilon)$-degree oracle of a graph $G = ([n], E)$ with $1 \le |E| \le \overline{m}$.
Output: Either "low" or "high."

1. Let $c = 0$ be a counter. Repeat for $t = (\alpha^s/d) \cdot \text{poly}(\log n, 1/\varepsilon)$ many iterations:
   - Sample $\mathbf{T} \subseteq [n] \setminus \{u\}$ by including each vertex independently with probability $\varepsilon/(\alpha^s \log n)$. If $\mathbf{T}$ is an independent set but $\mathbf{T} \cup \{u\}$ is not (obtained by querying IS_G), run a deterministic binary search to find an edge $(u, v) \in E$ with $v \in \mathbf{T}$.
   - Query $\mathsf{D}_{\mathsf{HL}}(v)$, and increment the counter if it outputs 0, i.e., $v \in L$.
2. If $c > (1 + \varepsilon/4)\varepsilon dt/(\alpha^s \log n)$, output "high;" otherwise, output "low."

Figure 7: Description of the `Check-Low-Degree` subroutine.

we have:

(4.17)

$$\Pr_{\mathbf{T}}[c \text{ is incremented}] \ge$$
$$\ge \Pr_{\mathbf{T}_2}[\mathbf{T}_2 = \emptyset] \left(\sum_{v \in \Gamma(u,L)} \Pr_{\mathbf{T}_1, \mathbf{T}_3} \left[\begin{array}{c} \mathbf{T}_1 = \{v\} \wedge \\ \mathbf{T}_1 \cup \mathbf{T}_3 \text{ is an IS} \end{array} \right] \right),$$

(4.18)

$$\Pr_{\mathbf{T}_2}[\mathbf{T}_2 = \emptyset] \ge \left(1 - \frac{\varepsilon}{\alpha^s \log n}\right)^{\alpha^{s+1}} \ge 1 - o(\varepsilon).$$

Next for each $v \in \Gamma(u, L)$, we have

$$\Pr_{\mathbf{T}_1}[\mathbf{T}_1 = \{v\}] \ge \frac{\varepsilon}{\alpha^s \log n} \left(1 - \frac{\varepsilon}{\alpha^s \log n}\right)^{\alpha^{s+1} - 1}$$
$$\ge \frac{\varepsilon(1 - o(\varepsilon))}{\alpha^s \log n}. \tag{4.19}$$

Conditioning on $\mathbf{T}_1 = \{v\}$, $\mathbf{T}_1 \cup \mathbf{T}_3$ is an independent set if and only if $\mathbf{T}_3 \cap \Gamma(v) = \emptyset$, and $\mathbf{T}_3$ is an independent set. Similarly to (4.18), since $v \in L$, the probability of $\mathbf{T}_3 \cap \Gamma(v) = \emptyset$ is at least $(1 - \varepsilon/(\alpha^s \log n))^{\alpha^{s+1}} \ge 1 - o(\varepsilon)$. By Lemma 2.2, $\mathbf{T}_3$ (after avoiding $\Gamma(v)$) is an independent set with probability at least $1 - o(\varepsilon)$. Therefore, we obtain that (4.17) is at least

$$\frac{\varepsilon d(1 + \varepsilon)(1 - o(\varepsilon))^3}{\alpha^s \log n} > \frac{\varepsilon d(1 + \epsilon/2)}{\alpha^s \log n}$$

from combining (4.18) and (4.19).

By a Chernoff bound, the counter will distinguish these two cases with probability at least $1 - \varepsilon^2/n^5$.

5 Lower Bound

We now turn to proving the lower bound on the query complexity of estimating the number of edges of an undirected graph G with access to the independent set oracle IS_G.

