When Local and Non-Local Meet: Quadratic Improvement for Edge Estimation with Independent Set Queries

Tomer Adar
Technion – Israel Institute of Technology, Haifa, Israel

Yahel Hotam
University of Haifa, Israel

Amit Levi
University of Haifa, Israel

Abstract

We study the problem of estimating the number of edges in an unknown graph. We consider a hybrid model in which an algorithm may issue independent set, degree, and neighbor queries. We show that this model admits strictly more efficient edge estimation than either access type alone. Specifically, we give a randomized algorithm that outputs a $(1 \pm \varepsilon)$-approximation of the number of edges using $O\left(\min\left(\sqrt{m}, \sqrt{\frac{n}{\sqrt{m}}}\right) \cdot \frac{\log n}{\varepsilon^{5/2}}\right)$ queries, and prove a nearly matching lower bound. In contrast, prior work shows that in the local query model (Goldreich and Ron, *Random Structures & Algorithms* 2008) and in the independent set query model (Beame *et al.* ITCS 2018, Chen *et al.* SODA 2020), edge estimation requires $\widetilde{\Theta}(n/\sqrt{m})$ queries in the same parameter regimes. Our results therefore yield a quadratic improvement in the hybrid model, and no asymptotically better improvement is possible.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Edge count estimation, Degree oracle, Neighbor oracle, Independent-set oracle

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2026.33

Category RANDOM

Related Version *Full Version*: https://arxiv.org/pdf/2601.21457

1 Introduction

Estimating the number of edges in an unknown graph is a central problem in sublinear algorithms. Formally, given query access to a graph $G = (V, E)$ on n vertices, the goal is to approximate $m = |E|$ within a $(1 \pm \varepsilon)$-multiplicative factor while making as few queries as possible. The problem captures the inherent tension between accuracy and information constraints and has been extensively studied (see e.g., [21, 22, 20, 19, 5, 23, 28, 8, 12])

A key theme in this line of work is that the achievable query complexity depends critically on the type of access allowed to the graph. Different query models expose fundamentally different kinds of information, leading to qualitatively different algorithmic techniques.

Until recently, most work in the sublinear algorithms community focused on *local access models*, where queries are restricted to a vertex and its neighborhood. Typical examples include degree queries, which return a vertex's degree, and neighbor queries, which return a specified neighbor. Such queries provide fine-grained, precise information about the graph.

Beame, Har-Peled, Ramamoorthy, Rashtchian, and Sinha [6] introduced a markedly different access model, the *independent-set (IS) oracle*. In this model, an algorithm may query any subset $S \subseteq V$, and the oracle responds with a single bit indicating whether S induces an edge-free subgraph. Unlike local queries, IS queries do not reveal the identity or

location of edges. Instead, they provide a coarse *non-local* information about the presence or absence of edges within a queried set. This model can be viewed as a form of combinatorial group testing, where a single query can simultaneously probe a large portion of the graph. As such, IS queries expose structural information that is fundamentally inaccessible through purely local inspection.

[6] first analyzed edge estimation in the IS model and obtained an upper bound of $\widetilde{O}(n^{2/3})$ queries. Chen, Levi, and Waingarten [15] refined this analysis and established nearly optimal bounds: $(1 \pm \varepsilon)$-approximate edge estimation can be performed using $\widetilde{O}\left(\min\left(\sqrt{m}, \frac{n}{\sqrt{m}}\right)\right)$ IS queries, and show that this dependence on n and m is optimal up to polylogarithmic factors.

Local queries and IS queries exhibit inherently different strengths. Local queries provide detailed information about individual vertices and their neighborhoods, whereas IS queries give coarse binary feedback over subsets. When considering the local model, (nearly) tight bounds of $\widetilde{\Theta}(n/\sqrt{m})$ queries for edge estimation are known [22, 21]. Therefore, there exist settings in which both oracles require the same asymptotic number of queries, but for entirely different reasons: local queries are limited by the need to sample enough edges from sparse regions, while IS queries are limited by the coarse granularity of global binary feedback. This illustrates that even when the query complexities coincide, the information revealed by each oracle is *fundamentally different*, reflecting their complementary powers.

This contrast naturally motivates the study of *hybrid* access models, in which an algorithm is allowed to use both local queries and IS queries. Such a model offers the potential to combine non-local aggregation with local precision: local queries can be used to identify informative regions of the graph or reveal degrees, whereas IS queries can efficiently summarize the presence of edges across large subsets. Understanding the power and limitations of this hybrid model is essential for clarifying how different forms of partial information interact, and for determining whether combining query types can lead to provable improvements over either model alone.

1.1 Our Results

In this work, we provide a tight characterization of this hybrid model.

▶ **Theorem 1** (Upper Bound). *Consider the query model consisting of IS, degree, and neighbor queries to an unknown graph $G = (V, E)$ with $|V| = n$ and $|E| = m$. For every $0 < \varepsilon < 1$, there exists a randomized algorithm that uses $O\left(\min\left(\sqrt{m}, \sqrt{\frac{n}{\sqrt{m}}}\right) \cdot \frac{\log n}{\varepsilon^{5/2}}\right)$ queries in expectation and outputs a $(1 \pm \varepsilon)$-approximation of m with probability at least $2/3$.*

▶ **Theorem 2** (Lower Bound). *For every $\varepsilon \in (0, 1)$, in the query model allowing independent set, degree, and neighbor queries, any randomized algorithm that estimates the number of edges m of an n-vertex graph to within a $(1 \pm \varepsilon)$ factor with success probability at least $2/3$ requires $\Omega\left(\min\left(\sqrt{m}, \sqrt{\frac{n}{\varepsilon\sqrt{m}}}\right)\right)$ queries.*

In particular, in regimes where each individual model would require $\widetilde{\Theta}(n/\sqrt{m})$ queries, the combination yields a *quadratic improvement*, whereas in other regimes it matches the best of the two models. These results demonstrate that IS and local queries are synergistic yet complementary, and that the hybrid model achieves the maximum possible improvement for the task of edge estimation.

1.2 Technical overview

1.2.1 Upper-Bound

Our algorithm is advice-based: it receives a guess $\bar{m}$ for the number of edges m, but treats this guess as untrusted. For each guess, the algorithm must either produce a $(1\pm\varepsilon)$-approximation to m, or determine that $\bar{m}$ is not within the correct scale. The goal is to obtain query complexity $\widetilde{O}\left(\min\left(\sqrt{m}, \sqrt{n/\sqrt{m}}\right)\right)$.

To achieve this goal, we interleave two exponentially progressing sequences of guesses. The upper sequence starts from n^2 and decreases geometrically, taking $\bar{m} = 2^{-\ell}n^2$ at iteration ℓ. The lower sequence increases geometrically, taking $\bar{m} = 2^{\ell/2}$. The different rates are chosen so that the cost of handling a guess from either sequence is comparable within the same iteration.

For a fixed guess $\bar{m}$, the algorithm partitions vertices according to a degree threshold k, classifying vertices as low-degree or high-degree. This induces a decomposition of the edge set into edges between two low-degree vertices (m_{LL}), edges between low- and high-degree vertices (m_{LH}), and edges between two high-degree vertices (m_{HH}). These components are handled separately, exploiting the fact that high-degree vertices are rare once k is chosen appropriately.

The main component is estimating m_{LL}. We adapt the edge-sampling approach of [6], which samples an induced subgraph by including each vertex independently with probability $1/\sqrt{\bar{m}}$. If all vertices were low-degree, then this estimator would have low variance. We use the degree oracle to identify edges incident to high-degree vertices and exclude them from counting to keep the variance low while preserving unbiasedness.

Edges between high-degree vertices have small contribution overall. For an appropriate choice of k, the number of such edges is $O(\varepsilon m)$, which can be absorbed into the allowed additive error. For low-high edges, we further refine the low-degree vertices by introducing a second threshold k', defining very-low-degree vertices, and only count the edges between them to high-degree vertices. With this refinement, for an appropriate choice of k, the number of excluded edges is $O(\varepsilon m)$, allowing us to only consider edges between very-low-degree vertices and high-degree vertices.

The remaining quantity, denoted m_{L_1H} (between very-low-degree vertices and high-degree vertices), is estimated via a weighted sampling procedure. We sample a vertex, then sample one of its neighbors, and count the edge with weight proportional to the degree of the sampled vertex whenever it is very-low-degree and its neighbor is high-degree. This yields an unbiased estimator whose variance is controlled using $O(n/(\varepsilon^3 k))$ samples.

We analyze the algorithm separately for the two guess sequences. We first consider the upper guess sequence, for which $\bar{m} \geq m$ holds throughout until a correct-scale guess is reached. In this regime, we set $k = \Theta\left(\sqrt{n\sqrt{\bar{m}}/\varepsilon}\right)$, except in the very dense case $\bar{m} = \Omega(\varepsilon n^2)$, where a separate optimization applies. With this choice, the combined cost of estimating m_{LL} and m_{L_1H} is $O\left(\frac{\log n}{\varepsilon^{5/2}} \cdot \sqrt{\frac{n}{\sqrt{\bar{m}}}}\right)$.

When $\bar{m} = \Theta(m)$, the algorithm outputs $\tilde{m} = (1 \pm O(\varepsilon))m$ with high probability. More generally, when $\bar{m}$ is larger than the true value, the output is noticeably smaller than $\bar{m}$, and in particular bounded by $m + \varepsilon\bar{m} \ll \bar{m}$ when $\bar{m} \gg m$. This gap allows the algorithm to reliably detect overly large guesses and continue to smaller ones.

