Edge Estimation with Independent Set Oracles

PAUL BEAME, University of Washington, Seattle
SARIEL HAR-PELED, University of Illinois, Urbana–Champaign
SIVARAMAKRISHNAN NATARAJAN RAMAMOORTHY, University of Washington, Seattle
CYRUS RASHTCHIAN, University of California, San Diego
MAKRAND SINHA, Centrum Wiskunde & Informatika

We study the task of estimating the number of edges in a graph, where the access to the graph is provided via an independent set oracle. Independent set queries draw motivation from group testing and have applications to the complexity of decision versus counting problems. We give two algorithms to estimate the number of edges in an n-vertex graph, using (i) $\mathrm{polylog}(n)$ bipartite independent set queries or (ii) $n^{2/3}\,\mathrm{polylog}(n)$ independent set queries.

CCS Concepts: • **Theory of computation → Design and analysis of algorithms**; **Streaming, sublinear and near linear time algorithms**;

Additional Key Words and Phrases: Independent set queries, graph parameter estimation

ACM Reference format:
Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. 2020. Edge Estimation with Independent Set Oracles. *ACM Trans. Algorithms* 16, 4, Article 52 (September 2020), 27 pages.
https://doi.org/10.1145/3404867

1 INTRODUCTION

We investigate the problem of estimating the number of edges in a simple, unweighted, undirected graph $G = ([\![n]\!], E)$, where $[\![n]\!] := \{1, 2, \ldots, n\}$ and $m = |E|$. Here, the only access to the graph

A preliminary version of this article appeared in the proceedings of ITCS 2018 [7]. This work was done while S. Har-Peled was visiting the University of Washington on a sabbatical in 2017. The work was partially completed while C. Rashtchian was a graduate student at the Paul G. Allen School of CSE, University of Washington, Seattle, and while at Microsoft Research. During the course of this work, M. Sinha was a graduate student at the Paul G. Allen School of CSE, University of Washington, Seattle.
P. Beame was supported in part by NSF grant CCF-1524246. S. Har-Peled was supported in part by NSF AF awards CCF-1421231 and CCF-1217462. S. N. Ramamoorthy was supported by the NSF under agreements CCF-1149637, CCF-1420268, and CCF-1524251. M. Sinha was supported by the NSF under agreements CCF-1149637, CCF-1420268, and CCF-1524251.
Authors' addresses: P. Beame, Paul G. Allen School of CSE, University of Washington, Paul G. Allen Center, Box 352350,185 E Stevens Way NE, Seattle, WA 98195-2350, USA; email: beame@cs.washington.edu; S. Har-Peled, Department of Computer Science, University of Illinois, 201 N. Goodwin Avenue, Urbana, IL 61801-2302, USA; email: sariel@illinois.edu; S. N. Ramamoorthy, Paul G. Allen School of CSE, University of Washington, 5215 15th Ave NE, Apt 7, Seattle, WA 98105, USA; email: sivanr@cs.washington.edu; C. Rashtchian, Department of Computer Science and Engineering, University of California, San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0404, USA; email: crashtchian@eng.ucsd.edu; M. Sinha, Centrum Wiskunde & Informatika, Science Park 123, 1098 XG Amsterdam, Netherlands; email: makrand@cs.washington.edu.

1549-6325/2020/09-ART52 $15.00
https://doi.org/10.1145/3404867

is provided via an oracle that answers independent set queries. For a parameter $\varepsilon > 0$, we wish to output an estimate $\widetilde{m}$ satisfying $(1-\varepsilon)m \le \widetilde{m} \le (1+\varepsilon)m$ with high probability. We consider randomized, adaptive algorithms with access to one of the two following oracles:

- ⋆ *BIS (bipartite independent set) oracle*: Given disjoint subsets $U, V \subseteq [\![n]\!]$, a BIS query answers whether there is no edge between U and V in G. Formally, the oracle returns whether $m(U,V) = 0$, where $m(U,V)$ denotes the number of edges with one endpoint in U and the other in V.
- ⋆ *IS (independent set) oracle*: Given a subset $U \subseteq [\![n]\!]$, an IS query answers whether U satisfies $m(U) = 0$, where $m(U)$ denotes the number of edges with both endpoints in U.

Previous work on graph parameter estimation has primarily focused on *local* queries, such as (i) *degree* queries (which output the degree of a vertex v), (ii) *edge existence* queries (which answer whether a pair $\{u, v\}$ forms an edge), or (iii) *neighbor* queries (which provide the i^{th} neighbor of a vertex v). Feige [23] and Goldreich and Ron [25] prove that there are cases where a polynomial number of such local queries are required.

These queries can only obtain *local* information about the graph. This motivates an investigation of other types of natural queries that may enable efficient parameter estimation. The independent set queries described previously generalize an edge existence query, and their non-locality opens the door for subpolynomial query algorithms for various graph parameter estimation tasks [2–5, 27, 30].

1.1 Motivation and Related Work

The most relevant motivation for BIS and IS queries comes from the area of sub-linear time algorithms for graph parameter estimation. BIS and IS queries also have interesting connections to the classical area of group testing, to emptiness versus counting questions in computational geometry, and to the complexity of decision versus counting problems.

Graph parameter estimation. Feige [23] showed how to use $O(\sqrt{n}/\varepsilon)$ degree queries to output $\widetilde{m}$ that satisfies $m \le \widetilde{m} \le (2+\varepsilon)m$, where $m = |E|$. Moreover, he showed that any algorithm achieving better than a 2-approximation must use a nearly linear number of degree queries. Goldreich and Ron [25] showed that by using both degree and neighbor queries, the approximation improves to $(1-\varepsilon)m \le \widetilde{m} \le (1+\varepsilon)m$ by using $O((n/\sqrt{m})\,\mathrm{poly}(\log m, 1/\varepsilon))$ queries. It is worth noting that Feige [23] and Goldreich and Ron [25] have identified certain hard instances showing that these upper bounds cannot be improved, up to polylog factors. Aliakbarpour et al. [1] do better than the lower bound of Goldreich and Ron [25] by allowing one to sample edges randomly.

Related work approximates the number of stars [26], the minimum vertex cover [29], the number of triangles [19, 32], and the number of k-cliques [20]. A special case of BIS query (where one of the bipartition sets is a singleton) has been used for testing k-colorability of graphs [8] and high degree vertex discovery [36].

Group testing. A classic estimation problem involves efficiently approximating the number of defective items or infected individuals in a certain collection or population [12, 17, 35]. To query a population, a small group is formed, and all individuals in the group are tested in one shot. For example, in genome-wide association studies, combined pools of DNA may be tested as a group for certain variants [28]. In group testing, the result of a test often indicates only whether there is at least one infected or defective unit, or if there is none. Such a dichotomous outcome resembles the IS/BIS queries. In the graph setting, group testing suggests testing pairwise interactions between many items or individuals instead of singular events.

Computational geometry. Certain geometric applications exhibit the phenomenon that emptiness queries have more efficient algorithms than counting queries. For example, in three dimensions, for a set P of n points, half-space counting queries (i.e., what is the size of the set $|P \cap h|$, for a query half-space h) can be answered in $O(n^{2/3})$ time, after near-linear time preprocessing. However, emptiness queries (i.e., is the set $P \cap h$ empty?) can be answered in $O(\log n)$ time. Aronov and Har-Peled [6] used this to show how to answer approximate counting queries (i.e., estimating $|P \cap h|$), with polylogarithmic emptiness queries.

As another geometric example, consider the task of counting edges in disk intersection graphs using GPUs [24]. For these graphs, IS queries decide if a subset of the disks has any intersection (this can be done using sweeping in $O(n \log n)$ time [11]). Using a GPU, one could quickly draw the disks and check if the sets share a common pixel. In cases like this—when IS and BIS oracles have fast implementations—algorithms exploiting independent set queries may be useful.

Decision versus counting complexity. A generalization of IS and BIS queries previously appeared in a line of work investigating the relationship between decision and counting problems [15, 33, 34]. Stockmeyer [33, 34] showed how to estimate the number of satisfying assignments for a circuit with queries to an NP oracle. Ron and Tsur [31] observed that Stockmeyer implicitly provided an algorithm for estimating set cardinality using *subset* queries, where a subset query specifies a subset $X \subseteq \mathcal{U}$ and answers whether $|X \cap S| = 0$ or not. Subset queries are significantly more general and flexible than IS and BIS queries because S corresponds to the set of edges in the graph and X is any subset of pairs of vertices. Namely, IS and BIS queries can be interpreted as restricted subset queries. In particular, the algorithms mentioned cannot be implemented directly using IS or BIS queries.

Indeed, consider subset queries in the context of estimating the number of edges in a graph. To this end, fix $|S| = m$ (i.e., the number of edges in the graph) and $|\mathcal{U}| = \binom{n}{2}$ (the number of possible edges). Stockmeyer provided an algorithm using only $O(\log \log m \operatorname{poly}(1/\varepsilon))$ subset queries to estimate m within a factor of $(1 + \varepsilon)$ with a constant success probability. Note that for a high probability bound, which is what we focus on in this article, the algorithm would naively require $O(\log n \cdot \log \log m \operatorname{poly}(1/\varepsilon))$ queries to achieve success probability at least $1 - 1/n$. Falahatgar et al. [22] gave an improved algorithm that estimates m up to a factor of $(1 + \varepsilon)$ with probability $1 - \delta$ using $2 \log \log m + O((1/\varepsilon^2) \log(1/\delta))$ subset queries. Nearly matching lower bounds are also known for subset queries [22, 31, 33, 34]. Ron and Tsur [31] also studied a restriction of subset queries, called *interval queries*, where they assume that the universe $\mathcal{U}$ is ordered and the subsets must be intervals of elements. We view the independent set queries that we study as another natural restriction of subset queries.

Analogous to Stockmeyer's results, a recent work of Dell and Lapinskas [15] provides a framework that relates edge estimation using BIS and edge existence queries to a question in fine-grained complexity. They study the relationship between decision and counting versions of problems such as 3SUM and Orthogonal Vectors. They proved that for a bipartite graph, using $O(\varepsilon^{-2} \log^6 n)$ BIS queries, and $\varepsilon^{-4} n \operatorname{polylog}(n)$ edge existence queries, one can output a number $\widetilde{m}$ such that with probability at least $1 - 1/n^2$, we have $(1 - \varepsilon)m \leq \widetilde{m} \leq (1 + \varepsilon)m$.

Dell and Lapinskas [15] used edge estimation to obtain approximate counting algorithms for problems in fine-grained complexity. For instance, given an algorithm for 3SUM with runtime T, they obtain an algorithm that estimates the number of YES instances of 3SUM with runtime $O(T\varepsilon^{-2} \log^6 n) + \varepsilon^{-4} n \operatorname{polylog}(n)$. The relationship is simple. The decision version of 3SUM corresponds to checking if there is at least one edge in a certain bipartite graph. The counting version then corresponds to counting the edges in this graph. We note that in their application, the large number $O(n \operatorname{polylog}(n))$ of edge existence queries does not affect the dominating term in the overall time in their reduction; the larger term in the time is a product of the time to decide 3SUM and the number of BIS queries.

Table 1. Comparison of the Best-Known Algorithms Using a Variety of Queries for Estimating the Number of Edges m in a Graph with n Vertices

Query Types	Approximation	No. of Queries (up to const. factors)	Reference
Edge existence	$1+\varepsilon$	(n^2/m) poly$(\log n, 1/\varepsilon)$	Folklore (see Section 5.2)
Degree	$2+\varepsilon$	$\sqrt{n}\log n/\varepsilon$	[23]
Degree + neighbor	$1+\varepsilon$	$\sqrt{n}$ poly$(\log n, 1/\varepsilon)$	[25]
Subset	$1+\varepsilon$	poly$(\log n, 1/\varepsilon)$	[22, 34]
BIS	$1+\varepsilon$	n poly$(\log n, 1/\varepsilon)$	[15]
BIS	$1+\varepsilon$	poly$(\log n, 1/\varepsilon)$	This Work
IS	$1+\varepsilon$	$\min(\sqrt{m},\ n^2/m)$ poly$(\log n, 1/\varepsilon)$	This Work

The bounds stated are for high probability results, with error probability at most $1/n$. Constant factors are suppressed for readability.