We restate the main lower bound theorem:

THEOREM 5.1. *Let n and m be two positive integers with $m \le \binom{n}{2}$. Any randomized algorithm with access to the independent set oracle IS_G of an unknown $G = ([n], E)$ must make $\min(\sqrt{m}, n/\sqrt{m}) \cdot (\text{poly} \log n)^{-1}$ many queries in order to distinguish whether $|E| \le m/2$ or $|E| \ge m$ with probability at least $2/3$.*

We first establish Theorem 5.1 for the case when $m \ge n$; the case when $m < n$ follows later with a simple reduction to the case when $m \ge n$. Now let m be an integer with

$$n \le m \le \frac{n^2}{\log^6 n}. \tag{5.20}$$

Note that we further assumed that $m \le n^2/\log^6 n$. When $m \ge n^2/\log^6 n$, the lower bound we aim for becomes $\widetilde{\Omega}(\log^3 n)$ which holds trivially since (1) $\widetilde{\Omega}$ hides a factor of polylog(n) and (2) solving the problem requires at least one query to IS_G given that $m \le \binom{n}{2}$.

Assuming that m satisfies (5.20), the proof proceeds by Yao's principle. In Section 5.1 we present two distributions $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$ over undirected graphs with vertex set $[n]$ such that $\mathbf{G} \sim \mathcal{D}_{\text{yes}}$ has fewer than $m/2$ edges with probability at least $1 - o(1)$ and $\mathbf{G} \sim \mathcal{D}_{\text{no}}$ has more than m edges with probability at least $1 - o(1)$. Next, we prove in Section 5.2 that every deterministic algorithm that distinguishes $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$ must make $\widetilde{\Omega}(n/\sqrt{m})$ independent set queries. This finishes the proof of Theorem 5.1 when $m \ge n$. We work on the case when $m < n$ via a reduction in Section 5.3.

5.1 Distributions Let $d \stackrel{\text{def}}{=} m/n$ (which is not necessarily an integer). Given that m satisfies (5.20), we have that

$$1 \le d \le \frac{n}{\log^6 n}. \tag{5.21}$$

Let q be the following positive integer: $q = \left\lceil \sqrt{\frac{n}{d}} \cdot \frac{1}{\log^3 n} \right\rceil = \Theta\left(\sqrt{\frac{n}{d}} \cdot \frac{1}{\log^3 n}\right)$. We consider the following two distributions supported on graphs with vertex set $[n]$:

- $\mathcal{D}_{\text{no}}$: A graph $\mathbf{G} \sim \mathcal{D}_{\text{no}}$ is sampled by first letting $\mathbf{A} \subseteq [n]$ be a uniformly random subset of $[n]$, and $\overline{\mathbf{A}} = [n] \setminus \mathbf{A}$. Furthermore, we sample $\mathbf{B} \subseteq \mathbf{A}$ by including each element of $\mathbf{A}$ in $\mathbf{B}$ independently with probability $d \log n/n$ (note that this is smaller than 1 by (5.21)). For each $i \in \mathbf{A} \setminus \mathbf{B}$ and $j \in \overline{\mathbf{A}}$, we include the edge (i, j) in $\mathbf{G}$ independently with probability d/n. Finally, we add the edge (i, j) to $\mathbf{G}$ for every $i \in \mathbf{B}$ and $j \in \overline{\mathbf{A}}$.

- $\mathcal{D}_{\text{yes}}$: A graph $\mathbf{G} \sim \mathcal{D}_{\text{yes}}$ is sampled by first letting $\mathbf{A} \subseteq [n]$ be a uniformly random subset of $[n]$, and $\overline{\mathbf{A}} = [n] \setminus \mathbf{A}$ as above. We set $\mathbf{B} = \emptyset$ by default in $\mathcal{D}_{\text{yes}}$.[12] For each $i \in \mathbf{A} \setminus \mathbf{B} = \mathbf{A}$ and each $j \in \overline{\mathbf{A}}$, we include the edge (i, j) in $\mathbf{G}$ independently with probability d/n.

We note that with probability at least $1 - o(1)$ over the draw of $\mathbf{G} \sim \mathcal{D}_{\text{yes}}$, $\mathbf{G}$ will have no more than $m/2$ many edges. This follows from Chernoff bound and the fact that there are at most $n^2/4$ many pairs between $\mathbf{A}$ and $\overline{\mathbf{A}}$ (so the expected number of edges is no more than $(n^2/4) \cdot (d/n) = m/4$). On the other hand, with probability at least $1 - o(1)$ over the draw of $\mathbf{G} \sim \mathcal{D}_{\text{no}}$, $\mathbf{G}$ will have $\Omega(dn \log n) \geq m$ edges. This is because with probability $1 - o(1)$, $|\overline{\mathbf{A}}| = \Omega(n)$ and $|\mathbf{B}| = \Omega(d \log n)$.