For the lower guess sequence, the algorithm cannot assume that $\bar{m} \geq m$. Instead, it sets the degree threshold to $k = \min(\bar{m}, n-1)$ and attempts to verify that no vertex exceeds this degree. If this verification succeeds, then all vertices are low-degree and $m_{LL} = m$, so

estimating m_{LL} alone suffices. If the guess $\bar{m}$ is too small, then either $m_{LL} > 2\bar{m}$, or there exist $\Omega(m)$ edges adjacent to vertices of degree larger than $\bar{m}$ (in particular, there exists a vertex of degree larger than $\bar{m}$). These cases are detected by running a complexity-safe version of the m_{LL} estimator, which aborts if it exceeds the cost expected when assuming that $\bar{m} \geq m$, and by searching for a high-degree vertex. The resulting cost per lower-sequence iteration is $O\left(\frac{\log n}{\varepsilon^2}\sqrt{\bar{m}}\right)$.

Standard amplification could be applied to each iteration to ensure correctness with high probability. However, a more refined analysis shows that such amplification is unnecessary. The failure probability of an upper-sequence iteration is $O(m/\bar{m})$, while that of a lower-sequence iteration is $O(\bar{m}/m)$. Since both guess sequences are exponential, the total failure probability accumulated over all iterations preceding the one with $\bar{m} \approx m$ remains bounded by a constant. The iteration for which $\bar{m} = \Theta(m)$ therefore succeeds with constant probability.

Finally, while the above discussion yields the desired query complexity with high probability, controlling the expected complexity requires care due to the possibility that "the" correct guess fails, and then further high-sequence guesses are smaller than m and low-sequence guesses are greater than m, making the rest of the run incompatible with the correctness analysis (even though complexity analysis still holds). To address this, we execute the guesses using an iterative deepening schedule. We first consider iteration 0, then iterations $(0, 1)$, then $(0, 1, 2)$, and so on. This scheduling ensures that the tail of the running-time distribution decays sufficiently fast, so that the expected query complexity matches the high-probability bound asymptotically.

1.2.2 Lower bound

We show that any deterministic algorithm with access to degree, neighbor and independent-set oracles cannot distinguish between two distributions over graph whose degrees are separated by a $(1+\Omega(\varepsilon))$-factor, unless it makes $\Omega\left(\min\left(\sqrt{m}, \sqrt{n/\varepsilon\sqrt{m}}\right)\right)$ queries in expectation (note that the input distribution is the source of randomness). By Yao's principle [29], this lower bound also applies to probabilistic algorithms with the same expected query complexity.

Canonical graph construction

To define the canonical graph $G_{n,d,\varepsilon}$, let $d_0(n, \epsilon) \stackrel{\text{def}}{=} 1/(n^{1/3}\epsilon^{2/3})$. For the simplicity of the overview, we focus on the case where $d \geq d_0(n, \varepsilon)$, whose lower bound is $\Omega(\sqrt{n/\varepsilon\sqrt{m}})$. The input graph consists of disjoint vertex sets K, L, H, and A whose sizes are:

$$n_k = \left\lfloor \sqrt{2nd} \right\rfloor, \quad n_h = \left\lceil \varepsilon^{1/2} n^{1/4} d^{3/4} \right\rceil, n_\ell = \left\lceil \varepsilon^{1/2} n^{3/4} d^{1/4} \right\rceil, \quad n_a = n - (n_k + n_h + n_\ell).$$

For the graph edges: the set K forms a clique, the sets L and H are the sides of a biclique and A consists of isolated vertices. The K-clique contributes $\Omega(m)$ edges (but not more than m), and the L-H graph contributes $\Theta(\varepsilon m)$ edges. Therefore, the number of edges is $\Theta(m)$ and the average degree is $\Theta(d)$.

Graph distributions

In the first distribution we permute the vertices of $G_{n,d,0}$, that is, the graph only consists of a clique but not a biclique, and it has at most m edges. In the second distribution we permute the vertices of $G_{n,d,\varepsilon}$, which has at least εm additional edges.

Intuition

As long as the algorithm does not query a vertex in the biclique or find a biclique-edge, the input distributions are indistinguishable. For local queries, the probability of hitting the biclique scales with the size of L. For independent-set queries, a query must include at least one vertex from H to detect an edge in the biclique. Large IS-queries are limited by the clique size $|K|$, since including two vertices from K makes the queried set not-independent regardless of whether or not it has a biclique-edge.

A more careful argument shows that for degree and neighbor queries, the probability of sampling a biclique vertex is $O((n_\ell + n_h)/n) = O(n_\ell/n)$. Hence, $\Omega\left(\frac{n}{n_\ell}\right) = \Omega\left(\sqrt{\frac{n}{\varepsilon\sqrt{m}}}\right)$ queries are needed to hit a biclique vertex with $\Omega(1)$ probability.

For IS-queries, the clique K limits the effective query size to $O(n/n_k) = O(\sqrt{n/d})$, and the probability of hitting H using such query is $O\left(\frac{n_h}{n} \cdot \sqrt{\frac{n}{d}}\right) = O\left(\frac{n_h}{\sqrt{nd}}\right)$. This probability leads to the same lower bound as before. Overall, as $d = \Theta(m/n)$, distinguishing the two distributions with probability $\Omega(1)$ requires $\Omega\left(\sqrt{\frac{n}{\varepsilon\sqrt{m}}}\right)$ queries, which establishes the desired lower bound.

For $d < d_0(n, \epsilon)$, we observe that the definition of n_h is fixed to 1, and therefore, we can reduce n_ℓ to $\lceil \varepsilon nd \rceil$. Besides the alternative value of n_ℓ, the construction is the same. Analyzing this small-d case uses similar ideas and results in an $\Omega(\sqrt{m})$ lower bound.

1.3 Related work

A generalization of independent set queries appears in a broader line of work investigating the relationship between decision and counting problems [25, 26, 24, 16, 17, 11]. Independent set queries also have deep connections to group testing, where one tests large collections of items to detect the presence of rare events [18, 27, 14]. Independent set queries also have close connections to learning theory. In particular, a classical line of work on learning hidden graphs studies *edge-detecting queries*, which are equivalent to independent set queries [3, 4, 1].

Several works have explored combinatorial or global query primitives beyond purely local access. Beyond standard IS queries, Beame *et al.* [6] introduced *bipartite independent set (BIS) queries*, which ask whether there exists at least one edge between two given vertex subsets, and used them to obtain tight sublinear bounds for edge count estimation. Subsequently, [2] designed *non-adaptive* algorithms for edge counting and sampling in the BIS model. The BIS model was later extended to more general settings (see e.g., [9, 13, 10]).

Perhaps the work most closely related to ours is that of Beretta, Chakrabarty, and Seshadhri [7], which studies average-degree estimation under a query model combining random-edge sampling with structural queries (e.g., pair queries and full neighborhood access). They also discuss a structural oracle that, given a set $S \subseteq V$, returns the number of edges induced by S at a cost proportional to $|S|$. These structural queries provide precise local or aggregate information but follow a different cost model than IS queries. In particular, it is possible to show that IS oracle can simulate their structural oracle using (almost) linear number of IS queries.

2 Preliminaries

Given a positive integer n, we denote $[n] = \{1, ..., n\}$. For a set A, we denote the power set of A by 2^A. All graphs $G = (V, E)$ considered in this paper are undirected and simple (meaning that there are no parallel edges or loops), with $V = [n]$ as their vertex set (we

often just write $G = ([n], E)$). We also let $m = |E|$ be the number of edges in the graph and $d = 2m/n$ be the average degree in the graph. We use $\deg(v)$ to denote the degree of a vertex $v \in [n]$. We use boldface letters (such as $\mathbf{X}$) to denote random variables. We use the notation $\tilde{x} = (1 \pm \varepsilon)x$ to denote that $(1-\varepsilon)x \leq \tilde{x} \leq (1+\varepsilon)x$.

An algorithm is allowed to perform three types of queries to access the (otherwise hidden) edge set of the graph E, as defined by the following oracles:

▶ **Definition 3** (Degree oracle). *Given an undirected graph $G = ([n], E)$, its* degree oracle *is a map $\mathsf{Deg} : [n] \to [n-1] \cup \{0\}$, which for a vertex $u \in [n]$ returns the degree* $\deg(u)$.

▶ **Definition 4** (Random Neighbor oracle). *Given an undirected graph $G = ([n], E)$, its* random neighbor oracle *is a map $\mathsf{Neigh} : [n] \to [n]$, which for a vertex $u \in [n]$ returns a uniformly random neighbor of u. If u has no neighbors, it returns a special character to signify it.*

▶ **Definition 5** (Independent set oracle). *Given an undirected graph $G = ([n], E)$, its* independent set oracle *is a boolean map $\mathsf{IS} : 2^{[n]} \to \{0, 1\}$, which for a subset of vertices $U \subseteq [n]$ returns 1 if and only if U is an independent set of G (i.e., the subgraph induced on U contains no edges).*

Some of our algorithms will include the assumption that $m \geq 1$, note that we can distinguish this from the case that $m = 0$ using a single query to IS.

2.1 Classifying vertices and edges

We classify the vertices in V into three disjoint sets by their degrees, and use those sets to classify the edges in the graph:

▶ **Definition 6.** *Given a graph $G = (V, E)$, let k' and k be two degree thresholds (not necessarily integers) for which $k \geq k' \geq 0$ (where $n = |V|$). Let V_{L_1}, V_{L_2}, V_H be a partition of $[n]$ into disjoint sets such that:*

$$V_{L_1} = \{u \in V : \deg(u) \leq k'\}, V_{L_2} = \{u \in V : k' < \deg(u) \leq k\}, V_H = \{u \in V : \deg(u) > k\}.$$

Also, let $V_L = V_{L_1} \cup V_{L_2} = \{u \in V : \deg(u) \leq k\}$.