1.2 Our Results

We describe two new algorithms. Let $G = ([\![n]\!], E)$ be a simple graph with $m = |E|$ edges.

The bipartite independence oracle. We present an algorithm that uses BIS queries and computes an estimate $\widetilde{m}$ for the number of edges in G such that $(1-\varepsilon)m \le \widetilde{m} \le (1+\varepsilon)m$. The algorithm performs $O(\varepsilon^{-4}\log^{14} n)$ BIS queries and succeeds with high probability (see Theorem 4.9 for a precise statement). Ignoring the cost of the queries, the running time is near linear (we mostly ignore running times in this article because query complexity is our main resource). Since polylog(n) BIS queries can simulate a degree query (see Section 4.4), one can obtain a $(2+\varepsilon)$-approximation of m by using Feige's algorithm [23], which uses degree queries. This gives an algorithm that uses $O(\sqrt{n}\,\mathrm{polylog}(n)/\mathrm{poly}(\varepsilon))$ BIS queries. Our new algorithm provides significantly better guarantees in terms of both the approximation and number of BIS queries.

The result is somewhat more general than stated previously. One can use the algorithm to estimate the number of edges in any induced subgraph of the original graph. Similarly, one can estimate the number of edges in the graph between any two disjoint subsets of vertices $U, V \subseteq [\![n]\!]$. In other words, the algorithm can estimate the size of $E(U,V) = \{uv \in E \mid u \in U, v \in V\}$.

Compared to the result of Dell and Lapinskas [15], our algorithm uses exponentially fewer queries since we do not spend $n\,\mathrm{polylog}(n)$ edge existence queries. Our improvement does not seem to imply anything for their applications in fine-grained complexity. We leave open the question of finding problems where a more efficient BIS algorithm would lead to new decision versus counting complexity results.

The ordinary independence oracle. We also present a second algorithm, using only IS queries to compute a $(1+\varepsilon)$-approximation. It performs $O(\varepsilon^{-4}\log^5 n + \min(n^2/m, \sqrt{m}) \cdot \varepsilon^{-2}\log^2 n)$IS queries (see Theorem 5.8). In particular, the number of IS queries is bounded by $O(\varepsilon^{-4}\log^5 n + \varepsilon^{-2}n^{2/3}\log^2 n)$. The first term in the minimum (i.e., $\approx n^2/m$) comes from a folklore algorithm for estimating set cardinality using membership queries (see Section 2.3). The second term in the minimum (i.e., $\approx \sqrt{m}$) is the number of queries used by our new algorithm.

We observe that BIS queries are surprisingly more effective for estimating the number of edges than IS queries. Shedding light on this dichotomy is one of the main contributions of this work.

Comparison with other queries. Table 1 summarizes the results for estimating the number of edges in a graph in the context of various query types. Given some of the results in Table 1 on edge estimation using other types of queries, a natural question is how well BIS and IS queries can simulate such queries. In Section 4.4, we show that $O(\varepsilon^{-2} \log n)$ BIS queries are sufficient to simulate degree queries. However, we do not know how to simulate a neighbor query (to find a specific neighbor) with few BIS queries, but a random neighbor of a vertex can be found with $O(\log n)$ BIS queries (see the work of Ben-Eliezer et al. [8]). For IS queries, it turns out that estimating the degree of a vertex v up to a constant factor requires at least $\Omega(n/\deg(v))$ IS queries (see Section 5.3).

Notation. Throughout, log and ln denote the logarithm taken in base 2 and e, respectively. For integers, u, k, let $[\![k]\!] = \{1, \ldots, k\}$ and $[\![u : k]\!] = \{u, \ldots, k\}$. The notation $x = \text{polylog}(n)$ means $x = O(\log^c n)$ for some constant $c > 0$. A collection of disjoint sets $U_1, \ldots, U_k$ such that $\bigcup_i U_i = U$, is a *partition* of the set U, into k *parts* (a part U_i might be an empty set). In particular, a (uniformly) *random partition* of U into k parts is chosen by coloring each element of U with a random number in $[\![k]\!]$ and identifying U_i with the elements colored with i.

Throughout, we use $G = ([\![n]\!], E)$ to the denote the input graph. The number of edges in G is denoted by $m = |E|$. For a set $U \subseteq [\![n]\!]$, let $E(U) = \{uv \in E \mid u, v \in U\}$ be the set of edges between vertices of U in G. For two disjoint sets $U, V \subseteq [\![n]\!]$, let $E(U, V)$ denote the set of edges between U and V: $E(U, V) = \{uv \in E \mid u \in U, v \in V\}$. Let $m(U)$ and $m(U, V)$ denote the number of edges in $E(U)$ and $E(U, V)$, respectively. We also abuse notation and let $m(H)$ be the number of edges in a subgraph H (e.g., $m(G) = m$).

High probability conventions. Through the article, the randomized algorithms presented would succeed with high probability—that is, with probability $\geq 1 - 1/n^{\Omega(1)}$. Formally, this means the probability of success is $\geq 1 - 1/n^c$, for some arbitrary constant $c > 0$. For all of these algorithms, the value of c can be increased to any arbitrary value (i.e., improving the probability of success of the algorithm) by increasing the asymptotic running time of the algorithm by a constant factor that depends only on c. For the sake of simplicity of exposition, we do not explicitly keep track of these constants (which are relatively well behaved).

1.3 Overview of the Algorithms

1.3.1 The BIS Algorithm. Our discussion of the BIS algorithm follows Figure 1, which depicts the main components of one level of our recursive algorithm. Our algorithms rely on several building blocks, as described next.

Exactly count edges. One can exactly count the edges between two subsets of vertices, with a number of queries that scales nearly linearly in the number of such edges. Specifically, a simple deterministic divide and conquer algorithm to compute $m(U, V)$ using $O(m(U, V) \log n)$BIS queries is described later in Lemma 4.1.

Sparsify. The idea is now to sparsify the graph in such a way that the number of remaining edges is a good estimate for the original number of edges (after scaling). Consider sparsifying the graph by coloring the vertices of graph and only looking at the edges going between certain pairs of color classes (in our algorithm, these pairs are a matching of the color classes). We prove that it suffices to only count the edges between these color classes, and we can ignore the edges with both endpoints inside a single color class.

For any k satisfying $1 \leq k \leq \lfloor n/2 \rfloor$, let $U_1, \ldots, U_k, V_1, \ldots, V_k$ be a uniformly random partition of $[\![n]\!]$. Then, we have

$$\mathbb{P}\left[\left|2k \sum_{i=1}^{k} m(U_i, V_i) - m\right| \geq ck\sqrt{m} \log n\right] \leq \frac{1}{n^4}, \tag{1.1}$$

where c is some constant. For the proof of this inequality, see Section 3. Specifically, if we set G_i to be the induced bipartite subgraph on U_i and V_i, then $2k \sum_i m(G_i)$ is a good estimate for $m(G)$.

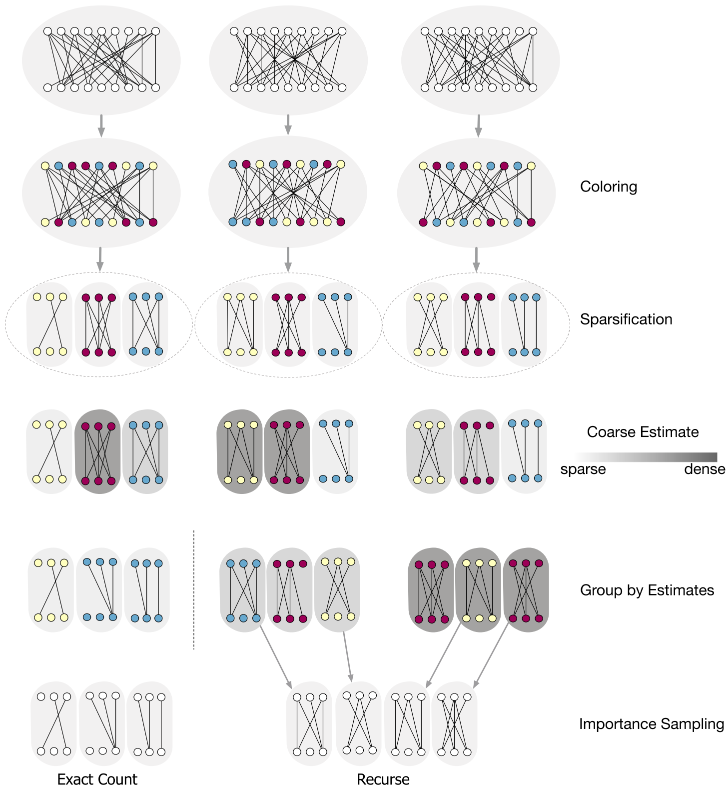

Fig. 1. A depiction of one level of the BIS algorithm. In the first step, we color the vertices and sparsify the graph by only looking at the edges between vertices of the same color. In the second step, we coarsely estimate the number of edges in each colored subgraph. Next, we group these subgraphs based on their coarse estimates, and we subsample from the groups with a relatively large number of edges. In the final step, we exactly count the edges in the sparse subgraphs, and we recurse on the dense subgraphs.

Now the graph is bipartite. The preceding sparsification method implies that we can assume without loss of generality that the graph is bipartite. Indeed, invoking the lemma with $k = 1$, we see that estimating the number of edges between the two color classes is equivalent to estimating the total number of edges, up to a factor of 2. For the rest of the discussion, we will consider colorings that respect the bipartition.

Coarse estimator. We give an algorithm that coarsely estimates the number of edges in a (bipartite) subgraph, up a $O(\log^2 n)$ factor, using only $O(\log^3 n)$ BIS queries.

The subproblems. After coloring the graph, we have reduced the problem to estimating the total number of edges in a collection of (disjoint) bipartite subgraphs. However, certain subgraphs may still have a large number of edges, and it would be too expensive to directly use the exact counting algorithm on them.

Reducing the number of subgraphs in a collection via importance sampling. Using the coarse estimates, we can form $O(\log n)$ groups of bipartite subgraphs, where each group contains subgraphs with a comparable number of edges. For the groups with only a polylogarithmic number of edges, we can exactly count edges using polylog(n) BIS queries via the exact count algorithm mentioned earlier. For the remaining groups, we subsample a polylogarithmic number of subgraphs from each group. This new estimate is a good approximation to the original quantity, with high probability. This corresponds to the technique of *importance sampling* that is used for variance reduction when estimating a sum of random variables that have comparable magnitudes.

Sparsify and reduce. We use the sparsification algorithm on each graph in our collection. This increases the number of subgraphs while reducing (by roughly a factor of k) the total number of edges in these graphs. The number of edges in the new collection is a reliable estimate for the number of edges in the old collection. We will choose k to be a constant so that every sparsification round reduces the number of edges by a constant factor.

If the number of graphs in the collection becomes too large, we reduce it in one of two ways. For the subgraphs with relatively few edges, we exactly count the number of edges using only $\text{polylog}(n)$ queries. For the dense subgraphs, we can apply the preceding importance sampling technique and retain only $\text{polylog}(n)$ subgraphs. Every basic operation in this scheme requires $\text{polylog}(n)$ BIS queries, and the number of subgraphs is $\text{polylog}(n)$. Therefore, a round can be implemented using $\text{polylog}(n)$ BIS queries. Now, since every round reduces the number of edges by a constant factor, the algorithm terminates after $O(\log n)$ rounds, resulting in the desired estimate for m using only $\text{polylog}(n)$ queries in total. Figure 1 depicts the main components of one round.

We have glossed over some details regarding the reweighting of intermediate estimates, as both the sparsfication and importance sampling steps involve subsampling and rescaling. To handle this, the algorithm will maintain a weight value for each subgraph in the collection (starting with unit weight). Then, these weights will be updated throughout the execution, and they will be used during coarse estimation. For the final estimate, the algorithm will output a weighted sum of the estimates for the remaining subgraphs in addition to the weighted version of the exactly counted subgraphs. By using these weights to properly rescale estimates and counts, the algorithm will achieve a good estimate for m with high probability.