As a result, Theorem 5.1 (when $m \geq n$) follows from Lemma 5.1 below because any randomized algorithm that can distinguish $|E| \leq m/2$ and $|E| \geq m$ with probability $2/3$ implies a deterministic algorithm $\texttt{Alg}$ with the same complexity such that

$$\Pr_{\mathbf{G} \sim \mathcal{D}_{\text{no}}} \big[\texttt{Alg}(\mathbf{G}) \text{ outputs "no"} \big] - \Pr_{\mathbf{G} \sim \mathcal{D}_{\text{yes}}} \big[\texttt{Alg}(\mathbf{G}) \text{ outputs "no"} \big] \geq 1/3 - o(1).$$

LEMMA 5.1. *Let* $\texttt{Alg}$ *be a deterministic algorithm that makes* q *independent set queries. Then*

$$\Pr_{\mathbf{G} \sim \mathcal{D}_{no}} \big[\texttt{Alg}(\mathbf{G}) \textit{ outputs "no"} \big] - \Pr_{\mathbf{G} \sim \mathcal{D}_{yes}} \big[\texttt{Alg}(\mathbf{G}) \textit{ outputs "no"} \big] \leq o(1).$$

5.2 Augmented oracle To prove Lemma 5.1, we will work with an *augmented* (independent set) oracle. We show that any deterministic algorithm with access to the original independent set oracle can be simulated exactly using the augmented oracle with the same query complexity (Lemma 5.2). As a result lower bounds for the augmented oracle (Lemma 5.3) carry over to the independent set oracle (Lemma 5.1).

The augmented oracle is specifically designed to be queried when the input graph is drawn from either $\mathcal{D}_{\text{yes}}$ or $\mathcal{D}_{\text{no}}$. Suppose that $\mathbf{G}$ is drawn from $\mathcal{D}_{\text{yes}}$ or $\mathcal{D}_{\text{no}}$ together with the auxiliary sets $\mathbf{A}$ and $\mathbf{B}$ (see Section 5.1). A deterministic algorithm can access the augmented oracle as follows:

- At any time during its execution, the algorithm maintains a triple (K, ℓ, e) which we will refer to as its current *knowledge triple*, where $K \subseteq [n]$ is a set of vertices, $\ell \colon K \to \{\overline{a}, a, b\}$ assigns one of three labels to each vertex in K, and $e \colon K \times K \to \{0, 1\}$. We refer to vertices in K as *known* vertices. Initially, $K = \emptyset$ (and both ℓ and e are trivial) and will grow as the result of queries made by the algorithm to the augmented oracle (see the next paragraph). For each vertex $i \in K$, $\ell(i)$ indicates whether $i \in \overline{\mathbf{A}}, \mathbf{A} \setminus \mathbf{B}$ or $\mathbf{B}$: if $i \in \overline{\mathbf{A}}$, then $\ell(i) = \overline{a}$; if $i \in \mathbf{A} \setminus \mathbf{B}$, then $\ell(i) = a$; if $i \in \mathbf{B}$, then $\ell(i) = b$.[13] Moreover, for any vertices $i, j \in K$, $e(i, j)$ is the indicator of whether (i, j) lies in $\mathbf{G}$ or not.