For two labels $\sigma_1, \sigma_2 \in \{L_1, L_2, L, H\}$, we use $E_{\sigma_1\sigma_2}$ to denote the set of edges between V_{σ_1} and V_{σ_2} (note that $E_{\sigma_1\sigma_2}$ and $E_{\sigma_2\sigma_1}$ are identical). Also, we let $m_{\sigma_1\sigma_2} = |E_{\sigma_1\sigma_2}|$ be the number of these edges.

At top-level, we "guess" (formally, search for) an advice $\bar{m}$, hopefully just slightly greater than the correct answer m. Based on $\bar{m}$ we define the threshold degrees k and k', and use different logic to estimate the number of edges between every pair of subsets. For some pairs the estimation is zero, since the choice of k, k' means that the number of relevant edges is no more than linear in the allowed additive error and thus are negligible.

For $\bar{m} > 0$ and $0 < \varepsilon < 1$, let $k(\bar{m}, \varepsilon) \stackrel{\text{def}}{=} \sqrt{\frac{2n\sqrt{\bar{m}}}{\varepsilon}}$ and $k' = \min(k, \bar{m}/\varepsilon k)$, and recall the sets of Definition 6. Clearly, both k and $\bar{m}/\varepsilon k$ are monotone increasing in $\bar{m}$ (when n and ε are considered fixed). Therefore, the set $V_H(\bar{m}, \varepsilon)$ and the union set $V_{L_2}(\bar{m}, \varepsilon) \cup V_H(\bar{m}, \varepsilon)$ are monotone non-increasing in $\bar{m}$. Note that when $k' = k$, we have $|V_{L_2}| = 0$ and thus $V_L = V_{L_1}$.

▶ **Lemma 7.** *Let $k(\bar{m}, \varepsilon) = \sqrt{2n\sqrt{\bar{m}}/\varepsilon}$. If $\bar{m} \geq \frac{1}{4}m$, then the number of edges between $V_{L_2} \cup V_H$ and V_H is at most $64\varepsilon\bar{m}$.*

▶ **Lemma 8.** *If $\bar{m} \geq \frac{1}{4}m$ and $k(\bar{m}, \varepsilon) = \sqrt{2n\sqrt{\bar{m}}/\varepsilon}$, then $m_{LL} + m_{L_1H} \geq m - 64\varepsilon\bar{m}$.*

3 Upper Bound

In this section, we prove an upper bound on the number of queries an algorithm with access to Deg, Neigh, IS needs to make in order to achieve a $1 \pm \varepsilon$ estimation of the number of edges in a graph. Specifically, we prove the following:

▶ **Theorem 9.** *There exists an algorithm Estimate-Edges(n, ε, G) that takes as input integer $n \geq 1$, parameter $\varepsilon \in (0, 1]$, and access to the oracles Deg, Neigh, IS of a graph $G = ([n], E)$ with $m = |E|$. Estimate-Edges makes an expected number of $O\left(\frac{\log n}{\varepsilon^{5/2}} \min\left(\sqrt{\frac{n}{\sqrt{m}}}, \sqrt{m}\right)\right)$ queries to the oracles and outputs a number $\tilde{\boldsymbol{m}}$ such that with probability at least $2/3$ satisfies $(1-\varepsilon)m \leq \tilde{\boldsymbol{m}} \leq (1+\varepsilon)m$.*

We prove Theorem 9 through Lemma 22, which is logically equivalent. We use the parameters and notation described in Table 1 throughout the paper.

Table 1 Notation and parameters used throughout this section.

Symbol	Description
$G = (V, E)$	The input graph.
n	Number of vertices in the graph ($n = \|V\|$).
m	Number of edges in the graph ($m = \|E\|$).
$\bar{m}$	Untrusted advice for the number of edges m, where $\bar{m} > 0$.
$\tilde{\boldsymbol{m}}$	Random variable output of an estimation procedure.
k	Degree threshold: vertices of degree at most k are *low-degree*, and vertices of degree greater than k are *high-degree*, with $k \geq 0$.
k'	Secondary degree threshold, satisfying $0 \leq k' \leq k$.

3.1 Estimating m_{LL} (with advice)

We restate the following lemma from [6]:

▶ **Lemma 10** ([6])**.** *There exists a deterministic procedure Enumerate-Edges whose input is a graph $G = (V, E)$ accessible through the independent-set oracle and a vertex set $A \subseteq V$, and its output is a list of all edges in the subgraph induced on A's vertices. The query complexity of this procedure is $O(1 + |E_A| \log n)$, where $n = |V|$ is the number of vertices and E_A is the set of edges in the subgraph induced on A's vertices.*

Procedure Sample-LL-Edges-Sparse, implemented in Algorithm 1, is an unbiased estimator for $\frac{m_{LL}}{\bar{m}}$. It draws a set $\mathbf{S} \subseteq V$ that every vertex belongs to with probability $1/\sqrt{\bar{m}}$, and then enumerates the edges in the subgraph induced on $\mathbf{S}$. Only edges between two low-degree vertices $(\deg(u), \deg(v) \leq k)$ are counted.

▶ **Lemma 11.** *The output of Procedure Sample-LL-Edges-Sparse (Algorithm 1) is a random variable whose expected value is $\frac{m_{LL}}{\bar{m}}$ and variance is at most $\left(1 + \frac{2k}{\sqrt{\bar{m}}}\right)\frac{m_{LL}}{\bar{m}}$. Moreover, the procedure's expected query complexity is $O(1 + \frac{m}{\bar{m}} \log n)$.*

In procedure Estimate-LL-Edges-Advice (Algorithm 2), we estimate m_{LL} by averaging the outcomes of $O\left(\frac{1+k/\sqrt{\bar{m}}}{\varepsilon^2}\right)$ invocations of Sample-LL-Edges-Sparse. Since each invocation has expected value $m_{LL}/\bar{m}$, this averaging yields good concentration around the mean.

Algorithm 1: Procedure Sample-LL-Edges-Sparse$(G, \bar{m}, k)$.

Input: Integer $n \geq 1$, parameters $k \geq 0$, $\bar{m} > 0$.
Input: Access to the oracles IS, Deg of a graph $G = ([n], E)$ with $m = |E|$.
Output: An unbiased estimation $\mathbf{X}$, for $\frac{m_{LL}}{\bar{m}}$.

1. Draw a subset $\mathbf{S} \subseteq V$ that every vertex belongs to with probability $1/\sqrt{\bar{m}}$, independently.
2. Run Enumerate-Edges$(G, \mathbf{S})$, and let $E_{\mathbf{S}}$ be the resulting list of edges.
3. Initialize $\mathbf{X} \leftarrow 0$.
4. For each edge $(\boldsymbol{u}, \boldsymbol{v}) \in E_{\mathbf{S}}$:
   a. Let $d_{\boldsymbol{u}} \leftarrow \deg(\boldsymbol{u})$ (via Deg).
   b. Let $d_{\boldsymbol{v}} \leftarrow \deg(\boldsymbol{v})$ (via Deg).
   c. If $d_{\boldsymbol{u}} \leq k$ and $d_{\boldsymbol{v}} \leq k$:
      i. Increment $\mathbf{X} \leftarrow \mathbf{X} + 1$.
5. Return $\mathbf{X}$.

Algorithm 2: Procedure Estimate-LL-Edges-Advice$(n, \varepsilon, G, \bar{m}, k)$.

Input: Integer $n \geq 1$, parameters $\varepsilon \in (0, 1], k \geq 0$, $\bar{m} > 0$.
Input: Access to the oracles IS, Deg of a graph $G = ([n], E)$ with $m = |E|$.
Output: A number $\tilde{\boldsymbol{m}}_{LL}$.

1. Let $\boldsymbol{c} \leftarrow 0$, $q \leftarrow \left\lceil \frac{600}{\varepsilon^2} \max\left(1, \frac{k}{\sqrt{\bar{m}}}\right) \right\rceil$.
2. For $i = 1, ..., q$:
   a. Run Sample-LL-Edges-Sparse$(n, G, \bar{m}, k)$ and let $\mathbf{X}_i$ be its result.
   b. Increment $\boldsymbol{c} \leftarrow \boldsymbol{c} + \frac{1}{q}\mathbf{X}_i$.
3. Return $\tilde{\boldsymbol{m}}_{LL} = \bar{m} \cdot \boldsymbol{c}$.

▶ **Lemma 12.** *Let $\tilde{\boldsymbol{m}}_{LL}$ be the (random) output of procedure Estimate-LL-Edges-Advice (Algorithm 2).*

- *If $\bar{m} \geq \frac{1}{4}m_{LL}$, then with probability at least $1 - \frac{m_{LL}}{200\bar{m}}$, $|\tilde{\boldsymbol{m}}_{LL} - m_{LL}| \leq \varepsilon\bar{m}$.*
- *If $\bar{m} < m_{LL}$, then with probability at least $1 - \frac{\bar{m}}{200m_{LL}}$, $\tilde{\boldsymbol{m}}_{LL} > (1 - \varepsilon)\bar{m}$.*

Moreover, the expected query complexity is $O\left(\frac{\log n}{\varepsilon^2}\left(1 + \frac{k}{\sqrt{\bar{m}}}\right)\left(1 + \frac{m}{\bar{m}}\right)\right)$. Note that if $\frac{1}{4}m_{LL} \leq \bar{m} < m_{LL}$, then both guarantees hold simultaneously.