1.3.2 The IS Algorithm. We move on to describe our second algorithm, based on IS queries. As with the BIS algorithm, the main building block for the IS algorithm is an efficient way to exactly count edges using IS queries. The exact counting algorithm works by first breaking the vertices of the graph into independent sets in a greedy fashion and then grouping these independent sets into larger independent sets using (yet again) a greedy algorithm. The resulting partition of the graph into independent sets has the property that every two sets have an edge between them, and this partition can be computed using a number of queries that is roughly m. This is beneficial, because when working on the induced subgraph on two independent sets, the IS queries can be interpreted as BIS queries. As such, edges between parts of the partition can be counted using the exact counting algorithm, modified to use IS queries. The end result is that for a given set $U \subseteq [\![n]\!]$, one can compute $m(U)$, the number of edges with both endpoints in U, using $O(m(U) \log n)$ IS queries. This algorithm is described in Section 5.1.

Now, we can sparsify the graph to reduce the overall number of IS queries. In contrast to the BIS queries, we do not know how to design a coarse estimator using only IS queries (see Section 5.3). This prohibits us from designing a similar algorithm. Instead, we estimate the number of edges in one shot by coloring the graph with a large number of colors and estimating the number of edges going between a matching of the color classes.

This is somewhat counterintuitive. An initial sparsification attempt might be to count only the edges going between a single pair of colors. If the total number of colors is $2k$, then we expect to see $m/\binom{2k}{2}$ edges between this pair. Therefore, we could set k to be large and invoke Lemma 5.3. Scaling by a factor of $\binom{2k}{2}$, we would hope to get an *unbiased* estimator for m.

Unfortunately, a star graph demonstrates that this approach does not work, due to the large variance of this estimator. If we randomly color the vertices of the star graph with $2k$ colors, then out of the $\binom{2k}{2}$ pairs of color classes, only $2k-1$ pairs have any edge going between color classes. Thus, if we only choose one pair of color classes, then with high probability one of the following two cases occurs: either (i) there is no edge crossing the color pair or (ii) the number of edges crossing the pair is $\approx m/2k$. In both cases, our estimate after scaling by a factor of $\binom{2k}{2}$ will be far from the truth.

At the other extreme, most edges will be present if we look at the edges crossing *all pairs* of color classes. Indeed, the only edges we miss have both endpoints in a color class, and this accounts for only a $1/2k$ fraction of the total number of edges. Thus, this does not achieve any substantial sparsification.

By using a matching of the color classes, we simultaneously get a reliable estimate of the number of edges and a sufficiently sparsified graph (see Lemma 3.2). Let $U_1, \ldots, U_k, V_1, \ldots, V_k$ be a random partition of the vertices into $2k$ color classes. This implies that with high probability, the estimator $2k \sum_{i=1}^{k} m(U_i, V_i)$ is in the range $m \pm O(k\sqrt{m} \log n)$. Hence, as long as we choose k to be less than $\varepsilon\sqrt{m}/\mathrm{polylog}(n)$, we approximate m up to a factor of $(1 + O(\varepsilon))$. We use geometric search to find such a k efficiently.

To get a bound on the number of IS queries, we claim that we can compute $\sum_{i=1}^{k} m(U_i, V_i)$ using Lemma 5.3, with a total of $(k + \frac{m}{k})\,\mathrm{polylog}(n)$ IS queries. The first term arises since we have to make at least one query for each of the k color pairs (even if there are no edges between them). For the second term, we pay for both (i) the edges between the color classes and (ii) the total number of edges with both endpoints within a color class (since the number of IS queries in Lemma 5.3 scales with $m(U \cup V)$). By the sparsification lemma, we know that (i) is bounded by $O(m/k)$ with high probability and we can prove an analogous statement for (ii). Hence, plugging in a value of $k \approx \varepsilon\sqrt{m}/\mathrm{polylog}(n)$, the total number of IS queries is bounded by $\sqrt{m}\,\mathrm{polylog}(n)/\varepsilon$.

1.4 Subsequent Work After Initial Publication

After the initial publication of our results [7], there has been some follow-up work [9, 10, 13, 16].

Answering one of the open questions of our earlier work [7], Chen et al. [13] provide nearly-matching upper and lower bounds on the number of IS queries for edge estimation. More precisely, they show that $O(\min(n/\sqrt{m},\ \sqrt{m}\,) \cdot \mathrm{poly}(\log(n), 1/\varepsilon))$IS queries are sufficient (the term n^2/m is the new result). They also prove that $\Omega\big(\min(n/\sqrt{m},\ \sqrt{m}\,)/\mathrm{polylog}(n))$ IS queries are necessary for a certain family of graphs.

Dell et al. [16] provide new connections between decision and approximate counting results for problems such as k-SUM, k-Orthogonal-Vectors, and k-Clique by relating the complexity to edge estimation using certain queries. In particular, their work extends the previous work of Dell and Lapinskas [15] to the case of k-hypergraphs, and they consider a generalization of BIS queries to k-partite set queries. As one of their technical results, they improve the dependence on ε in Theorem 4.9 from ε^{-4} down to ε^{-2}.

Bhattacharya et al. [9, 10] also consider the generalization of BIS queries to tripartite set queries, where they use such queries to estimate the number of triangles in a graph.

1.5 Outline

The rest of the article is organized as follows. We start in Section 2 by reviewing some necessary tools—concentration inequalities, importance sampling, and set size estimation via membership queries. In Section 3, we prove our sparsification result (Lemma 3.2).

In Section 4, we describe the algorithm for edge estimation for the BIS case. Section 4.1 describes the exact counting algorithm. In Section 4.2, we present the algorithm that uses BIS queries to coarsely estimate the number of edges between two subsets of vertices (Lemma 4.8). We combine these building blocks to construct our edge estimation algorithm using BIS queries in Section 4.3.

The case of IS queries is tackled in Section 5. In Section 5.1, we formally present the algorithms to exactly count edges between two subsets of vertices (Lemma 5.3). In Section 5.2, we present our algorithm using IS queries. In Section 5.3, we provide some discussion of why the IS case seems to be harder than the BIS case. We conclude in Section 6 and discuss open questions.

2 PRELIMINARIES

Here we present some standard tools that we need later on.

2.1 Concentration Bounds

For proofs of the following concentration bounds, see the book by Dubhashi and Panconesi [18].

Lemma 2.1 (Hoeffding's inequality). *Let $X_1, \ldots, X_r$ be independent random variables satisfying $X_i \in [a_i, b_i]$ for $i \in [r]$. Then, for $X = X_1 + \cdots + X_r$ and any $s > 0$, we have $\mathbb{P}[|X - \mathbb{E}[X]| \geq s] \leq 2\exp(-2\,s^2/\sum_{i=1}^{r}(b_i - a_i)^2)$.*

Lemma 2.2 (Chernoff-Hoeffding inequality [18, Theorem 1.1]). *Let $X_1, \ldots, X_r$ be r independent random variables with $0 \leq X_i \leq 1$, and let $X = \sum_{i=1}^{r} X_i$. For $\mu = \mathbb{E}[X]$, let ℓ and u be real numbers such that $\ell \leq \mu \leq u$. Then, we have that*

(A) For any $\Delta > 0$, we have $\mathbb{P}[X \leq \ell - \Delta] \leq \exp(-2\Delta^2/r)$ and $\mathbb{P}[X \geq u + \Delta] \leq \exp(-2\Delta^2/r)$.
(B) For any $0 \leq \delta < 1$, we have $\mathbb{P}[X \leq (1-\delta)\mu] \leq \exp(-\mu\delta^2/2)$.
(C) For any $0 \leq \delta \leq 1$, we have $\mathbb{P}[X \geq (1+\delta)\mu] \leq \exp(-\mu\delta^2/3)$.

We need a version of Azuma's inequality that takes into account a rare bad event—the following is a restatement of Theorem 8.3 from Chung and Lu [14] in a simplified form (which sufficient for our purposes).

Lemma 2.3 ([14]). *Let f be any function of r independent random variables $Y_1, \ldots, Y_r$, and let $X_i = \mathbb{E}[f(Y_1, \ldots, Y_r) \mid Y_1, \ldots, Y_i]$, for $i \in [\![r]\!]$, and $X_0 = \mathbb{E}[f(Y_1, \ldots, Y_r)]$. Say that a sequence $Y_1, \ldots, Y_r$ is* bad *if there exists an index i such that $|X_i - X_{i-1}| > c_i$, where $c_1, \ldots, c_r$ are some non-negative numbers. Let $\mathcal{B}$ be the event that a bad sequence happened, and let $S = \sum_{i=1}^{r} c_i^2$. We have that $[|X_r - X_0| \geq \lambda] \leq 2\exp(-\lambda^2/2S) + \mathbb{P}[\mathcal{B}]$.*

2.2 Importance Sampling

Importance sampling is a technique for estimating a sum of terms. Assume that for each term in the summation, we can cheaply and quickly get an initial, coarse estimate of its value. Furthermore, assume that better estimates are possible but expensive. Importance sampling shows how to sample terms in the summation, then acquire a better estimate *only for the sampled terms*, to get a good estimate for the full summation. In particular, the number of samples is bounded independently of the original number of terms, depending instead on the coarseness of the initial estimates, the probability of success, and the quality of the final output estimate.

Lemma 2.4 (Importance Sampling). *Let $U = \{u_1, \ldots, u_r\}$ be a set of numbers, all contained in the interval $[\alpha/b, \alpha b]$, for $\alpha > 0$ and $b \geq 1$. Let $\gamma, \varepsilon > 0$ be parameters. Consider the sum $\Gamma = \sum_{i=1}^{r} u_i$. For an arbitrary $t \geq \frac{b^4}{2\varepsilon^2}(1 + \ln\frac{1}{\gamma})$, and $i = 1, \ldots, t$, let X_i be a random sample chosen uniformly (and independently) from the set U (i.e., let j_i be uniformly and randomly picked from $[\![r]\!]$, and let $X_i = u_{j_i}$). Then, the estimate $Y = (r/t)\sum_{i=1}^{t} X_i$ for the value of Γ satisfies $\mathbb{P}[|Y - \Gamma| \geq \varepsilon\Gamma] \leq \gamma$.*

Proof. Observe that $r(\alpha/b) \leq \Gamma \leq r\alpha b$, and

$$\mu = \mathbb{E}[Y] = \mathbb{E}\left[(r/t)\sum_{i=1}^{t} X_i\right] = (r/t)\sum_{i=1}^{t} \mathbb{E}[X_i] = \frac{r}{t} \cdot t \cdot \frac{\Gamma}{r} = \Gamma.$$

Furthermore, we have $Z = (t/r)Y = \sum_{i=1}^{t} X_i$, $\mathbb{E}[Z] = (t/r)\Gamma$, and for the length, Δ_i, of the interval containing X_i, we have $\Delta_i^2 = (\alpha b - \alpha/b)^2 \leq \alpha^2 b^2$.

Using $r\alpha/b \leq \Gamma$, by Lemma 2.1, we have

$$\begin{aligned}
\mathbb{P}[|Y - \Gamma| \geq \varepsilon\Gamma] &= \mathbb{P}\left[\left|\frac{t}{r}Y - \frac{t}{r}\Gamma\right| \geq \frac{t\varepsilon\Gamma}{r}\right] \leq \mathbb{P}\left[\left|\sum_{i=1}^{t} X_i - \frac{t}{r}\Gamma\right| \geq \frac{t\varepsilon r\alpha/b}{r}\right] \\
&= \mathbb{P}\left[\left|\sum_{i=1}^{t} X_i - \mathbb{E}[Z]\right| \geq \frac{t\varepsilon\alpha}{b}\right] \leq 2\exp\left(-\frac{2\,(t\varepsilon\alpha/b)^2}{\sum_{i=1}^{t}\Delta_i^2}\right) \leq 2\exp\left(-\frac{2\,(t\varepsilon\alpha/b)^2}{t\alpha^2 b^2}\right) \\
&= 2\exp\left(-\frac{2t\varepsilon^2}{b^4}\right) \leq \gamma. \qquad \square
\end{aligned}$$

The preceding lemma enables us to reduce a summation with many numbers into a much shorter summation (while introducing some error, naturally). The list/summation reduction algorithm we need is described next.