- At the beginning of each round, based on its current knowledge triple (K, ℓ, e), the algorithm can deterministically send a query specified by a set $Q \subseteq [n] \setminus K$ to the augmented oracle. The oracle then reacts to the query as follows:
  - If $|Q| \leq t$, where t denotes the following integer parameter
  $$t \stackrel{\text{def}}{=} \left\lceil \sqrt{n/d} \cdot \log n \right\rceil = \Theta\left(\sqrt{n/d} \cdot \log n\right),$$
  the oracle sends a new knowledge triple (K, ℓ, e) to the algorithm with $K \leftarrow K \cup Q$ and with both ℓ and e updated according to $\mathbf{G}$, $\mathbf{A}$ and $\mathbf{B}$.
  - If $|Q| > t$, the oracle samples a subset $\mathbf{L} \subseteq Q$ of size t uniformly at random. If $\mathbf{L}$ is not an independent set of $\mathbf{G}$, the oracle sends a new knowledge triple (K, ℓ, e) to the algorithm with $K \leftarrow K \cup \mathbf{L}$. If $\mathbf{L}$ happens to be an independent set of $\mathbf{G}$, we say the oracle "*fails*" and it sends a new knowledge triple (K, ℓ, e) with $K \leftarrow [n]$ (i.e., in this case the oracle simply gives up and sends the whole graph to the algorithm).

[12] We introduce $\mathbf{B}$ in $\mathcal{D}_{\text{yes}}$ only for the purpose of analysis later.

[13] Recall that when $\mathbf{G} \sim \mathcal{D}_{\text{yes}}$, we set $\mathbf{B} = \emptyset$ by default. As a result, $\ell(i) = b$ can never happen when $\mathbf{G} \sim \mathcal{D}_{\text{yes}}$.

Note that even when the algorithm is deterministic, the augmented oracle is randomized due to $\mathbf{L}$.

We show that any algorithm with access to the original independent set oracle can be simulated using the augmented oracle with the same query complexity.

LEMMA 5.2. *Let* `Alg` *be a deterministic algorithm with access to the independent set oracle. Then there is a deterministic algorithm* `Alg`* *with access to the augmented oracle (running over* $\mathbf{G}$ *drawn from either* $\mathcal{D}_{yes}$ *or* $\mathcal{D}_{no}$ *only) such that* `Alg`* *has the same query complexity as* `Alg`*,*

$$\Pr_{\mathbf{G}\sim\mathcal{D}_{no}}\big[\texttt{Alg}(\mathbf{G}) \text{ outputs "no"}\big] = \Pr_{\mathbf{G}\sim\mathcal{D}_{no}}\big[\texttt{Alg}^*(\mathbf{G}) \text{ outputs "no"}\big],$$

and the same equation holds for $\mathcal{D}_{yes}$[14].

Given Lemma 5.2, Lemma 5.1 follows directly from the following lemma:

LEMMA 5.3. *Let* `Alg`* *be any deterministic algorithm that makes* q *queries to the augmented oracle (over graphs* $\mathbf{G}$ *drawn from either* $\mathcal{D}_{yes}$ *or* $\mathcal{D}_{no}$ *only). Then we have*

$$\Pr_{\mathbf{G}\sim\mathcal{D}_{no}}\big[\texttt{Alg}^*(\mathbf{G}) \text{ outputs "no"}\big] - \Pr_{\mathbf{G}\sim\mathcal{D}_{yes}}\big[\texttt{Alg}^*(\mathbf{G}) \text{ outputs "no"}\big] \le o(1).$$

We start with some intuition. Given access to the augmented oracle, an algorithm will aim to make the set K as large as possible in order to maximize the chance of $\mathbf{B}\cap K \neq \emptyset$ (in which case one can conclude that $\mathbf{G}\sim\mathcal{D}_{\text{no}}$). Now if the algorithm makes a query $Q\subseteq[n]\setminus K$ with $|Q|\le t=\sqrt{n/d}\cdot\log n$, the probability of $Q\cap\mathbf{B}\neq\emptyset$ is $o(1/q)$ given that $|\mathbf{B}|$ is only roughly $d\log n$ (this will be made formal in the proof of Lemma 5.5). On the other hand, if $|Q|>t$, then a vertex in $\mathbf{B}$ can be added to K because either $\mathbf{L}\cap\mathbf{B}\neq\emptyset$ (which happens with low probability by a similar analysis since $|\mathbf{L}|=t$) or the oracle "fails" (which we show that is unlikely to happen).