3.2 Estimating m_{LH} (with advice)

In this subsection we provide an estimation algorithm for the number of edges between low-degree vertices ($\deg(u) \leq k$) and high-degree vertices ($\deg(u) > k$). Recall that the number of low-high edges is denoted by m_{LH}, and the set of these edges is denoted by E_{LH}.

We define the "very-low" degree threshold as $k' = \min(k, \bar{m}/\varepsilon k)$. Let V_{L_1} denote the set of vertices of degree at most k', let E_{L_1H} be the set of edges between vertices in V_{L_1} and high-degree vertices, and let $m_{L_1H} = |E_{L_1H}|$. Since $V_{L_1} \subseteq V_L$, it follows that $m_{L_1H} \leq m_{LH}$. Moreover, Lemma 7 implies that the number of edges between high-degree vertices and vertices of degree exceeding k' is at most $4\varepsilon\bar{m}$. Consequently, $m_{L_1H} \leq m_{LH} \leq m_{L_1H} + 4\varepsilon\bar{m}$.

The estimator for m_{L_1H}, implemented as procedure Estimate-L_1H-Edges-Advice (Algorithm 3), averages over $O(nk'/\varepsilon^2\bar{m})$ samples as follows: we uniformly draw a vertex $\boldsymbol{u}$ and a neighbor $\boldsymbol{v}$ of $\boldsymbol{u}$ (if $\deg(\boldsymbol{u}) = 0$, we proceed to the next sample). If $\boldsymbol{u}$'s degree is "very-low" (i.e., $\deg(\boldsymbol{u}) \le k'$) and $\boldsymbol{v}$'s degree is high ($\deg(\boldsymbol{v}) > k$), then we add $\deg(\boldsymbol{u})$ to a counter $\boldsymbol{c}$. For all other cases, we do not increment the counter. The result is normalized by an n factor.

Algorithm 3: Procedure Estimate-L_1H-Edges-Advice$(n, \varepsilon, G, \bar{m}, k)$.

Input: Integers $n \ge 1$, parameters $\varepsilon \in (0, 1], k \ge 0$, $\bar{m} > 0$.
Input: Access to the oracles Deg, Neigh of a graph $G = ([n], E)$ with $m = |E|$.
Output: An unbiased estimator $\tilde{\boldsymbol{m}}_{L_1H}$ for m_{L_1H}.

1. Initialize $\boldsymbol{c} \leftarrow 0$.
2. Let $k' \leftarrow \min\left(k, \frac{\bar{m}}{\varepsilon k}\right)$.
3. Let $t \leftarrow \left\lceil \frac{200nk'}{\varepsilon^2\bar{m}} \right\rceil$.
4. For $i = 1, ..., t$:
   a. Sample a vertex $\boldsymbol{u} \in V$ uniformly at random.
   b. Sample a neighbor $\boldsymbol{v}$ of $\boldsymbol{u}$ uniformly at random (via Neigh).
   c. Let $\boldsymbol{d}_u \leftarrow \deg(\boldsymbol{u})$ (via Deg).
   d. Let $\boldsymbol{d}_v \leftarrow \deg(\boldsymbol{v})$ (via Deg).
   e. If $\boldsymbol{d}_u \le k'$ and $\boldsymbol{d}_v > k$:
      i. Increment $\boldsymbol{c} \leftarrow \boldsymbol{c} + \boldsymbol{d}_u$.
5. Return $\tilde{\boldsymbol{m}}_{L_1H} = \frac{n}{t}\boldsymbol{c}$.

▶ **Lemma 13.** *Let $\tilde{\boldsymbol{m}}_{L_1,H}$ be the (random) output of Estimate-L_1H-Edges-Advice (Algorithm 3) when executed with parameters $(G, \varepsilon, \bar{m}, k)$.*

- *If $\bar{m} \ge \frac{1}{4}m_{L_1H}$, then with probability at least $1 - \frac{m_{L_1H}}{200\bar{m}}$, $|\tilde{\boldsymbol{m}}_{L_1H} - m_{L_1H}| \le \varepsilon\bar{m}$.*
- *If $\bar{m} < m_{L_1H}$, then with probability at least $1 - \frac{\bar{m}}{200m_{L_1H}}$, $\tilde{\boldsymbol{m}}_{L_1H} > (1-\varepsilon)\bar{m}$.*

Moreover, the query complexity is bounded by $O(n/\varepsilon^3 k)$.

3.3 Handling the case where $\bar{m} < m$

Beame's algorithm (Enumerate-Edges) is meant to be executed as a whole. For our algorithm, we need a more fine-grained analysis.

▶ **Lemma 14.** *Enumerate-Edges (Lemma 10) can be implemented with an additional guarantee: for every $1 \le t \le |E_A|$, the t-th edge is added to the list after performing at most $O(1+t\log n)$ queries. In other words, it has $O(1)$ initialization phase and amortized $O(\log n)$ "next-element" phase.*

Procedure Small-$\bar{m}$-Guard (Algorithm 6) separates the regime in which the advice $\bar{m}$ is valid ($\bar{m} \ge m$) from the regime in which it is substantially underestimated ($\bar{m} < m/4$) or when it is structurally wrong ($\max_u \deg(u) > \bar{m}$). In the former case, the procedure accepts with probability at least $1 - 1/1000$ (completeness), while in the latter it rejects with probability at least $1 - \bar{m}/1000m$ (soundness). The procedure searches for vertices whose degree exceeds $\bar{m}$ and applies a moderated variant of Sample-LL-Edges-Sparse in order to reject instances in which the number of edges is significantly larger than what is consistent with the assumption $\bar{m} \ge m$.

For readability, we provide a parameterized algorithm and a corresponding parameterized analysis for completeness $1 - c$ and soundness $1 - c\bar{m}/m$, where $c \leq 1$. For the lemma statement, we use $c = 1/1000$.

Case 0 (trivial)

$m = O(\text{poly}(1/c))$. This is an easy case since we can just look for $O(\text{poly}(1/c))$ edges at the cost of $O_c(\log n)$ queries to find out whether or not $\bar{m} \geq \min(m, \text{poly}(1/c)) = m$.

Case I (typical)

$m_{LL} \geq \frac{3}{4}m$, where the threshold degree k is defined as $\min(\bar{m}, n-1)$.

Case II (very-high degree)

There exists a vertex of degree at least $\sqrt{\bar{m}}\ln(m/c)$.

Case III (sparse)

$m = \Omega(\text{poly}(1/c))$ and $m_{LL} < \frac{3}{4}m$ (where the threshold degree k is defined as $\min(\bar{m}, n-1)$) and the maximum degree is at most $\sqrt{\bar{m}}\ln(m/c)$.

Observe that Case I is "quantitative" (to be resolved by large-deviation inequalities), whereas Case II is a "qualitative" (to be resolved by looking for a witness), and therefore, they are treated using different algorithms. Case III is analyzed through the qualitative algorithm resolving Case II.

3.3.1 Quantity test (Case I)

In the quantity test (Guard-Quantity, Algorithm 4) we execute an edge sampler similar to Estimate-LL-Edges-Advice (Algorithm 2), but we use the lazy variant of Enumerate-Edges (Lemma 14) to keep the query complexity low even if $\bar{m}$ is significantly smaller than m. Additionally, if we happen to encounter a vertex whose degree is greater than $\bar{m}$, then we reject immediately.

▶ **Observation 15.** *Considering Algorithm 4 (Guard-Quantity), ignoring the possibility of an early termination (but without counting the high-degree-adjacent edges that would cause it), the gain in* $\mathbf{X}$ *in every iteration distributes the same as in Algorithm 1 (Sample-LL-Edges-Sparse).*

▶ **Observation 16.** *Assume that* $\bar{m} \geq 1$. *Considering Algorithm 4 (Guard-Quantity), ignoring the possibility of an early termination (but without counting the high-degree-adjacent edges that would cause it), the gain in* $\mathbf{X}$ *in every iteration has expected value* $\frac{m_{LL}}{\bar{m}}$ *and its variance is bounded by* $3\sqrt{\bar{m}} \cdot \frac{m_{LL}}{\bar{m}}$.

3.3.2 Quality test (Case II, Case III)

In the quality test (Guard-Quality, Algorithm 5) we use a similar logic to the quantitative test, but instead of having independent iterations, we choose the sets such that every vertex belongs to exactly one iteration. This guarantees that every potential witness belongs to the union of the iteration sets, which is not guaranteed in Algorithm 4. We reject as soon as we encounter a vertex whose degree is strictly greater than $\bar{m}$, or when we find significantly more edges than what we expect to if we would run Estimate-LL-Edges-Advice.

Algorithm 4: Procedure Guard-Quantity$(c; G, \bar{m})$.

Complexity: $O(\frac{1}{c}\sqrt{\bar{m}}\log n)$.
Completeness: If $\bar{m} \geq m$, then we accept with probability at least $1 - c/2$.
Soundness: If $\bar{m} < \frac{1}{4}m$ and $m_{LL} \geq \frac{3}{4}m$, then we reject with probability at least $1 - \frac{c\bar{m}}{m}$.
Soundness: If $m_{LL} < \frac{1}{2}m$ and the maximum degree in G is less than $\sqrt{\bar{m}}\ln(m/c)$, then we reject with probability at least $1 - \frac{c\bar{m}}{m}$.