Lemma 2.5 (Summation reduction). *Let $(\mathcal{H}_1, w_1, e_1), \ldots, (\mathcal{H}_r, w_r, e_r)$ be given, where $\mathcal{H}_i$'s are some structures, and w_i and e_i are numbers, for $i = 1, \ldots, r$. Every structure $\mathcal{H}_i$ has an associated unknown cost $\overline{\mathrm{w}}(\mathcal{H}_i) \geq 0$. The quantity of interest that we would like to compute/approximate is*

$$\Gamma = \sum_i w_i \cdot \overline{\mathrm{w}}(\mathcal{H}_i).$$

To this end, we have parameters $\xi > 0$, γ, b, and M such that:

(i) $\forall i \quad w_i, e_i \geq 1$,
(ii) $\forall i \quad e_i/b \leq \overline{\mathrm{w}}(\mathcal{H}_i) \leq e_i b$, *and*
(iii) $\Gamma \leq M$.

Then, one can compute a new (hopefully shorter) sequence of triples $(\mathcal{H}'_1, w'_1, e'_1), \ldots, (\mathcal{H}'_t, w'_t, e'_t)$ (the new sequence is a subsequence of the original sequence with reweighting). The new sequence complies with the preceding conditions, and furthermore, the estimate

$$Y = \sum_{i=1}^{t} w'_i \overline{\mathrm{w}}\left(\mathcal{H}'_i\right)$$

is a multiplicative $(1 \pm \xi)$-approximation to Γ, with probability $\geq 1 - \gamma$. The running time of the algorithm is $O(r)$, and size of the output sequence is $t = O(b^4 \xi^{-2}(\log\log M + \log \gamma^{-1}) \log M)$.

Proof. We break the interval $[1, M]$ into $\log M$ intervals in the natural way, where the j^{th} interval is $J_j = [2^{j-1}, 2^j)$, for $j = 1, \ldots, h = \lceil \log M \rceil$, except if M is a power of 2, in which case the last interval is closed and also includes $2^h = M$. Input triples are sorted into h groups $U_1, \ldots, U_h$, where an input triple $(\mathcal{H}, w, e)$ is in U_j, if $ew \in J_j$. This mapping can be done in $O(r)$ time.

Let $\alpha = O(b^4 \xi^{-2}[1 + \ln(h/\gamma)])$. For $j = 1, \ldots, h$, if $|U_j| \leq \alpha$, then set $R_j = U_j$, and otherwise compute a sample R_j from U_j of size α. We associate weight $W_j = |U_j|/|R_j|$ with R_j. If a triple $(\mathcal{H}, w, e) \in U_j$, then we have that $w \cdot \overline{\mathrm{w}}(\mathcal{H}) \in [2^{j-1}/b, 2^j b]$.

For all $j \in [\![h]\!]$, let $\Gamma_j = \sum_{(\mathcal{H},w,e)\in U_j} w \cdot \overline{\mathrm{w}}(\mathcal{H})$ be the total weight of structures in the j^{th} group. By Lemma 2.4, we have, with probability $\geq 1 - \gamma/h$, that

$$Y_j = \left(W_j \sum\nolimits_{(\mathcal{H},w,e)\in R_j} \overline{\mathrm{w}}\,(\mathcal{H}) \cdot w\right) \in \left[(1-\xi)\Gamma_j, (1+\xi)\Gamma_j\right].$$

Summing these inequalities over all $j \in [\![h]\!]$, implies that Y is the desired approximation with probability $\geq 1 - \gamma$.

Specifically, the output sequence is constructed as follows. For all $j \in [\![h]\!]$, and for every triple $(\mathcal{H}, w, e) \in R_j$, we add $(\mathcal{H}, w \cdot W_j, e)$ to the output sequence. Clearly, the output sequence has $t = h\alpha = O\left(b^4\xi^{-2}(\log\log M + \log\gamma^{-1})\log M\right)$ elements. □

Remark. (A) The algorithm of Lemma 2.5 does not use the entities $\mathcal{H}_i$ directly at all. In particular, the $\mathcal{H}_i'$s are just copies of some original structures. The only thing that the preceding lemma uses is the estimates $e_1, \ldots, e_r$ and the weights $w_1, \ldots, w_r$.

(B) The sampling size used in Lemma 2.5 can probably be improved by a polylog factors by sampling directly from all $\log M$ classes simultaneously.

Remark 2.6. We are going to use Lemma 2.5, with $\xi = O(\varepsilon/\log n)$, $\gamma = 1/n^{O(1)}$, $b = O(\log n)$, and $M = n^2$. As such, the size of the output list is

$$L_{\text{len}} = O(\log^4 n \cdot \varepsilon^{-2}\log^2 n \cdot (\log\log n + \log n)\log n) = O(\varepsilon^{-2}\log^8 n).$$

2.3 Estimating Subset Size via Membership Oracle Queries

We present here a standard tool for estimating the size of a subset via membership oracle queries. This is well known, but we provide the details for the sake of completeness.

Lemma 2.7. *Consider two (finite) sets $B \subseteq U$, where $n = |U|$. Let $\varepsilon \in (0,1)$ and $\gamma \in (0, 1/2)$ be parameters. Let $g > 0$ be a user-provided guess for the size of $|B|$. Consider a random sample R, taken with replacement from U, of size $r = \lceil c_5\varepsilon^{-2}(n/g)\log\gamma^{-1}\rceil$, where c_5 is sufficiently large. Next, consider the estimate $Y = (n/r)|R \cap B|$ to $|B|$. Then, we have the following:*

(A) If $Y < g/2$, then $|B| < g$,
(B) If $Y \geq g/2$, then $(1-\varepsilon)Y \leq |B| \leq (1+\varepsilon)Y$.

Both of the preceding statements hold with probability $\geq 1 - \gamma$.

Proof. (A) The bad scenario here is that $|B| \geq g$, but $Y < g/2$. Let $X_i = 1 \iff$ the i^{th} sample element is in B. We have that $Y = (n/r)X$, where $X = \sum_{i=1}^r X_i$. By assumption, we have

$$\mu = \mathbb{E}\,[X] = \frac{r\,|B|}{n} \geq \frac{rg}{n} \geq c_5\frac{\log\gamma^{-1}}{\varepsilon^2}. \tag{2.1}$$

As such, by Chernoff's inequality (Lemma 2.2(B)), we have that $\mathbb{P}[Y < g/2] = \mathbb{P}[X < rg/(2n)] = \mathbb{P}[X < (1-1/2)\mathbb{E}[X]] \leq \exp(-\mu/8) \leq \gamma^{c_5/(8\varepsilon^2\ln 2)}$, and this is $\leq \gamma$ for c_5 a sufficiently large constant.

(B) We have two cases to consider. For the first case, suppose that $|B| < g/4$. In this case, if $X = \sum_{i=1}^r X_i$ is the random variable as described part (A), then each X_i is an indicator variable with probability $p = |B|/n < g/(4n)$ and $\mathbb{P}[Y \geq g/2] = \mathbb{P}[X \geq rg/(2n)] \leq \mathbb{P}[X' \geq rg/(2n)]$, where X' is the sum of r independent Bernoulli trials with success probability $g/(4n)$. Now, $\mathbb{E}[X'] = \frac{rg}{4n} \geq c_5\frac{\log\gamma^{-1}}{4\varepsilon^2}$, so

$$\mathbb{P}[Y \geq g/2] \leq \mathbb{P}[X' \geq rg/(2n)] = \mathbb{P}[X' \geq (1+1)\mathbb{E}\,[X']] \leq \exp\left(-\mathbb{E}\,[X']\,/3\right) \leq \gamma^{c_5/(12\varepsilon^2\ln 2)}$$

by Chernoff's inequality (Lemma 2.2(C)) and again this is $\leq \gamma$ for c_5 a sufficiently large constant.

For the second case, suppose that $|B| \geq g/4$. Then, $\mathbb{E}[X] \geq \mathbb{E}[X'] \geq c_5 \frac{\log \gamma^{-1}}{4\varepsilon^2}$, and since Y is a fixed multiple of X, by Chernoff's inequality (Lemma 2.2(B)), we have

$$\mathbb{P}[Y < (1-\varepsilon)\mathbb{E}[Y]] = \mathbb{P}[X < (1-\varepsilon)\mathbb{E}\,[X]] \leq \exp(-\mathbb{E}[X]\varepsilon^2/2) \leq \gamma^{c_5/(8\ln 2)},$$

which is $\leq \gamma/2$ for $c_5 \geq 16 \ln 2$. Similarly, by Chernoff's inequality (Lemma 2.2(C)),

$$\mathbb{P}[Y > (1+\varepsilon)\mathbb{E}[Y]] = \mathbb{P}[X > (1+\varepsilon)\mathbb{E}\,[X]] \leq \exp(-\mathbb{E}\,[X]\,\varepsilon^2/3) \leq \gamma^{c_5/(12\ln 2)},$$

which is $\leq \gamma/2$ for $c_5 \geq 24 \ln 2$, as $\gamma \leq 1/2$. Adding these two failure probabilities together gives a bound of at most γ as required. □

Lemma 2.8. *Consider two sets $B \subseteq U$, where $n = |U|$. Let $\xi, \gamma \in (0,1)$ be parameters such that $\gamma < 1/\log n$. Assume that one is given an access to a membership oracle that, given an element $x \in U$, returns whether or not $x \in B$. Then, one can compute an estimate s such that $(1-\xi)|B| \leq s \leq (1+\xi)|B|$, and computing this estimate requires $O((n/|B|)\xi^{-2}\log\gamma^{-1})$ oracle queries. The returned estimate is correct with probability $\geq 1-\gamma$.*

Proof. Let $g_i = n/2^{i+2}$. For $i = 1, \ldots, \log n$, use the algorithm of Lemma 2.7 with $\varepsilon = 0.5$, with the probability of failure being $\gamma/(8\log n)$, and let Y_i be the returned estimate. The algorithm stops this loop as soon as $Y_i \geq 4g_i$. Let I be the value of i when the loop stopped. The algorithm now calls Lemma 2.7 again with g_I and $\varepsilon = \xi$, and returns the value of Y, as the desired estimate.

Overall, for $T = 1 + \lceil \log n \rceil$, the preceding makes T calls to the subroutine of Lemma 2.7, and the probability that any of them to fail is $T\gamma/(8\log n) < \gamma$. Assume that all invocations of Lemma 2.7 were successful. In particular, Lemma 2.7 guarantees that if $Y > 4g_I \geq g_I/2$, then the estimate returned is $(1 \pm \varepsilon)$-approximation to the desired quantity.

Computing Y_i requires $r_i = O((n/g_i)\log(\log n/\gamma)) = O(2^i \log \gamma)$ oracle membership queries. As such, the number of membership queries performed by the algorithm overall is

$$\sum_i r_i + O((n/g_I)\varepsilon^{-2}\log(\log n/\gamma)) = O((n/\,|B|)\varepsilon^{-2}\log\gamma). \qquad □$$

2.3.1 Estimating Subset Size via Emptiness Oracle Queries. Consider the variant where we are given a set $X \subseteq U$. Given a query set $Q \subseteq U$, we have an *emptiness oracle* that tells us whether $Q \cap X$ is empty. Using an emptiness oracle, one can get a $(1 \pm \varepsilon)$-approximate the size of X using relatively few queries. The following result is implied by the work of Aronov and Har-Peled [6, Theorem 5.6] and Falahatgar et al. [22]—the latter result has better bounds if the failure probability is not required to be polynomially small.