To proceed with the proof of Lemma 5.3 we view `Alg`* as a tree of depth q in which each internal node is labelled by a query set, each leaf is labelled either "yes" or "no," and each edge is labelled by a knowledge triple (K,ℓ,e) as the result of the previous query received from the augmented oracle. Let (u,v) be an edge with v being a child of u. The label of (u,v) is the current knowledge triple (K,ℓ,e) of the algorithm when it arrives at v and thus, the query set Q at v is a subset of $[n]\setminus K$. We will refer to the label of (u,v) as the current knowledge triple of v; for the root we have $K=\emptyset$.

We introduce the following definition of good and bad nodes, which is inspired by the intuition that an algorithm would aim for reaching a K with $K\cap\mathbf{B}\neq\emptyset$. We then state two lemmas based on this definition and use them to prove Lemma 5.3.

DEFINITION 5.4. *We say a node* v *in the tree of an algorithm* `Alg`* *is* good *if its current knowledge triple* (K,ℓ,e) *satisfies* $\ell^{-1}(b)=\emptyset$*. We say* v *is* bad *otherwise.*

We note that $\mathbf{G}\sim\mathcal{D}_{\text{yes}}$ can never reach a bad node since $\mathbf{B}=\emptyset$ in this case.

LEMMA 5.5. *We have (the probability is over* $\mathbf{G}\sim\mathcal{D}_{no}$ *and randomness of the augmented oracle)* $\Pr_{\mathbf{G}\sim\mathcal{D}_{no}}\big[\texttt{Alg}^*(\mathbf{G}) \text{ reaches a bad node}\big]=o(1)$.

LEMMA 5.6. *For every good node* v *in the tree of* `Alg`**, we have* $\Pr_{\mathbf{G}\sim\mathcal{D}_{no}}\big[\texttt{Alg}^*(\mathbf{G}) \text{ reaches } v\big] \le \Pr_{\mathbf{G}\sim\mathcal{D}_{yes}}\big[\texttt{Alg}^*(\mathbf{G}) \text{ reaches } v\big]$.

Using the above two lemmas we derive Lemma 5.3 (we refer the reader to the full version for the proofs).

5.3 Proof of Theorem 5.1 The case when $m\ge n$ follows directly from Lemma 5.3.

When $m<n$, we use the observation that every randomized algorithm with parameters n and m (i.e., determining whether an input graph $G=([n],E)$ satisfies $|E|\le m/2$ or $|E|\ge m$) implies a randomized algorithm with parameters m and m (i.e., determining whether a given graph $G'=([m],E')$ has $|E'|\le m/2$ or $|E'|\ge m$) with the same query complexity by simply embedding the input graph $G'=([m],E')$ in a graph $G=([n],E)$ using its first m vertices (and noting that the independent set oracle of G can be simulated using that of G' query by query). The latter task, by Lemma 5.3, has a lower bound of $\tilde{\Omega}(\sqrt{m})$. This finish the proof of the theorem when $m<n$.

[14] Note that in the right hand side the probability is not only over the draw of $\mathbf{G}\sim\mathcal{D}_{\text{no}}$ but also the randomness of the augmented oracle. The same comment applies to similar expressions in the rest of the section.

References

[1] M. Aliakbarpour, A. S. Biswas, T. Gouleakis, J. Peebles, R. Rubinfeld, and A. Yodpinyanee. Sublinear-time algorithms for counting star subgraphs with applications to join selectivity estimation. *arXiv preprint arXiv:1601.04233*, 2016.