1. Initialize $\mathbf{X} \leftarrow 0$.
2. Let $t \leftarrow \lceil \frac{96}{c}\sqrt{\bar{m}} \rceil$.
3. For t times:
   a. Draw a set $\mathbf{S} \subseteq V$ that every vertex belongs to with probability $\frac{1}{\sqrt{\bar{m}}}$, iid.
   b. Initialize Enumerate-Edges, let $L_{\mathbf{S}}$ be the result, accessible sequentially.
   c. While $L_{\mathbf{S}}$ has an unfeteched edge:
      i. Fetch $(\boldsymbol{u}, \boldsymbol{v}) \in L_{\mathbf{S}}$.
      ii. Let $d_{\boldsymbol{u}} \leftarrow \deg(\boldsymbol{u})$ via Deg query.
      iii. Let $d_{\boldsymbol{v}} \leftarrow \deg(\boldsymbol{v})$ via Deg query.
      iv. If $d_{\boldsymbol{u}} > \bar{m}$ or $d_{\boldsymbol{v}} > \bar{m}$:
         A. Return REJECT.
      v. Increment $\mathbf{X} \leftarrow \mathbf{X} + 1$.
      vi. If $\mathbf{X} \geq (5/4)t$:
         A. Return REJECT.
4. Return ACCEPT.

3.3.3 The guard procedure

The guard procedure is merely a conjunction of a deterministic fixed-count edge enumeration (to detect Case 0), the quantitative guard procedure (to detect Case I) and the qualitative procedure (to detect Case II and Case III).

3.4 Estimating m (with advice)

We combine the previously described partial-estimation procedures into two estimation procedures: Estimate-Edges-Advice-High, which is optimized for large ms, and Estimate-Edges-Advice-High, which is optimized for small ms.

In Estimate-Edges-Advice-High, we use the threshold degree $k = \Theta(\sqrt{n\sqrt{\bar{m}}/\varepsilon})$, unless $m = \Omega(\varepsilon n^2)$, in which case we use $k = n - 1$.

▶ **Lemma 17.** *Let $\tilde{\boldsymbol{m}}$ be the (random) output of Estimate-Edges-Advice-High (Algorithm 7) when given parameters ε, $G = (V, E)$ and $\bar{m}$. If $\bar{m} \geq \frac{1}{4}m$, then $|\tilde{\boldsymbol{m}} - m| \leq 70\varepsilon\bar{m}$ with probability at least $1 - \frac{m}{100\bar{m}}$.*

▶ **Lemma 18.** *Let $\tilde{\boldsymbol{m}}$ be the (random) output of Estimate-Edges-Advice-Low (Algorithm 8) when given parameters ε, $G = (V, E)$ and $\bar{m}$. The following is guaranteed for $m = |E|$ (unknown to the algorithm):*

- *If $\bar{m} \geq m$, then with probability at least $1 - \frac{m}{200\bar{m}} - \frac{1}{1000}$, $|\tilde{\boldsymbol{m}} - m| \leq \varepsilon\bar{m}$.*
- *If $\bar{m} \geq \frac{1}{4}m$, then with probability at least $1 - \frac{m}{200\bar{m}}$, $|\tilde{\boldsymbol{m}} - m| \leq \varepsilon\bar{m}$ or $\tilde{\boldsymbol{m}} > (1 - \varepsilon)\bar{m}$.*
- *$1 \leq \bar{m} < m$, then with probability at least $1 - \frac{\bar{m}}{200m}$, $\tilde{\boldsymbol{m}} > (1 - \varepsilon)\bar{m}$.*

Algorithm 5: Procedure Guard-Quality$(c; G, \bar{m})$.

Complexity: $O(\frac{1}{c}\sqrt{\bar{m}}\log n)$.
Completeness: If $\bar{m} \geq m$, then we accept with probability at least $1 - c/2$.
Soundness: If $\max_u \deg(u) \geq \sqrt{\bar{m}}\ln(m/c)$, then we reject with probability at least $1 - \frac{c\bar{m}}{m}$.

1. Let $t \leftarrow \lceil\sqrt{\bar{m}}\rceil$.
2. For each $u \in V$:
   a. Draw $\boldsymbol{i}_u \in \{1, \ldots, t\}$ uniformly and independently.
3. Initialize $\mathbf{X} \leftarrow 0$.
4. For i from 1 to t:
   a. Let $\mathbf{S}_i \leftarrow \{u \in V : \boldsymbol{i}_u = i\}$.
   b. Initialize Enumerate-Edges, let $L_{\mathbf{S}}$ be the result, accessible sequentially.
   c. While $L_{\mathbf{S}}$ has an unfeteched edge:
      i. Fetch $(\boldsymbol{u}, \boldsymbol{v}) \in L_{\mathbf{S}}$.
      ii. Let $d_{\boldsymbol{u}} \leftarrow \deg(\boldsymbol{u})$ via Deg query.
      iii. Let $d_{\boldsymbol{v}} \leftarrow \deg(\boldsymbol{v})$ via Deg query.
      iv. If $d_{\boldsymbol{u}} > \bar{m}$ or $d_{\boldsymbol{v}} > \bar{m}$:
         A. Return REJECT.
      v. Increment $\mathbf{X} \leftarrow \mathbf{X} + 1$.
      vi. If $\mathbf{X} \geq 2t/c$:
         A. Return REJECT.
5. Return ACCEPT.

3.5 Estimating m without any advice

In this subsection we introduce our main procedure, Estimate-Edges. At a high level, the algorithm begins with an extreme initial estimate for $\bar{m}$ and iteratively refines it toward an accurate value of m. A straightforward serial execution of these iterations (a simple for-loop) achieves correctness with high probability, but its query complexity is small only with high probability rather than in expectation. Therefore, we execute the iterations in an iterative-deepening schedule, i.e., in rounds where we run the prefixes (0), (0, 1), (0, 1, 2), and so on.

We start by presenting a single iteration of the algorithm (Algorithm 9). We then give an iteration-bounded variant, Estimate-Edges-Bounded (Algorithm 10), which attains a correct estimate with high probability once the number of iterations is sufficiently large; however, this version guarantees low query complexity only with high probability, not in expectation.

Finally, we provide the iteration-unbounded version, Estimate-Edges (Algorithm 11), which preserves correctness while ensuring that the expected query complexity remains small.

To analyze Estimate-Edges-Iteration (and its callers) when executed on an input graph G with n vertices (where n is known to the algorithm) and m edges (unknown to the algorithm), we define the following parameters:

- $\ell_{\text{big}}(G)$ – the unique integer for which $\sqrt{2}m \leq 2^{-\ell}n^2 < \sqrt{8}m$.
- $\ell_{\text{small}}(G)$ – the unique integer for which $\sqrt{2}m \leq 2^{\ell/2} < 2m$.
- $\ell_0(G) = \min(\ell_{\text{big}}(G), \ell_{\text{small}}(G))$.

For convenience, we usually omit the G parameter and refer to ℓ_{big}, ℓ_{small} and ℓ_0.

Algorithm 6: Procedure Small-$\bar{m}$-Guard$(G, \bar{m})$.

Complexity: $O(\sqrt{\bar{m}} \log n)$.
Completeness: If $\bar{m} \geq m$ and $\max_u \deg(u) \leq \bar{m}$, then we accept with probability at least $1 - \frac{1}{1000}$.
Soundness: If $\bar{m} < \frac{1}{4}m$, then we reject with probability at least $1 - \frac{\bar{m}}{1000m}$.
Soundness: If $\max_u \deg(u) > \bar{m}$, then we reject with probability at least $1 - \frac{\bar{m}}{1000m}$.

1. Let $c \leftarrow 1/1000$.
2. Let $\bar{m}^* \leftarrow \max(16(1 + \ln c^{-1})^{14}, 1/c)$.
3. Initialize Enumerate-Edges for the whole graph, let L be the result list, accessible sequentially.
4. Initialize $X \leftarrow 0$.
5. While $X \leq \bar{m}^*$ and L has an unfeteched edge:
   a. Consume an arbitrary edge of L.
   b. Increment $X \leftarrow X + 1$.
6. If $X > \bar{m}$:
   a. Return REJECT.
7. Run Guard-Quantity$(c; G, \bar{m})$.
8. If The quantity guard rejects:
   a. Return REJECT.
9. Run Guard-Quality$(c; G, \bar{m})$.
10. If The quality guard rejects:
    a. Return REJECT.
11. Return ACCEPT.

Algorithm 7: Procedure Estimate-Edges-Advice-High$(\varepsilon, G, \bar{m})$.

Input: A graph $G = (V, E)$ (V is explicitly given) accessible through Deg, Neigh, IS.
Input: An accuracy parameter $\varepsilon > 0$.
Input: $n \geq 2/\varepsilon^2$.
Input: An untrusted advice $\bar{m} > 0$.
Output: A number $\tilde{\boldsymbol{m}}$.
Complexity: $O\left(\frac{\log n}{\varepsilon^{5/2}}(1 + \frac{m}{\bar{m}})\sqrt{n/\sqrt{\bar{m}}}\right)$.

1. If $\bar{m} \geq \frac{1}{4}\varepsilon n^2$:
   a. Set $k \leftarrow n - 1$.
   b. Run Estimate-LL-Edges-Advice$(n, \varepsilon, G, \bar{m}, k)$ and return the result as $\tilde{\boldsymbol{m}}$.
   c. Return $\tilde{\boldsymbol{m}}$.
2. Else:
   a. Set $k \leftarrow \frac{\sqrt{2}}{\sqrt{\varepsilon}} n^{\frac{1}{2}} \bar{m}^{\frac{1}{4}}$.
   b. Run Estimate-LL-Edges-Advice$(n, \varepsilon, G, \bar{m}, k)$ and let $\tilde{\boldsymbol{m}}_{LL}$ be the result.
   c. Run Estimate-L$_1$H-Edges-Advice$(n, \varepsilon, G, \bar{m}, k)$ and let $\tilde{\boldsymbol{m}}_{L_1H}$ be the result.
   d. Return $\tilde{\boldsymbol{m}} = \tilde{\boldsymbol{m}}_{LL} + \tilde{\boldsymbol{m}}_{L_1H}$.