Lemma 2.9 ([6, 22]). *Consider a set $X \subseteq U$, where $n = |U|$. Let $\varepsilon \in (0,1)$ be a parameter. Assume that one is given an access to an emptiness oracle that, given a query set $Q \subseteq U$, returns whether or not $X \cap Q \neq \emptyset$. Then, one can compute an estimate s such that $(1-\varepsilon)|X| \leq s \leq (1+\varepsilon)|X|$, using $O(\varepsilon^{-2}\log n)$ emptiness queries. The returned estimate is correct with probability $\geq 1 - 1/n^{\Omega(1)}$.*

We sketch the basic idea of the algorithm used in the preceding lemma. For a guess g of the size of X, consider a random sample Q where every element of U is picked with probability $1/g$. The probability that Q avoids X is $\alpha(g) = (1-1/g)^{|X|}$. The function $\alpha(g)$ is (i) monotonically increasing, (ii) close to zero when $g \ll |X|$, (iii) $\approx 1/e$ for $g = |X|$, and (iv) close to 1 if $g \gg |X|$. One can estimate the value $\alpha(g)$ by repeated random sampling and checking if the random sample intersects X using emptiness queries. Given such an estimate, one can then perform an approximate binary search for the value of g such that $\alpha(g) = 1/e$, which corresponds to $g = |X|$. See the work of Arnov and Har-Peled [6] and Falahatgar et al. [22] for further details.

3 EDGE SPARSIFICATION BY RANDOM COLORING

In this section, we present and prove that coloring vertices and counting only edges between specific color classes provides a reliable estimate for the number of edges in the graph. This is distinct from standard graph sparsification algorithms, which usually sparsify the edges of the graph directly (usually by sampling edges).

We need the following technical lemma.

Lemma 3.1. *Let C be a set of r elements, colored randomly by k colors—specifically, for every element $x \in C$, one chooses randomly (independently and uniformly) a color for it from the set $[\![k]\!]$. For $i \in [\![k]\!]$, let n_i be the number of elements of C with color i. Let n be a positive integer and $c > 1$ be an arbitrary constant. Then,*

(A) For any color $i \in [\![k]\!]$, we have $\mathbb{P}[|n_i - r/k| > \sqrt{(cr/2)\ln n}\,] \leq 2/n^c$.
(B) For any two distinct colors $i, j \in [\![k]\!]$, we have $\mathbb{P}[|n_i - n_j| > \sqrt{2cr \ln n}\,] \leq 4/n^c$.
(C) For any two distinct colors $i, j \in [\![k]\!]$, we have $\mathbb{E}[|n_i - n_j|] \leq \sqrt{2r/k}$.

Proof. (A) For $\ell \in [\![r]\!]$, let X_ℓ be the indicator variable that is 1 with probability $1/k$ and 0 otherwise. For $X = \sum_{\ell=1}^{r} X_\ell$, notice that n_i is distributed identically to X and that $\mathbb{E}[X] = \mathbb{E}[n_i] = r/k$. Using Chernoff's inequality (Lemma 2.2(A)), we have

$$\mathbb{P}\left[|X - \mathbb{E}\,[X]| > \sqrt{(cr/2)\ln n}\right] \leq 2\exp\left(-\tfrac{2}{r}\cdot\tfrac{cr}{2}\ln n\right) \leq 2/n^c.$$

(B) Observe that $|n_i - n_j| \leq |n_i - r/k| + |r/k - n_j|$, and the claim follows from (A).

(C) For $t = 1, \ldots, r$, let $X_t = 1$ if the t^{th} element of C is colored by color i, and let $X_t = -1$ if this element is colored by color j. Otherwise, set $X_t = 0$. Clearly, the desired quantity is $\mu = \mathbb{E}[|X|]$, where $X = \sum_{t=1}^{r} X_t$. We have that $\mathbb{P}[X_t = 1] = \mathbb{P}[X_t = -1] = 1/k$, $\mathbb{E}[X_t] = 0$ and that $\mathbb{E}[X_t^2] = 2/k$. As such, by the independence of the X_is, we have $\mathbb{E}[X^2] = 2\sum_{i<j}\mathbb{E}[X_iX_j] + \sum_{i=1}^{r}\mathbb{E}[X_i^2] = r\frac{2}{k}$. Finally, we have $\mathbb{V}[|X|] = \mathbb{E}[|X|^2] - \mu^2 \geq 0$. As such, $\mu = \mathbb{E}[|X|] \leq \sqrt{\mathbb{E}[|X|^2]} = \sqrt{\mathbb{E}[X^2]} \leq \sqrt{2r/k}$. □

Lemma 3.2. *(A) There exists an absolute constant ς such that the following holds. For every n, let $G = ([\![n]\!], E)$ be a graph with m edges. For any $1 \leq k \leq \lfloor n/2 \rfloor$, let $U_1, \ldots, U_{2k}$ be a uniformly random partition of $[\![n]\!]$. Then,*

$$\mathbb{P}\left[\left|\frac{m}{2k} - \sum_{i=1}^{k} m\left(U_i, U_{k+i}\right)\right| \geq \varsigma\sqrt{m}\log n\right] \leq \frac{1}{n^4}, \quad \textit{and} \quad \mathbb{P}\left[\left|\frac{m}{2k} - \sum_{i=1}^{2k} m\left(U_i\right)\right| \geq \varsigma\sqrt{m}\log n\right] \leq \frac{1}{n^4}.$$

(B) There exists an absolute constant ς such that the following holds. Similarly, for every n, disjoint sets $U, V \subseteq [\![n]\!]$, and k such that $2 \leq k \leq \max\{|U|, |V|\}$, let $U_1, \ldots, U_k, V_1, \ldots, V_k$ be uniformly random partitions of U and V, respectively. Then,

$$\mathbb{P}\left[\left|m\left(U, V\right) - k\sum_{i=1}^{k} m\left(U_i, V_i\right)\right| \geq \varsigma k\sqrt{m\left(U, V\right)}\log n\right] \leq 1/n^4.$$

Proof. (A) Consider the random process that colors vertex t, at time $t \in [\![n]\!]$, with a uniformly random color $Y_t \in [\![2k]\!]$. The colors correspond to the partition of $[\![n]\!]$ into classes $U_1, \ldots, U_{2k}$. Define

$$f(Y_1, \ldots, Y_n) = \sum_{i=1}^{k} m\left(U_i, U_{k+i}\right).$$

The probability of a specific edge uv to be counted by f is $1/(2k)$. Indeed, fix the color of u, and observe that there is only one choice of the color of v, such that uv would be counted. As such, $\mathbb{E}[f] = m/(2k)$ and $0 \leq f(Y_1, \ldots, Y_n) \leq m$.

Consider the Doob martingale $X_0, X_1, \ldots, X_n$, where $X_t = \mathbb{E}\left[f(Y_{[\![n]\!]}) \mid Y_{[\![t]\!]}\right]$, where $Y_{[\![t]\!]} \equiv Y_1, \ldots, Y_t$. We are interested in bounding the quantity $|X_t - X_{t-1}|$. To this end, fix the value of $Y_{[\![t-1]\!]}$, and let

$$g(\alpha) = \mathbb{E}[f(Y_{[\![n]\!]}) \mid Y_{[\![t-1]\!]} \cap (Y_t = \alpha)].$$

We have that $X_{t-1} = \mathbb{E}\left[f(Y_{[\![n]\!]}) \mid Y_{[\![t-1]\!]}\right] = \sum_{\alpha=1}^{2k} g(\alpha)/2k$. Namely, the value of X_{t-1} is an average of the values in $\mathcal{G} = \{g(1), g(2), \ldots, g(2k)\}$. Clearly, $X_t \in \mathcal{G}$. As such, we have that $|X_t - X_{t-1}| \leq \max_{i,j} |g(i) - g(j)|$.

Let $N(t)$ be the set of neighbors of t in the graph and $\deg(t) = |N(t)|$ be the degree of t. Let $N_{<t} = N(t) \cap [\![t-1]\!]$ and $N_{>t} = N(t) \cap [\![t+1 : n]\!]$ be the before/after set of neighbors of t, respectively. Let $C^i_{<t}$ (respectively, $C^i_{>t}$) be the number of neighbors of t in $N_{<t}$ (respectively, $N_{>t}$) colored with color i. For a color $i \in [\![2k]\!]$, let $\pi(i) = 1 + ((k + i - 1) \bmod 2k)$ be its matching color.

Fix two distinct colors $i, j \in [\![2k]\!]$, and let

$$\Delta_t = |g(i) - g(j)| = \left| C^{\pi(i)}_{<t} + \mathbb{E}\left[C^{\pi(i)}_{>t} \mid Y_t = i\right] - C^{\pi(j)}_{<t} - \mathbb{E}\left[C^{\pi(j)}_{>t} \mid Y_t = j\right] \right|.$$

To see why the preceding is true, observe that any edge involving two vertices in $[\![t-1]\!]$ has the same contribution to $g(i)$ and $g(j)$. Similarly, an edge with a vertex in $[\![t-1]\!]$, and a vertex in $[\![t+1 : n]\!]$, has the same contribution to both terms. The same argument holds for an edge involving vertices with indices strictly larger than t. As such, only the edges adjacent to t have a different contribution, which is as stated. Rearranging, we have by Lemma 3.1 with $C = N(t)$ and $r = \deg(t)$, with probability at least $1 - \beta$ for $\beta = 4/n^c$ for any constant $c > 1$, that

$$\Delta_t = \left| C^{\pi(i)}_{<t} + \mathbb{E}\left[C^{\pi(i)}_{>t}\right] - C^{\pi(j)}_{<t} - \mathbb{E}\left[C^{\pi(j)}_{>t}\right] \right| \leq \left| C^{\pi(i)}_{<t} - C^{\pi(j)}_{<t} \right| + \mathbb{E}\left[\left| C^{\pi(i)}_{>t} - C^{\pi(j)}_{>t} \right|\right] \leq c_t$$

for $c_t = \sqrt{2c\,\deg(t) \ln n} + \sqrt{\deg(t)/k} \leq 3c\sqrt{\deg(t) \ln n}$. Let $\mathcal{B}$ be the event that any Δ_t (for any choice of i, j or t) exceeds c_t, and observe that we can choose a constant $c > 1$ such that $\mathbb{P}[\mathcal{B}] \leq (2k)^2 n\beta \leq 1/n^{10}$.

Let $S = \sum_{t=1}^n c_t^2 = \sum_{t=1}^n 9c^2 \deg(t) \ln n = O(m \ln n)$, and $s = \varsigma\sqrt{m} \ln n$. Applying Lemma 2.3 to $X_{[\![n]\!]}$, we have

$$\mathbb{P}[|f - m/2k| > s] = \mathbb{P}[|X_n - X_0| > s] \leq 2\exp(-s^2/2S) + \mathbb{P}[\mathcal{B}] \leq 2/n^{10} + 1/n^{10} \leq 1/n^4,$$

for ς sufficiently large.

For the second claim in part (A), a nearly-identical argument works, with $f(Y_1, \ldots, Y_n) = \sum_{i=1}^{2k} m(U_i)$. Part (B) also follows by a similar argument as part (A)—for example, identifying V_i with U_{k+i} throughout. □

Remark 3.3. Given an induced bipartite graph $G = (U, V, E)$ with m edges, coloring it with k colors, and taking the bipartite subgraphs of the resulting matching of the coloring, as done in Lemma 3.2, results in k new disjoint bipartite (induced) subgraphs, $G_i = (U_i, V_i, E_i)$, for $i = 1, \ldots, k$, with total number of edges $\Gamma = \sum_{i=1}^k m(G_i)$. Furthermore, we have that $k \cdot \Gamma$ is a $(1 \pm \xi)$-approximation to $m(G)$, where $\xi = (\varsigma k \sqrt{m} \log n)/m$, with high probability. For our purposes, we need

$$\xi \leq \frac{\varepsilon}{8 \log n} \iff \frac{(\varsigma k \sqrt{m} \log n)}{m} \leq \frac{\varepsilon}{8 \log n} \iff \frac{8(\varsigma k \log^2 n)}{\varepsilon} \leq \sqrt{m} \iff m = \Omega(k^2 \varepsilon^{-2} \log^4 n).$$

Setting $k = 4$, the preceding implies that one can apply the refinement algorithm of Lemma 3.2 if $m = \Omega(\varepsilon^{-2} \log^4 n)$. With high probability, the number of edges in the new k subgraphs (i.e., Γ), scaled by k, is a good estimate (i.e., within a $1 \pm \varepsilon/(8 \log n)$ factor) for the number of edges in the original graph, and furthermore, the number of edges in the new subgraphs is small (formally, $\mathbb{E}[\Gamma] \leq m/4$, and with high probability $\Gamma \leq m/2$).