[2] S. Assadi, M. Kapralov, and S. Khanna. A simple sublinear-time algorithm for counting arbitrary

subgraphs via edge sampling. In *Proceedings of the 2019 ACM Conference on Innovations in Theoretical Computer Science (ITCS '2019)*, 2019.
[3] P. Beame, S. Har-Peled, S. N. Ramamoorthy, C. Rashtchian, and M. Sinha. Edge estimation with independent set oracles. In *Proceedings of the 2018 ACM Conference on Innovations in Theoretical Computer Science (ITCS '2018)*, 2018.
[4] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on Computing*, 34(6):1370–1379, 2005.
[5] C. L. Chen and W. H. Swallow. Using group testing to estimate a proportion, and to test the binomial model. *Biometrics*, pages 1035–1046, 1990.
[6] G. Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. *ACM Transactions on Database Systems (TODS)*, 30(1):249–278, 2005.
[7] A. Czumaj, F. Ergün, L. Fortnow, A. Magen, I. Newman, R. Rubinfeld, and C. Sohler. Approximating the weight of the euclidean minimum spanning tree in sublinear time. *SIAM Journal on Computing*, 35(1):91–109, 2005.
[8] A. Czumaj and C. Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM Journal on Computing*, 39(3):904–922, 2009.
[9] R. Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943.
[10] D. Du and F. K. Hwang. *Combinatorial group testing and its applications*, volume 12. World Scientific, 2000.
[11] T. Eden, A. Levi, D. Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017.
[12] T. Eden, D. Ron, and C. Seshadhri. On approximating the number of k-cliques in sublinear time. In *Proceedings of the 50th ACM Symposium on the Theory of Computing (STOC '2018)*, pages 722–734. ACM, 2018.
[13] T. Eden and W. Rosenbaum. On sampling edges almost uniformly. In *Proceedings of the 1st Symposium on Simplicity in Algorithms, (SOSA '2018)*, 2018.
[14] U. Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing*, 35(4):964–984, 2006.
[15] O. Goldreich and D. Ron. Approximating average parameters of graphs. *Random Structures & Algorithms*, 32(4):473–493, 2008.
[16] M. Gonen, D. Ron, and Y. Shavitt. Counting stars and other small subgraphs in sublinear-time. *SIAM Journal on Computing*, 25(3):1365–1411, 2011.
[17] A. Hassidim, J. A. Kelner, H. N. Nguyen, and K. Onak. Local graph partitions for approximation and testing. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2009)*, pages 22–31. IEEE, 2009.
[18] P. Indyk, H. Q. Ngo, and A. Rudra. Efficiently decodable non-adaptive group testing. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA '2010)*, pages 1126–1142. Society for Industrial and Applied Mathematics, 2010.
[19] A. Kumar, C. Seshadhri, and A. Stolman. Finding forbidden minors in sublinear time: A n^ 1/2+ o (1)-query one-sided tester for minor closed properties on bounded degree graphs. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2018)*, pages 509–520. IEEE, 2018.
[20] A. Kumar, C. Seshadhri, and A. Stolman. Random walks and forbidden minors ii: a poly($d\varepsilon^{-1}$)-query tester for minor-closed properties of bounded degree graphs. In *Proceedings of the 51st ACM Symposium on the Theory of Computing (STOC '2019)*, 2019.
[21] A. J. Macula and L. J. Popyack. A group testing method for finding patterns in data. *Discrete applied mathematics*, 144(1-2):149–157, 2004.
[22] S. Marko and D. Ron. Approximating the distance to properties in bounded-degree and general sparse graphs. *ACM Transactions on Algorithms*, 5(2):22, 2009.
[23] H. Q. Ngo and D.-Z. Du. A survey on combinatorial group testing algorithms with applications to dna library screening. *Discrete mathematical problems with medical applications*, 55:171–182, 2000.
[24] H. N. Nguyen and K. Onak. Constant-time approximation algorithms via local improvements. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2008)*, pages 327–336. IEEE, 2008.
[25] K. Onak, D. Ron, M. Rosen, and R. Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. pages 1123–1131. Society for Industrial and Applied Mathematics, 2012.
[26] M. Parnas and D. Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007.
[27] D. Ron and G. Tsur. The power of an example: Hidden set size approximation using group queries and conditional sampling. *ACM Transactions on Computation Theory*, 8(4):15, 2016.
[28] W. H. Swallow. Group testing for estimating infection rates and probabilities of disease transmission. *Phytopathology (USA)*, 1985.
[29] Y. Yoshida, M. Yamamoto, and H. Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proceedings of the 41st ACM Symposium on the Theory of Computing (STOC '2009)*, pages 225–234. ACM, 2009.