Algorithm 8: Procedure Estimate-Edges-Advice-Low$(\varepsilon, G, \bar{m})$.

Input: A graph $G = (V, E)$ (V is explicitly given) accessible through Deg, Neigh, IS.
Input: An accuracy parameter $\varepsilon > 0$.
Input: An untrusted advice $\bar{m} \geq 1$.
Output: A number $\tilde{\boldsymbol{m}}$.
Complexity: $O(\frac{\log n}{\varepsilon^2}\sqrt{\bar{m}})$.

1. Set $k \leftarrow \min(\bar{m}, n-1)$.
2. If Small-$\bar{m}$-Guard$(G, \bar{m})$ accepts:
   a. Run Estimate-LL-Edges-Advice$(n, \frac{\varepsilon}{10}, G, \bar{m}, k)$ and return the result as $\tilde{\boldsymbol{m}}$.
3. Return $+\infty$.

Algorithm 9: Procedure Estimate-Edges-Iteration$(n, \varepsilon, G, \ell)$.

Input: $0 < \varepsilon \leq \frac{1}{15}$.
Input: G has n vertices (explicitly given) and $m \geq 1$ edges (unknown to the algorithm).
Complexity: $O(2^{\max(0,\ell-\ell_0)} \cdot 2^{\ell/4} \cdot \log n/\varepsilon^{5/2})$.

1. Let $\bar{m}_{\text{big}} \leftarrow \frac{n^2}{2^\ell}$, $\bar{m}_{\text{small}} \leftarrow 2^{\ell/2}$. (*cannot* assume that $\bar{m}_{\text{big}} \geq \bar{m}_{\text{small}}$)
2. Call Estimate-Edges-Advice-High$(n, \frac{1}{1000}\varepsilon, G, \bar{m}_{\text{big}})$ and let $\tilde{\boldsymbol{m}}_{\text{big}}$ be the result.
3. If $\frac{1}{4}\bar{m}_{\text{big}} \leq \tilde{\boldsymbol{m}}_{\text{big}} \leq \frac{4}{5}\bar{m}_{\text{big}}$:
   a. Return $\tilde{\boldsymbol{m}}_{\text{big}}$ as $\tilde{\boldsymbol{m}}$.
4. Call Estimate-Edges-Advice-Low$(n, \frac{1}{10}\varepsilon, G, \bar{m}_{\text{small}})$ and let $\tilde{\boldsymbol{m}}_{\text{small}}$ be the result.
5. If $\tilde{\boldsymbol{m}}_{\text{small}} \leq \frac{1}{\sqrt[4]{2}}\bar{m}_{\text{small}}$:
   a. Return $\tilde{\boldsymbol{m}}_{\text{small}}$ as $\tilde{\boldsymbol{m}}$.
6. Return REJECT.

▶ **Lemma 19.** *Assume that $\varepsilon \leq \frac{1}{15}$. If $\ell \leq \ell_0 - 2$, then with probability at least $1 - \frac{1}{50} \cdot 2^{(\ell-\ell_0)/2}$, Estimate-Edges-Iteration (Algorithm 9) either rejects or returns a number in the range $(1 \pm \varepsilon)m$. If $\ell = \ell_0 - 1$, then this probability is at least $1 - \frac{1}{100} \cdot 2^{(\ell-\ell_0)/2} - \frac{1}{100}$.*

▶ **Lemma 20.** *For $\varepsilon \leq \frac{1}{15}$, if $\ell = \ell_0$, then Estimate-Edges-Iteration (Algorithm 9) returns a number in the range $(1 \pm \varepsilon)m$ with probability at least $1 - \frac{1}{55}$.*

▶ **Lemma 21.** *Assume that $\varepsilon \leq \frac{1}{15}$. Considering Estimate-Edges-Bounded:*

- *If $\ell_{\max} \leq \ell_0 - 2$, then the probability of the ℓth call to Estimate-Edges-Iteration to return a non-reject value outside the range $(1 \pm \varepsilon)m$ is bounded by $\frac{2+\sqrt{2}}{50} \cdot 2^{(\ell-\ell_0)/2}$.*
- *If $\ell_{\max} = \ell_0 - 1$, then the probability to return a number outside the range $(1 \pm \varepsilon)m$ is bounded by $\frac{3}{50}$.*
- *If $\ell_{\max} \geq \ell_0$, then the probability to return a number in the range $(1 \pm \varepsilon)m$ is at least $1 - 3/34$.*

▶ **Lemma 22.** *For $\varepsilon \leq \frac{1}{15}$ and an input graph G with n vertices and m edges. Estimate-Edges (Algorithm 11) returns a number in the range $(1 \pm \varepsilon)m$ with probability at least $2/3$. Moreover, its expected query complexity bounded by $O\left(\frac{\log n}{\varepsilon^{5/2}} \min\left(\sqrt{m}, \sqrt{\frac{n}{\sqrt{m}}}\right)\right)$.*

Algorithm 10: Procedure Estimate-Edges-Bounded$(n, \varepsilon, G, \ell_{\max})$.

Input: $0 < \varepsilon \leq \frac{1}{10}$.
Input: G has n vertices (explicitly given) and m edges (unknown to the algorithm).
Input: $m \geq 1$.

1. For ℓ from 0 to $\ell_{\max}$:
   a. Cal Estimate-Edges-Iteration$(n, \varepsilon, G, \ell)$ and let $\tilde{\boldsymbol{m}}$ be its result.
   b. If $\tilde{\boldsymbol{m}} \neq$ REJECT:
      i. Return $\tilde{\boldsymbol{m}}$.
2. Return REJECT.

Algorithm 11: Procedure Estimate-Edges(n, ε, G).

Input: Integer n, parameter $\varepsilon \in (0, 1]$.
Input: Access to the oracles Deg, Neigh, IS of a graph $G = ([n], E)$.
Output: A number $\tilde{\boldsymbol{m}}$, an estimate for m.

1. If $\varepsilon > \frac{1}{15}$:
   a. Set $\varepsilon \leftarrow \frac{1}{15}$.
2. If IS$(V) = 1$:
   a. Return 0.
3. If $n < 2/\varepsilon^2$:
   a. For each $u \in V$:
      i. Let $d_u \leftarrow \deg(u)$ (via Deg).
   b. Return $\frac{1}{2}\sum_{u \in V} d_u$.
4. Let $\ell \leftarrow 0$.
5. While TRUE:
   a. Call Estimate-Edges-Bounded$(n, \varepsilon, G, \ell)$ and let $\tilde{\boldsymbol{m}}$ be its result.
   b. If $\tilde{\boldsymbol{m}} \neq$ REJECT:
      i. Return $\tilde{\boldsymbol{m}}$.
   c. $\ell \leftarrow \ell + 1$.

4 Lower Bound

In this section we prove the following lower-bound for edge estimation in the hybrid model. In particular, the lower bound follows from the following (stronger) statement.

▶ **Theorem 23.** *Consider any algorithm that is given query access to an unknown graph $G = (V, E)$ on n vertices and m edges via degree queries, neighbor queries and IS queries. Let $\varepsilon \in (0, 1)$ and define $R \stackrel{\text{def}}{=} \frac{1}{30} \min\left(\sqrt{m}, \sqrt{\frac{n}{\varepsilon\sqrt{m}}}\right)$. For any such algorithm whose expected number of queries is q (where q may depend on G), the probability that the algorithm outputs an estimate $\widehat{\boldsymbol{m}}$ satisfying $(1-\varepsilon)m \leq \widehat{\boldsymbol{m}} \leq (1+\varepsilon)m$ is at most $\frac{1}{2} + q/R$.*

Theorem 23 above bounds the success probability of any algorithm as a function of its expected number of queries. Rearranging this bound and applying Markov's inequality yields the following corollary for algorithms that succeed with probability at least 2/3.

▶ **Corollary 24.** *Let $\varepsilon \in (0,1)$. Every algorithm whose input is an access to a graph G of m edges (where m is unknown to the algorithm) through degree queries, neighbor queries and IS queries, whose output $\widehat{\boldsymbol{m}}$ satisfies $(1-\varepsilon)m \leq \widehat{\boldsymbol{m}} \leq (1+\varepsilon)m$ with probability at least $2/3$, must make $\Omega\left(\min\left(\sqrt{m}, \sqrt{\frac{n}{\varepsilon\sqrt{m}}}\right)\right)$ queries.*

The proof of Theorem 23 proceeds via a reduction from a specially designed string-testing problem to the problem of estimating the number of edges in a graph with a particular structure.

4.1 The string distinction problem

Let s be a string of length n over the alphabet $\Sigma = \{\mathrm{A}, \mathrm{K}, \mathrm{L}, \mathrm{H}\}$. We define a duality map as follows: A and K are the dual of themselves ($\overline{\mathrm{A}} = \mathrm{A}$, $\overline{\mathrm{K}} = \mathrm{K}$), and L and H are the dual of each other ($\overline{\mathrm{L}} = \mathrm{H}$, $\overline{\mathrm{H}} = \mathrm{L}$).