4 EDGE ESTIMATION USING BIS QUERIES

Here we show how to get exact and approximate count for the number of edges in a graph using BIS queries (see also [5, 30]).

4.1 Exactly Counting Edges Using BIS Queries

Lemma 4.1. *Given two disjoint sets $U, V \subseteq [\![n]\!]$, one can (deterministically) compute $E(U, V)$, and thus $m(U, V) = |E(U, V)|$, using $O(1 + m(U, V) \log n)$ BIS queries. Alternatively, given a parameter (informally, a query budget) $t = \Omega(\log n)$, one can decide if the given graph has $\leq t/\log n$ edges (or more) using $O(t)$ BIS queries.*

Proof. We use a recursive divide-and-conquer approach, which intuitively builds a quadtree over the pair (U, V). Specifically, consider the incidence matrix M of size $|U| \times |V|$, where a column corresponds to an element of V and a row to an element of U. An entry in the matrix is equal to 1 if there is an edge between the corresponding nodes in the original graph, and it is zero otherwise. The task at hand as such is to count the number of ones in the matrix. A BIS query then corresponds to deciding if an induced submatrix is all zero. We now conceptually build a tree (i.e., a quadtree), by partitioning the matrix into four submatrices of the same dimensions (in the natural way), and recursively build a quadtree for each submatrix. Intuitively, the algorithm counts the 1s in the matrix, by tracking each of the 1s to their corresponding leaf node in the quadtree.

To this end, the algorithm first issues the query BIS (U, V). If the return value is false, then there are no edges between U and V, and the algorithm sets $m(U, V)$ to zero, and returns. If $|U| = |V| = 1$, then this also determines if $m(U, V)$ is 0 or 1 in this case, and the algorithm returns. The remaining case is that $m(U, V) \neq 0$, and the algorithm recurses on the four children of (U, V), which will correspond to the pairs $(U_1, V_1), (U_1, V_2), (U_2, V_1)$, and (U_2, V_2), where U_1, U_2 and V_1, V_2 are equipartitions of U and V, respectively. We are using here the identity

$$m\left(U, V\right) = m\left(U_1, V_1\right) + m\left(U_1, V_2\right) + m\left(U_2, V_1\right) + m\left(U_2, V_2\right).$$

If $m(U, V) = 0$ holds, then the number of queries is exactly equal to 1, and the lemma is true in this case. For the rest of the proof, we assume that $m(U, V) \geq 1$. To bound the number of queries, imagine building the whole quadtree for the adjacency matrix of $U \times V$ with entries for $E(U, V)$. Let X be the set of 1 entries in this matrix, and let $k = |X|$ (i.e., X corresponds to set of leaves that are labeled 1 in the quadtree). The height of the quadtree is $h = O(\max\{\log |U|, \log |V|\})$. Let X_1 be the set of nodes in the quadtree that are either in X or are ancestors of nodes of X. It is not hard to verify that $|X_1| = O(k + k \log(|U||V|)) = O(k \log n)$. Finally, let X_2 be the set of nodes in the quadtree that are either in X_1 or their parent is in X_1. Clearly, the algorithm visits only the nodes of X_2 in the recursion, thus implying the desired bound.

As for the budgeted version, run the algorithm until it has accumulated $T = O(t/\log n)$ edges in the working set, where $T > t$. If this never happens, then the number of edges of the graph is at most T, as desired, and the preceding analysis applies. Otherwise, the algorithm stops, and applying the same argument as before, we get that the number of BIS queries is bounded by $O(T \log n) = O(t)$. □

Remark. The number of BIS queries made by the algorithm of Lemma 4.1 is at least $\max\{m(U, V), 1\}$, since every edge with one endpoint in U and the other in V is identified (on its own, explicitly) by such a query.

We note that we can use the above algorithm to exactly identify the edges of an *arbitrary* graph using BIS queries with a cost of $O(\log n)$ overhead per edge (see also [5, 30]). However, we will not need to do so in the sequel.

Lemma 4.2. *Given a vertex $v \in [\![n]\!]$, one can compute all of the edges adjacent to v in G using $O(1 + \deg(v) \log n)$ queries.*

Proof. Let $V = \{v\}$, and $U = [\![n]\!] \setminus V$, and observe that $\deg(v) = m(V, U)$. The algorithm of Lemma 4.1 can now be used, observing that it can be modified to report all edges found, thus implying the result. □

Lemma 4.3. *Given a vertex $v \in [\![n]\!]$, and a graph $G = ([\![n]\!], E)$, let C be the connected component of v in G. The set of edges in C (i.e., $E(C)$) can be computed using $O(1 + m(C) \log n)$ BIS queries, where $m(C)$ is the number of edges in C.*

Proof. Do a breadth-first search in G starting from v. Whenever reaching a vertex for the first time, compute its adjacent edges using Lemma 4.2. Clearly, the breadth-first search visits all vertices in C and therefore computes all edges in this connected component. The bound on the number of queries readily follows by observing that $\sum_{v \in V(C)} d(v) \log n$ is $O(m(C) \log n)$. □

Lemma 4.4. *For a graph $G = ([\![n]\!], E)$, one can deterministically compute $m(E)$ exactly, using at most $O(\log n + |E| \log n)$ BIS queries. Alternatively, given a parameter (informally, a query budget) $t = \Omega(\log n)$, one can decide whether the given graph has at most $t/\log n$ edges, or more than this number, using $O(t)$ BIS queries.*

Proof. If we are given a set Z that contains at least one vertex in each connected component of G, then the result follows readily by applying the algorithm of Lemma 4.3 to the vertices of Z in order, deleting the vertices of each connected component as it is being discovered from Z. The total number of BIS queries is $O(|E| \log n)$, as the edges of each connected component are discovered in different invocations of Lemma 4.3.

We remain with the task of computing Z. Let $U_0 = [\![n]\!]$. For $i = 1, \ldots, T = \lceil \log_2 n \rceil$, let A_i be the elements of U_{i-1} whose i^{th} bit in their binary representation is 1. Let $B_i = U_{i-1} \setminus A_i$. Compute all edges in $E_i = E(A_i, B_i)$ using the algorithm of Lemma 4.1. This requires $O(1 + |E_i| \log n)$ queries. Let $V_i = V(E_i)$. We add V_i to Z and set $U_i = U_{i-1} \setminus V_i$.

Observe that every edge e in G has an index i such that its two vertices differ in the i^{th} bit. Note that either one of the endpoints of e was already added to Z before the i^{th} iteration, or it would be discovered and its endpoints added to Z in the i^{th} iteration. As such, the set Z is computed correctly. Since $E_1, \ldots, E_T$ are disjoint sets, it follows that computing Z requires $O(T + \sum_i |E_i|) = O(\log n + |E| \log n)$ BIS queries.

For the budgeted version, we run the algorithm until $\tau = \Omega(t)$ BIS queries have been performed. If this does not happen, then the graph has at most τ edges, and they were reported by the algorithm. Otherwise, we know that the graph must have at least $\tau/\log n$ edges, as desired. □

4.2 The Coarse Estimator Algorithm

Let $G = G([\![n]\!], E)$ be a graph and let $U, V \subseteq [\![n]\!]$ be disjoint subsets of the vertices. The task at hand is to estimate $m(U, V)$, using polylog BIS queries.

For a subset $S \subseteq [\![n]\!]$, define $N(S)$ to be the union of the neighbors of all vertices in S. For a vertex v, let $\deg_S(v)$ denote the number of neighbors of v that lie in S. For $i \in [\![\log n]\!]$, define the set of vertices in U with degree between 2^i and 2^{i+1} as

ALGORITHM 4.1: CheckEstimate$(U, V, \widetilde{e})$

Input: $((U, V), \widetilde{e})$ where $U, V \subseteq [\![n]\!]$ are disjoint and $\widetilde{e}$ is a (rough) guess for the value of $m(U, V)$

for $i = 0, 1, \ldots, \log n$ **do**
  Sample $U' \subseteq U$ by choosing each vertex in U with probability $\min(2^i/\widetilde{e}, 1)$.
  Sample $V' \subseteq V$ by choosing each vertex of V with probability $1/2^i$.
  if $m(U', V') \neq 0$ **then**
    Output **accept**;
Output **reject**.

$$U_i = \{u \mid u \in U,\ 2^i < \deg_V(u) \leq 2^{i+1}\},$$

and let U_0 denote the vertices in U with $\deg_V(v) \leq 2$.

Claim 4.5. *There exists an $\alpha \in \{0, 1, \ldots, \log n\}$ such that*

$$m(U_\alpha, V) \geq \frac{m(U, V)}{\log n + 1} \qquad \textit{and} \qquad |U_\alpha| \geq \frac{m(U, V)}{2^{\alpha+1}(\log n + 1)}.$$

Proof. Since $\sum_{i=0}^{\log n} m(U_i, V) = m(U, V)$, the first inequality is stating that there is a term as large as the average. As for the second inequality, observe that for every i, we have $|U_i|2^i \leq m(U_i, V) \leq |U_i|2^{i+1}$. Hence, using the first inequality, $|U_\alpha| \geq \frac{m(U_\alpha, V)}{2^{\alpha+1}} \geq \frac{m(U,V)}{2^\alpha} \cdot \frac{1}{2(\log n+1)}$. □

Suppose that we have an estimate $\widetilde{e}$ for the number of edges between U and V in the graph. Consider the test CheckEstimate, depicted in Algorithm 4.1, for checking if the estimate $\widetilde{e}$ is correct up to polylogarithmic factors using a logarithmic number of BIS queries.

Claim 4.6. *Let $n \geq 16$. If $m(U, V) > 0$, then*

(A) if $\widetilde{e} \geq 4m(U, V)(\log n + 1)$, then CheckEstimate$(U, V, \widetilde{e})$ *accepts with probability at most* $1/4$.
(B) if $\widetilde{e} \leq \frac{m(U,V)}{4 \log n}$, then CheckEstimate$(U, V, \widetilde{e})$ *accepts with probability at least* $1/2$.

Proof. (A) For any value of the loop variable i, the probability that a fixed edge is present in the induced subgraph on U' and V' is $\min(2^i/\widetilde{e}, 1) \cdot (1/2^i) \leq 1/\widetilde{e}$. Thus, $\mathbb{E}[m(U', V')] \leq m(U, V)/\widetilde{e} \leq \frac{1}{4(\log n+1)}$. For a fixed iteration i, by Markov's inequality, we have

$$\mathbb{P}[m(U', V') \neq 0] = \mathbb{P}[m(U', V') \geq 1] \leq \mathbb{E}[m(U', V')] \leq \frac{1}{4(\log n + 1)}.$$

By a union bound over the loop variable values, the probability that the test accepts is at most $1/4$.

(B) It is enough to show that the probability is at least $1/2$ when the loop variable attains the value α given by Claim 4.5. In this case, we have that $|U_\alpha| \geq \frac{m(U,V)}{2^{\alpha+1}(\log n+1)}$, and thus

$$\begin{aligned}\mathbb{P}[U' \cap U_\alpha = \varnothing] &= \left(1 - \frac{2^\alpha}{\widetilde{e}}\right)^{|U_\alpha|} \leq \exp\left(-\frac{2^\alpha}{\widetilde{e}} \cdot |U_\alpha|\right) \leq \exp\left(-\frac{2^\alpha}{m(U, V)/(4\log n)} \cdot \frac{m(U, V)}{2^{\alpha+1}(\log n + 1)}\right)\\ &\leq \exp\left(-\frac{4\log n}{2(\log n + 1)}\right) \leq \frac{1}{e^{1.6}},\end{aligned}$$

since $n \geq 16$. Furthermore, since $\deg_V(u) \geq 2^\alpha$ for all $u \in U_\alpha$, it follows that when $U' \cap U_\alpha \neq \varnothing$, then $|N(U' \cap U_\alpha)| \geq 2^\alpha$. Thus, we can bound

$$\mathbb{P}[V' \cap N(U' \cap U_\alpha) = \varnothing \mid U' \cap U_\alpha \neq \varnothing] \leq \left(1 - \frac{1}{2^\alpha}\right)^{2^\alpha} \leq \frac{1}{e}.$$

ALGORITHM 4.2: CoarseEstimator(U, V)

Input: (U, V) where $U, V \subseteq [\![n]\!]$ are disjoint
Output: An estimate $\widetilde{e}$ for the number of edges $m(U, V)$ computed using BIS queries

```
if m(U,V) = 0 then Output 0;
for j = 2 log n, ..., 0 do
    Run t := 128 log n independent trials of CheckEstimate(U, V, 2^j).
    if at least 3t/8 of them output accept then
        Output 2^j;
```

From the preceding, we get

$$\mathbb{P}\left[m\left(U', V'\right) \neq 0\right] = \mathbb{P}[U' \cap U_\alpha \neq \varnothing] \cdot \mathbb{P}[V' \cap N(U' \cap U_\alpha) \neq \varnothing \mid U' \cap U_\alpha \neq \varnothing]$$
$$\geq \left(1 - \frac{1}{e^{1.6}}\right)\left(1 - \frac{1}{e}\right) \geq \frac{1}{2}. \qquad \square$$

Armed with the preceding test, we can easily estimate the number of edges up to a $O(\log n)$ factor by doing a search, where we start with $\widetilde{e} = n^2$ and halve the number of edges each iteration. The algorithm is depicted in Algorithm 4.2.

Claim 4.7. *For $n \geq 16$, CoarseEstimator(U, V) outputs $\widetilde{e} \leq n^2$ satisfying*

$$\frac{m\left(U, V\right)}{8 \log n} \leq \widetilde{e} \leq 8m\left(U, V\right) \log n,$$

with probability at least $1 - 4n^{-4} \log n$. The number of BIS queries made is $c_{\mathrm{ce}} \log^3 n$ for a constant c_{ce}.

Proof. For any fixed value of the loop variable j such that $2^j \geq 4m(U, V)(\log n + 1)$, the expected number of accepts is at most $t/4$ using Claim 4.6(A), where $t = 128 \log n$. The probability that we see at least $3t/8 = t/4 + t/8$ accepts is bounded by $\exp(-2(t/8)^2/t) = \exp(-t/32) \leq n^{-4}$ by Chernoff's inequality (Lemma 2.2(A)). Taking the union over all values of j, the probability that the algorithm returns 2^j, when $2^j \geq 4m(U, V)(\log n + 1)$, is at most $2n^{-4} \log n$.

However, when $2^j \leq m(U, V)/(4 \log n)$, the expected number of accepts is at least $t/2$, by Claim 4.6(B), and so the probability that we see at least $3t/8 = t/2 - t/8$ accepts is at least $1 - \exp(-2t/8^2) \geq 1 - n^{-4}$ by Chernoff's inequality (Lemma 2.2(A)). Hence, conditioned on the event that the algorithm has not already returned a bigger value of j, the probability that we accept for the unique j that satisfies $m(U, V)/8 \leq 2^j \log n < m(U, V)/4$ is at least $1 - n^{-4}$.

Overall, by a union bound, the probability that the estimator outputs an estimate $\widetilde{e}$ that does not satisfy $(8 \log n)^{-1} \leq \widetilde{e}/m(U, V) \leq 8 \log n$ is at most $4n^{-4} \log n$. The number of BIS queries is bounded by $O(\log^3 n)$, since for each value of j there are $t = 128 \log n$ trials of CheckEstimate, each of which makes $\log n + 1$ queries to the BIS oracle. $\square$

Summarizing the preceding, we get the following result

Lemma 4.8. *For $n \geq 16$, and arbitrary $U, V \subseteq [\![n]\!]$ that are disjoint, the randomized algorithm CoarseEstimator(U, V) makes at most $c_{\mathrm{ce}} \log^3 n$ BIS queries (for a constant c_{ce}) and outputs $\widetilde{e} \leq n^2$ such that with probability at least $1 - 4n^{-4} \log n$, we have $(8 \log n)^{-1} \leq \widetilde{e}/m(U, V) \leq 8 \log n$.*

4.3 The Overall BIS Approximation Algorithm

Given a graph $G = ([\![n]\!], E)$, next we describe an algorithm that makes $\mathrm{polylog}(n)/\varepsilon^4$ BIS queries to estimate the number of edges in the graph within a factor of $(1 \pm \varepsilon)$.

The algorithm for estimating the number of edges in the graph is going to maintain a data-structure $\mathcal{D}$ containing the following:

(A) An accumulator φ—this is a counter that maintains an estimate of the number of edges already handled.
(B) A list of triples $(U_1, V_1, \mathrm{w}_1), \ldots, (U_u, V_u, \mathrm{w}_u)$, where $U_i, V_i \subseteq [\![n]\!]$ and $\mathrm{w}_1 > 1$ is a non-negative weight.

The *estimate* based on $\mathcal{D}$ of the number of edges in the original graph $G = ([\![n]\!], E)$ is

$$m(\mathcal{D}) = \varphi + \sum_i \mathrm{w}_i \cdot m(U_i, V_i).$$

The number of *active* edges in $\mathcal{D}$ is $m_{\mathrm{active}}(\mathcal{D}) = \sum_i m(U_i, V_i)$.

4.3.1 Cleanup, Refine, and Reduce. The algorithm uses three subroutines: Cleanup, Refine, and Reduce, described next.

(A) Cleanup: The cleanup stage removes from $\mathcal{D}$ all induced subgraphs that have few edges by explicitly counting their number of edges. Let

$$L_{\mathrm{small}} = \Theta(\varepsilon^{-2} \log^4 n), \tag{4.1}$$

as specified by Remark 3.3. Given the data structure $\mathcal{D}$, the algorithm scans the list of triples $(U, V, \mathrm{w}) \in \mathcal{D}$. For each triple (U, V, w), using the algorithm of Lemma 4.1, it decides if $m(U, V) \leq 2L_{\mathrm{small}}$. If so, the value of $m(U, V)$ was just computed, and it adds $\mathrm{w} \cdot m(U, V)$ to φ. Finally, it removes this triple from $\mathcal{D}$.

If $\mathcal{D}$ has no triples in it, then the algorithm returns φ as the desired approximation.

(B) Refine: We are given the data structure $\mathcal{D}$, where the graph associated with every triple has at least L_{small} edges. The algorithm replaces every triple $(U, V, \mathrm{w}) \in \mathcal{D}$ by the four induced subgraphs resulting from 4-coloring the graph $G(U, V)$, as described by Lemma 3.2(B) (see also Remark 3.3). Specifically, the coloring results in the pairs (U_i, V_i), for $i = 1, 2, 3, 4$. The triple (U, V, w) is replaced in $\mathcal{D}$ by the triples $\{(U_1, V_1, 4\mathrm{w}), \ldots, (U_4, V_4, 4\mathrm{w})\}$. This increases the number of triples in $\mathcal{D}$ by a factor of 4.

(C) Reduce: If $\mathcal{D}$ has more than $2L_{\mathrm{len}}$ triples, where $L_{\mathrm{len}} = O(\varepsilon^{-2} \log^8 n)$ as specified by Remark 2.6, then the algorithm reduces the number of triples.

To this end, the algorithm first computes for each triple $(U, V, \mathrm{w}) \in \mathcal{D}$, a coarse estimate $\tilde{e}$ of the number of edges in $m(U, V)$, such that $m(U, V)/(8 \log n) \leq \tilde{e} \leq m(U, V) 8 \log n$, by using Algorithm 4.2. This requires $O(\log^3 n)$BIS queries per triple.

Next, the algorithm uses the summation reduction algorithm of Lemma 2.5 applied to the list of triples in $\mathcal{D}$, with $\xi = \varepsilon/(8 \log n)$. This reduces the number of triples in $\mathcal{D}$ to be at most L_{len} while introducing a multiplicative error of $(1 \pm \xi)$.

4.3.2 The Algorithm in Detail. The algorithm input is the graph $G = ([\![n]\!], E)$ and a parameter $\varepsilon > 0$. Let $\mathcal{R} = O(\varepsilon^{-4} \log^{14} n)$ be some parameter. The algorithm works as follows:

(A) Check if G has at most $O(\mathcal{R}/\log^2 n)$ edges, using the algorithm of Lemma 4.4, which requires $O(\mathcal{R})$ BIS queries. If so, the algorithm returns the exact number of edges in G and stops.
(B) Compute a random 2-coloring of the vertices of the graph, creating two sets $U \cup V = [\![n]\!]$ (see Lemma 3.2 (A)). We now create a data structure as described earlier, with $\mathcal{D} = [\varphi, (U, V, 2)]$, where φ is initialized to value 0.
(C) As long as $\mathcal{D}$ contains some triple, the algorithm does the following:

(a) Performs Cleanup on $\mathcal{D}$, as described in Section 4.3.1(A).
(b) Performs Refine on $\mathcal{D}$, as described in Section 4.3.1(B).
(c) Performs Reduce on $\mathcal{D}$, as described in Section 4.3.1(C).

(D) The algorithm now returns the value φ as the desired approximation.

4.3.3 Analysis. Number of iterations. Initially, the number of active edges is at most m. Every time Refine is executed, this number reduces by a factor of 2 with high probability using Lemma 3.2(B) (in expectation, the reduction is by a factor of 4). As such, after $\lceil \log m \rceil \leq \lceil \log \binom{n}{2} \rceil \leq 2 \log n$ iterations there are no active edges, and then the algorithm terminates.

Number of BIS queries. Clearly, because Reduce is used on $\mathcal{D}$ in each iteration, the algorithm maintains the invariant that the number of triples in $\mathcal{D}$ is at most $O(L_{\mathrm{len}})$, where $L_{\mathrm{len}} = O(\varepsilon^{-2} \log^8 n)$ as specified by Remark 2.6.

The procedure Cleanup applies the algorithm of Lemma 4.1 to decide whether a triple in the list has at least $2L_{\mathrm{small}}$ edges associated with it, or fewer edges, where $L_{\mathrm{small}} = \Theta(\varepsilon^{-2} \log^4 n)$ (see Equation (4.1) and Remark 3.3). This takes $O(L_{\mathrm{small}} \log n)$ BIS queries. Overall, the Cleanup step performs $O(L_{\mathrm{small}} L_{\mathrm{len}} \log n)$ queries in each iteration. The procedure Refine does not perform any BIS queries. The procedure Reduce performs $O(L_{\mathrm{len}} \log^3 n)$ BIS queries in the estimation stage.

As such, overall, the algorithm performs $O(L_{\mathrm{small}} L_{\mathrm{len}} \log n) = O(\varepsilon^{-2} \log^4 n \cdot \varepsilon^{-2} \log^8 n \cdot \log n) = O(\varepsilon^{-4} \log^{13} n)$ BIS queries per iteration. There are $O(\log n)$ iterations, and as such, the overall number of BIS queries is $\mathcal{R} = O(\varepsilon^{-4} \log^{14} n)$, which also bounds the number of BIS queries in the first step of the algorithm.