For $\sigma \in \Sigma$ and a string $s \in \Sigma^n$, let $n_\sigma(s) = |\{1 \leq i \leq n : s(i) = \sigma\}|$. For every $1 \leq i \leq n$, let

$$U_i \stackrel{\text{def}}{=} \begin{cases} \emptyset, & \text{if } s(i) = \mathrm{A}, \\ \{\, j \in [n] \setminus \{i\} : s(j) = \overline{s(i)} \,\}, & \text{otherwise.} \end{cases}$$

We assume that s is accessible through two oracles:

- **Full query.** Given an index $1 \leq i \leq n$, the oracle returns three entries:
  - The symbol $s(i)$.
  - The value $|U_i|$.
  - A uniformly random index from U_i, or $\perp$ if $U_i = \emptyset$.
- **Light query.** Given an index $1 \leq i \leq n$, the oracle returns the symbol $s(i)$.

We consider the task of distinguishing between strings of length n for which $n_{\mathrm{L}}(s) \cdot n_{\mathrm{H}}(s) = 0$ and strings for which $n_{\mathrm{L}}(s) \cdot n_{\mathrm{H}}(s) > \varepsilon\binom{n_{\mathrm{K}}(s)}{2}$. We say that an algorithm *distinguishes* between the above cases if one of the following occurs:

▶ **Definition 25** (Full-query distinguishing predicate)**.** *An algorithm performs a full query on a vertex i for which $s(i) \in \{\mathrm{L}, \mathrm{H}\}$ and obtained a non-$\perp$ index as the third entry of the result tuple.*

▶ **Definition 26** (Light-query distinguishing predicate)**.** *An algorithm performs a light query on a vertex i for which $s(i) = \mathrm{H}$ and another light query on a vertex j for which $s(j) = \mathrm{L}$.*

Observe that if either predicate occurs, the algorithm can deduce that $n_{\mathrm{L}}(s) \cdot n_{\mathrm{H}}(s) \neq 0$. Conversely, if neither predicate occurs, the algorithm cannot distinguish between the case where all $\{\mathrm{L}, \mathrm{H}\}$ symbols are the same (all L or all H) and the case where they are not.

4.2 Reduction to edge estimation

We can interpret a string s as a description of a graph of the following structure. A clique over the set $\{i : s(i) = \mathrm{K}\}$ and a biclique whose sides are $\{i : s(i) = \mathrm{L}\}$ and $\{i : s(i) = \mathrm{H}\}$. The number of edges is exactly $\binom{n_{\mathrm{K}}(s)}{2} + n_{\mathrm{L}}(s) \cdot n_{\mathrm{H}}(s)$. More formally,

▶ **Definition 27** (Reduction graph)**.** *Let $s \in \Sigma^n$ be a string. The* reduction graph *G_s is the graph with vertex set $\{1, \ldots, n\}$ defined as follows. An edge is placed between every pair of vertices i, j such that $s(i) = s(j) = \mathrm{K}$, and between every pair of vertices i, j such that*

$s(i) = \mathrm{L}$ *and* $s(j) = \mathrm{H}$. *Equivalently,* G_s *consists of a clique induced by the vertices labeled* K *and a complete bipartite graph between the vertices labeled* L *and those labeled* H. *Figure 1 illustrates the structure of* G_s.

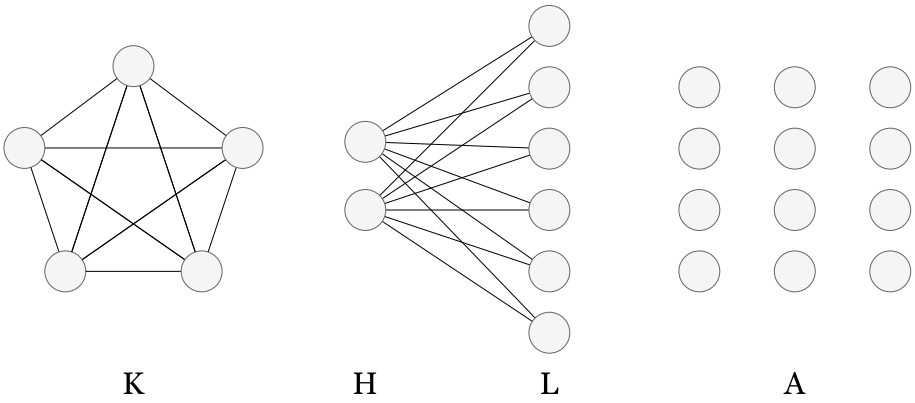

Figure 1 A diagram of the graph structure.

We describe a reduction that simulates graph queries using string queries. At the t-th step of the simulation, the following process is executed:

1. The t-th graph query, denoted Λ_t, is determined as a function of the outcomes of the preceding graph queries and the randomness of the algorithm.
2. The query Λ_t is translated into a sequence of string queries to the string s. Let $I_{t,1}, \ldots, I_{t,M_t}$ denote this sequence, where M_t is the number of string queries performed at step t. The value of M_t may depend on the outcomes of previous string queries, on the string s, and on the internal randomness of the translation algorithm.
3. Using the responses to the string queries $I_{t,1}, \ldots, I_{t,M_t}$, the simulation deterministically computes Y_t, the outcome of the t-th graph query.

At this point we describe the translation algorithm. Degree and neighbor queries are simulated using a single full string query, whereas an independent-set query is simulated by a randomized, adaptively determined sequence of light string queries.

Formally, the simulation proceeds as follows.

- **Degree query.** Given a vertex u we issue a full string query to s at position u and return the second component of the resulting triplet.
- **Neighbor query.** Given a vertex u we issue a full string query to s at position u and return the third component of the resulting triplet.
- **Independent-set query.** Given a vertex set S we first apply a uniformly random permutation to the vertices of S. We then process the vertices in this order, issuing light string queries, until one of the following stopping conditions is met:
  1. Two vertices labeled K are revealed, certifying that S is not an independent set.
  2. One vertex labeled L and one vertex labeled H are revealed, certifying that S is not an independent set.
  3. All vertices in S have been processed without triggering either of the above conditions, in which case S is declared an independent set.

▶ Remark 28. In the simulation of an independent-set query, whenever the symbol of a vertex has already been revealed, either by a previous string query or as part of the output of another query, we reuse this information and do not issue an additional query for that vertex.

The correctness of the simulation is trivial by the construction details. In the following we bound the simulation complexity. For local queries this is directly implied from the implementation:

▶ **Observation 29.** *The simulation of any local query (degree or neighbor) incurs exactly one full string query.*

It remains to bound the expected number of light string queries used in the simulation of an independent-set query.

▶ **Lemma 30.** *A vertex is said to be* involved *in a string query if it is either (i) directly queried, or (ii) appears in the output of that query. Every simulated graph query involves at most two vertices u such that $s(u) = \mathrm{K}$.*

Proof. We consider the two types of simulated graph queries.

For a local query (degree or neighbor), the simulation performs a single full string query. Such a query involves the queried vertex and at most one additional vertex appearing as part of the output (as the third entry). Hence, the simulation of a local query involves at most two vertices in total, and therefore at most two vertices labeled K.

For an independent-set query, the simulation issues a sequence of light string queries, each involving exactly one vertex (the queried vertex). By construction, the simulation halts immediately once it has revealed two vertices u with $s(u) = \mathrm{K}$. Consequently, throughout the execution of the simulation, at most two K-vertices can be involved in the string query. ◀

▶ **Lemma 31.** *Assume that $\boldsymbol{s} \in \{\mathrm{K}, \mathrm{A}\}^n$ is drawn from a distribution that is invariant under permutations of its coordinates; that is, all permutations of $\boldsymbol{s}$ have equal probability (while strings that are not permutations of one another may have different probabilities). Suppose that the t-th query issued by the graph algorithm is an independent-set query. If $t < \frac{1}{2} n_{\mathrm{K}}(\boldsymbol{s})$, then conditioned on any outcomes of the first $t-1$ queries, the simulation of the t-th query performs at most $\frac{2n}{n_{\mathrm{K}}(\boldsymbol{s}) - 2t}$ light string queries in expectation.*

4.3 A hard to distinguish construction

To construct a pair of hard-to-distinguish graphs, we define a few parameters. For given ε, n and n_k, let $d = n_k/2n$ (so that $\sqrt{2nd} = n_k$). Based on n, n_k, we define n_h and n_ℓ as following:

- $n_h = \left\lceil \sqrt{\varepsilon} n^{1/4} d^{3/4} \right\rceil$.
- $n_\ell = \left\lceil \sqrt{\varepsilon} n^{3/4} d^{1/4} \right\rceil$ if $n_h \geq 2$, and otherwise $n_\ell = \lceil \varepsilon n d \rceil$.

We state the following simple observation.

▶ **Observation 32.** *Let n_ℓ and n_h be defined as above. Then,*

1. *$n_h = 1$ if and only if $d \leq 1/(\varepsilon^{2/3} n^{1/3})$.*
2. *If $n_h = 1$, then $n_\ell + n_h \leq 3n/\sqrt{nd}$.*
3. *If $n_h \geq 2$, then $n_\ell + n_h \leq 4\sqrt{\varepsilon}\, n^{3/4} d^{1/4}$.*

We now define a distribution over hard-to-distinguish pairs of strings.

▶ **Definition 33** (Hard-to-distinguish strings)**.** *Let n_k, n_h, and n_ℓ be given. We draw a pair of strings $(\boldsymbol{s}_1, \boldsymbol{s}_2)$ as follows:*

- *Choose a set $\mathbf{K} \subseteq \{1, \ldots, n\}$ of size n_k uniformly at random.*
- *Define $\mathbf{A}_1 = \{1, \ldots, n\} \setminus \mathbf{K}$.*

- *Define* $\boldsymbol{s}_1$ *by* $\boldsymbol{s}_1(i) = \begin{cases} \text{K} & \text{if } i \in \mathbf{K}, \\ \text{A}, & \text{if } i \in \mathbf{A}_1. \end{cases}$
- *Choose a set* $\mathbf{H}_2 \subseteq \mathbf{A}_1$ *of size* n_h *uniformly at random.*
- *Choose a set* $\mathbf{L}_2 \subseteq (\mathbf{A}_1 \setminus \mathbf{H}_2)$ *of size* n_ℓ *uniformly at random.*
- *Define* $\mathbf{A}_2 = \mathbf{A}_1 \setminus (\mathbf{H}_2 \cup \mathbf{L}_2)$.
- *Define* $\boldsymbol{s}_2$ *by* $\boldsymbol{s}_2(i) = \begin{cases} \text{K} & \text{if } i \in \mathbf{K}, \\ \text{H} & \text{if } i \in \mathbf{H}_2, \\ \text{L}, & \text{if } i \in \mathbf{L}_2, \\ \text{A}, & \text{if } i \in \mathbf{A}_2. \end{cases}$

A pair of hard-to-distinguish graphs is obtained by sampling $(\boldsymbol{s}_1, \boldsymbol{s}_2)$ according to this distribution and defining the corresponding graphs $G_{\boldsymbol{s}_1}$ and $G_{\boldsymbol{s}_2}$.

▶ **Lemma 34.** *For given* $0 < \varepsilon \leq \frac{1}{11}$, $n \geq 16$ *and* $1 \leq n_k \leq \frac{1}{2}n$, *let* $R = \frac{1}{20}\min\left(n_k, \sqrt{\frac{n}{\varepsilon n_k}}\right)$. *Assume that we draw a hard-to-distinguish pair* $(\boldsymbol{s}_1, \boldsymbol{s}_2)$ *based on the parameters* ε, n *and* n_k. *Then, for any randomized algorithm making at most* q *graph queries in expectation, the probability that the sequences of query answers produced on inputs* $G_{\boldsymbol{s}_1}$ *and* $G_{\boldsymbol{s}_2}$ *differ is at most* q/R.

We can now finish with the proof of the lower bound.

Proof of Theorem 23. Consider a randomized algorithm $\mathcal{A}$ that makes at most q queries.

We draw a $(\boldsymbol{s}_1, \boldsymbol{s}_2)$ according to the parameters $\varepsilon' = 3\varepsilon$, n and $n_k = \lfloor\sqrt{2m}\rfloor$. Note that for $m \geq 6$ we have that $\sqrt{m} \leq n_k \leq 2\sqrt{m}$.

By Lemma 34, the query answer sequences of $\mathcal{A}$ on $G_{\boldsymbol{s}_1}$ and $G_{\boldsymbol{s}_2}$ are $20q/\min(n_k, \sqrt{n/(\varepsilon n_k)})$-close. Consequently, the probability that $\mathcal{A}$ distinguishes $G_{\boldsymbol{s}_1}$ from $G_{\boldsymbol{s}_2}$ is bounded by

$$\mathbf{Pr}[\mathcal{A}(G_{\boldsymbol{s}_1}) \neq \mathcal{A}(G_{\boldsymbol{s}_2})] \leq \frac{1}{2} + \frac{20q}{\min\left(n_k, \sqrt{\frac{n}{\varepsilon n_k}}\right)} \leq \frac{1}{2} + \frac{20q}{\min\left(\sqrt{m}, \sqrt{\frac{n}{\varepsilon\sqrt{2m}}}\right)} < \frac{1}{2} + \frac{q}{R}.$$

Let m_1, m_2 be the number of edges in $G_{\boldsymbol{s}_1}$ and $G_{\boldsymbol{s}_2}$ respectively. Observe that $m_1 = \binom{n_k}{2} \leq \frac{1}{2}(n_k)^2 \leq m$ and that $m_2 \geq m_1 + \varepsilon' m \geq (1+\varepsilon')m_1 = (1+3\varepsilon)m_1$. If $\mathcal{A}$ would estimate the number of edges in G within $(1 \pm \varepsilon)$-multiplicative error with probability at least $\frac{1}{2} + q/R$, then it would be a contradiction, since $(1+\varepsilon)m_1 < (1-\varepsilon)(1+3\varepsilon)m_1 \leq (1-\varepsilon)m_2$. ◀

References

1. Hasan Abasi and Bshouty Nader. On learning graphs with edge-detecting queries. In *Algorithmic Learning Theory*, pages 3–30. PMLR, 2019. URL: `http://proceedings.mlr.press/v98/abasi19a.html`.
2. Raghavendra Addanki, Andrew McGregor, and Cameron Musco. Non-adaptive edge counting and sampling via bipartite independent set queries. *arXiv preprint*, 2022. `doi:10.48550/arXiv.2207.02817`.
3. Dana Angluin and Jiang Chen. Learning a hidden graph using $O(\log n)$ queries per edge. *Journal of Computer and System Sciences*, 74(4):546–556, 2008. `doi:10.1016/J.JCSS.2007.06.006`.
4. Dana Angluin, Jiang Chen, and Manfred Warmuth. Learning a hidden hypergraph. *Journal of Machine Learning Research*, 7(79):2215–2236, 2006. URL: `http://jmlr.org/papers/v7/angluin06a.html`.
5. Sepehr Assadi, Michael Kapralov, and Sanjeev 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.

6 Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. In *Proceedings of the 2018 ACM Conference on Innovations in Theoretical Computer Science (ITCS '2018)*, 2018.

7 Lorenzo Beretta, Deeparnab Chakrabarty, and C Seshadhri. Faster estimation of the average degree of a graph using random edges and structural queries. In *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 939–971. SIAM, 2026. doi:10.1137/1.9781611978971.38.

8 Lorenzo Beretta and Jakub Tětek. Better sum estimation via weighted sampling. *ACM Transactions on Algorithms*, 20(3):1–33, 2024. URL: https://doi.org/10.1145/3650030.

9 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Hyperedge estimation using polylogarithmic subset queries. *arXiv preprint*, 2019. arXiv:1908.04196.

10 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. On triangle estimation using tripartite independent set queries. *Theory of Computing Systems*, 65(8):1165–1192, 2021. doi:10.1007/S00224-021-10043-Y.

11 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Faster counting and sampling algorithms using colorful decision oracle. *TOCT*, 16(2):1–19, 2024. doi:10.1145/3657605.

12 Arijit Bishnu, Debarshi Chanda, and Gopinath Mishra. Arboricity and random edge queries matter for triangle counting using sublinear queries. *arXiv preprint*, 2025. doi:10.48550/arXiv.2502.15379.

13 Arijit Bishnu, Arijit Ghosh, Gopinath Mishra, and Manaswi Paraashar. Efficiently sampling and estimating from substructures using linear algebraic queries. *arXiv preprint*, 2019. arXiv:1906.07398.

14 Chao L Chen and William H Swallow. Using group testing to estimate a proportion, and to test the binomial model. *Biometrics*, pages 1035–1046, 1990.

15 Xi Chen, Amit Levi, and Erik Waingarten. Nearly optimal edge estimation with independent set queries. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2916–2935. SIAM, 2020. doi:10.1137/1.9781611975994.177.

16 Holger Dell and John Lapinskas. Fine-grained reductions from approximate counting to decision. *TOCT*, 13(2):1–24, 2021. doi:10.1145/3442352.

17 Holger Dell, John Lapinskas, and Kitty Meeks. Approximately counting and sampling small witnesses using a colorful decision oracle. *SIAM Journal on Computing*, 51(4):849–899, 2022. doi:10.1137/19M130604X.

18 Robert Dorfman. The detection of defective members of large populations. *The Annals of mathematical statistics*, 14(4):436–440, 1943.

19 Talya Eden, Dana Ron, and C Seshadhri. Sublinear time estimation of degree distribution moments: The arboricity connection. *SIAM Journal on Discrete Mathematics*, 33(4):2267–2285, 2019. doi:10.1137/17M1159014.

20 Talya Eden and Will Rosenbaum. On sampling edges almost uniformly. In *Proceedings of the 1st Symposium on Simplicity in Algorithms (SOSA)*, pages 7–1, 2018.

21 Uriel 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. doi:10.1137/S0097539704447304.

22 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random Structures & Algorithms*, 32(4):473–493, 2008. doi:10.1002/RSA.20203.

23 Amit Levi, Ramesh Krishnan S Pallavoor, Sofya Raskhodnikova, and Nithin Varma. Erasure-resilient sublinear-time graph algorithms. *TOCT*, 14(1):1–22, 2021. doi:10.1145/3488250.

24 Dana Ron and Gilad Tsur. The power of an example: Hidden set size approximation using group queries and conditional sampling. *ACM Trans. Comput. Theory*, 8(4):1–19, 2016. URL: https://doi.org/10.1145/2930657.

25 Larry Stockmeyer. The complexity of approximate counting. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing (STOC)*, pages 118–126, 1983.

26 Larry Stockmeyer. On approximation algorithms for $\#P$. *SIAM Journal on Computing*, 14(4):849–861, 1985. doi:10.1137/0214060.

27 William H Swallow. Group testing for estimating infection rates and probabilities of disease transmission. *Phytopathology (USA)*, 1985.

28 Jakub Tětek and Mikkel Thorup. Edge sampling and graph parameter estimation via vertex neighborhood accesses. In *Proceedings of the 54th annual ACM SIGACT symposium on theory of computing (STOC)*, pages 1116–1129, 2022.

29 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 222–227. IEEE Computer Society, 1977.