Approximation error. The initial 2-coloring of the graph, in (B), introduces a $(1 \pm \varepsilon_0)$-multiplicative error, by Lemma 3.2, where

$$\varepsilon_0 = O(\sqrt{1/m} \log n) \ll \xi = \frac{\varepsilon}{8 \log n}.$$

Inside each iteration, Cleanup introduces no error. By the choice of parameters, Refine introduces a multiplicative error that is at most $1 \pm \xi$ (see Remark 3.3). Similarly, Reduce introduces a multiplicative error bounded by $1 \pm \xi$ (see Remark 2.6). As such, the multiplicative approximation of the algorithms lies in the interval

$$[(1 - \varepsilon_0)(1 - \xi)^{2 \log n}, (1 + \varepsilon_0)(1 - \xi)^{2 \log n}] \subseteq [1 - \varepsilon, 1 + \varepsilon],$$

since $(1 - \varepsilon/(8 \log n))^{1+2 \log n} \geq 1 - \varepsilon$ and $(1 + \varepsilon/(8 \log n))^{1+2 \log n} \leq 1 + \varepsilon$ as easy calculations show.

Probability of success. Throughout this analysis, c will be a constant that can be chosen to be arbitrarily large. The algorithm may fail due to the following reasons: (i) the random 2-coloring in step (B) gives an estimate that is far from its expectation (this probability is at most $1/n^c$ using Lemma 3.2(A)), (ii) the Refine step fails (the probability for the failure of each iteration is at most $1/n^c$ using Lemma 3.2(B)), (iii) the coarse estimate in Reduce step fails (the probability for the failure of each iteration is at most $1/n^c$ using Claim 4.7), and last (iv) the summation reduction in the Reduce step fails (the probability for the failure of each iteration is at most $1/n^c$ using Lemma 2.5). Overall, every step performed by the algorithm had probability at most $1/n^c$ to fail. The algorithm performs $O(\mathrm{polylog}(n))$ steps with high probability, which implies that the algorithm succeeds with probability at least $1 - 1/n^{O(1)}$.

4.3.4 The Overall BIS Result.

Theorem 4.9. *Let $G = ([\![n]\!], E)$ be an undirected graph. For a parameter $\varepsilon \in (0, 1)$, one can compute an estimate $\widetilde{m}$ for the number of edges in G such that $(1 - \varepsilon)m(G) \leq \widetilde{m} \leq (1 + \varepsilon)m(G)$, where*

$m(G)$ is the number of edges of G. The algorithm performs $O(\varepsilon^{-4}\log^{14} n)$ BIS queries and succeeds with probability $\geq 1 - 1/n^{O(1)}$.

4.4 Degree Estimation Using BIS Queries

We provide an auxiliary degree estimation result, connecting BIS queries to local queries (e.g., [23, 25]).

Lemma 4.10. *Given a graph $G = ([\![n]\!], E)$, a parameter $\varepsilon \in (0,1)$, and a vertex $v \in [\![n]\!]$, one can $(1 \pm \varepsilon)$-approximate $\deg(v)$ in G using $O(\varepsilon^{-2}\log n)$ BIS queries. The approximation is correct with high probability.*

Proof. Let $N(v) = \{i \mid vi \in E\}$ be the set of neighbors of v, and let $E_v = \{vi \mid vi \in E\}$ be the corresponding set of edges. We have $\deg(v) = |N(v)| = |E_v|$. Given a set of edges $E_Q \subseteq E_v = \{vi \mid i \in [\![n]\!]\}$, the corresponding set of vertices is $Q = \{i \mid vi \in E_Q\}$. In particular, $Q \cap N(v) \neq \emptyset \iff E_Q \cap E_v \neq \emptyset$. Deciding if $E_Q \cap E_v \neq \emptyset$ is equivalent to deciding if any of the edges adjacent to v is in E_Q, and this is answered by the BIS query for $(\{v\}, Q)$. Namely, the BIS oracle can function as an emptiness oracle for $N(v) \subseteq [\![n]\!]$. Now, using the algorithm of Lemma 2.9, we can $(1 \pm \varepsilon)$-approximation $|N(v)|$ using $O(\varepsilon^{-2}\log n)$ queries, as claimed. □

5 EDGE ESTIMATION USING IS QUERIES

This section describes and analyzes our IS query algorithm (Theorem 5.8). At the end, we also discuss limitations of IS queries, suggesting that IS queries may indeed be weaker than BIS queries.

5.1 Exactly Counting Edges Using IS Queries

We start with an exact edge counting algorithm for IS querie (see also [5, 30]). At a high-level, we use Lemma 4.1 after efficiently computing a suitable decomposition of our graph.

Lemma 5.1. *Given disjoint sets of vertices $U, V \subseteq [\![n]\!]$ such that both U and V are independent sets, one can compute the number of edges $m(U \cup V)$ using $O(m(U \cup V)\log n)$ IS queries, assuming $m(U, V) > 0$.*

Proof. Since U and V are disjoint and independent, we have that $m(U \cup V) = m(U, V)$. Furthermore, for any $U' \subseteq U$ and $V' \subseteq V$, the query $\mathsf{BIS}(U', V')$ is equivalent to the query $\mathsf{IS}(U' \cup V')$. As such, we can use the algorithm of Lemma 4.1, using the IS queries as a replacement for the BIS queries, yielding the result. □

The next step is to break the set of interest U into independent sets.

Lemma 5.2. *Given a set $U \subseteq [\![n]\!]$, one can decompose it into disjoint independent sets $V_1, V_2, \ldots, V_t$ such that*

(a) $U = \bigcup_{i=1}^{t} V_i$, and
(b) for any $i, j \in [\![t]\!]$, with $i < j$, we have $m(V_i, V_j) > 0$.

Furthermore, computing this decomposition uses only $O(1 + m(U)\log n)$ IS queries.

Proof. Order the elements of $U = \{u_1, \ldots, u_k\}$ arbitrarily. The idea is to break U into independent sets, where each independent set is an interval $I_j = \{u_{i_j}, u_{i_j+1}, \ldots, u_{i_{j+1}-1}\}$. This can be done in a greedy fashion from left to right, discovering the index where an interval stops being an independent set. Assume inductively that one has computed the first j such independent intervals $I_1, \ldots, I_j$, and also assume that $I_j \cup \{u_{i_{j+1}}\}$ is not an independent set. Next, using binary search on the range $\{i_{j+1} + 1, \ldots, n\}$, find the maximal β such that $\{u_{i_{j+1}}, \ldots, u_\beta\}$ is independent.

Set $i_{j+2} = \beta + 1$, $I_{j+1} = \{u_{i_{j+1}}, \ldots, u_{i_{j+2}-1}\}$, and continue to the next iteration. Note that each binary search for computing an interval uses $O(\log n)$ IS queries.

For any j, we have $m(I_j, I_{j+1}) \geq 1$, which implies that the number of computed intervals τ satisfies $\tau \leq m(U) + 1$. As such, this stage uses $O((1 + m(U)) \log n)$IS queries. This results in a decomposition of U into τ independent sets $I_1, \ldots, I_\tau$.

In the second stage, starting with the computed collection of independent sets, the algorithm greedily tries to merge sets. In each step, the algorithm takes two independent sets B, W in the current collection (for which it might be possible that their merged set is independent), and the algorithm uses an IS query to check whether $B \cup W$ is an independent set. If it is, then the algorithm merges the two sets into one independent set (replacing B, W by the set $B \cup W$ in the current collection of sets). Otherwise, the algorithm marks the two sets B and W as being incompatible with each other. Note that if B, W are incompatible, then for any $B' \supseteq B$ and $W' \supseteq W$, the sets B' and W' are also incompatible. Namely, incompatibility is preserved under merger of independent sets, and the algorithm can keep track of the incompatible pairs under merger (importantly, a merger cannot decrease the number of incompatible pairs). The algorithm stops when all current sets are pairwise incompatible.

Each merge of two independent sets can be charged to the number of independent sets decreasing by one. Each pair of sets that is discovered to be incompatible can be charged to the edge witnessing that the merged set is *not* independent. Since every edge is only charged once by this process, it follows that the total number of IS queries performed by the second stage of the algorithm is at most $\tau + m(U) \leq 2m(U) + 1$.

The resulting collection of independent sets has the desired properties, completing the proof. □

LEMMA 5.3. *Given $U \subseteq [\![n]\!]$, one can deterministically compute $E(U)$, using $O(1 + m(U) \log n)$ IS queries. Alternatively, given a budget $t > 0$ and set $U \subseteq [\![n]\!]$, one can decide if $m(U) > t$ using $O(t \log n)$ IS queries.*

PROOF. Using the algorithm of Lemma 5.2, compute the decomposition of U into independent sets $V_1, \ldots, V_t$. By construction, for any $i < j$, we have that $m(V_i, V_j) \geq 1$, as some vertex of V_i is connected to some vertex in V_j. As such, going over all $1 \leq i < j \leq t$, compute the set of edges $E(V_i, V_j)$ using the algorithm of Lemma 5.1. This requires $O(m(V_i, V_j) \log n)$ IS queries. As such, the total number of IS queries used by this algorithm is $O(m(U) \log n + \sum_{i<j} m(V_i, V_j) \log n) = O(m(U) \log n)$.

The budgeted version follows by running the algorithm until $\lceil c \log n \rceil$ IS queries have been performed, for c a sufficiently large constant. If this happens, then the number of edges in the graph is larger than t (as otherwise, the preceding implies that the algorithm would have already terminated), and the algorithm stops and outputs this fact. □

5.2 Algorithms for Edge Estimation Using IS Queries

Our IS algorithm has two main subroutines. We first describe and analyze these, then we combine them for the overall algorithm, which is presented in Theorem 5.8.

5.2.1 Growing Search. The following is an immediate consequence of Lemma 5.3.

LEMMA 5.4. *Let*

$$L_{\text{base}} = \lceil c_1 \varepsilon^{-4} \log^4 n \rceil,$$

where c_1 is some sufficiently large constant. Given a set U, one can decide if $m(U) \leq L_{\text{base}}$, and if so get the exact value of $m(U)$, using $O(\varepsilon^{-4} \log^5 n)$ IS queries.

LEMMA 5.5. *Given parameters $t, \varepsilon \in (0, 1]$, and a set $U \subseteq [\![n]\!]$, such that $m(U) \geq \max(L_{\text{base}}, t^2)$, an algorithm can decide if $m(U) > 2t^2$, or alternatively return a $(1 \pm \varepsilon)$-approximation to $m(U)$ if $t^2 \leq m(U) \leq 2t^2$. The algorithm uses $O(\varepsilon^{-1} t \log^2 n)$ IS queries and succeeds with probability $1 - 1/n^{O(1)}$.*

PROOF. We color the vertices in U randomly using $k = \lceil t\varepsilon/(\varsigma \log n) \rceil$ colors for a constant ς to be specified shortly, and let $U_1, \ldots, U_k$ be the resulting partition. By Lemma 3.2, we have for the estimate $\Gamma = \sum_{i=1}^{k} m(U_i)$ that

$$|m(U) - k \cdot \Gamma| \leq \varsigma k \sqrt{m(U)} \log n,$$

and this holds with probability $\geq 1 - n^{-c_3}$, where c_3 is an arbitrarily large constant, and ς is a constant that depends only on c_3. For this to be a $(1 \pm \varepsilon)$-approximation, we need that

$$\frac{\varsigma k \sqrt{m(U)} \log n}{m(U)} \leq \varepsilon.$$

This in turn is equivalent to

$$m(U) \geq \left(\frac{\varsigma k \log n}{\varepsilon} \right)^2 = t^2,$$

which holds because of the assumption that $m(U) \geq \max\{L_{\text{base}}, t^2\}$ in the statement.

To proceed, the algorithm starts computing the terms in the summation defining Γ, using the algorithm of Lemma 5.3. If at any point in time the summation exceeds $M = 8(t^2/k) = O(\varepsilon^{-1} t \log n)$, then the algorithm stops and reports that $m(U) > 2t^2$. Otherwise, the algorithm returns the computed count $k \cdot \Gamma$ as the desired approximation. In both cases, we are correct with high probability by Lemma 3.2.

We now bound the number of IS queries. If the algorithm computed Γ by determining exact edge counts for $m(U_i)$ for all $i \in [\![k]\!]$, then the number of queries would be $\sum_{i=1}^{k} O(1 + m(U_i) \log n)$. However, the choice of stopping early if the number of queries exceeds $M = O(\varepsilon^{-1} t \log n)$ implies that the total number of queries is bounded by $O(k + M \log n) = O(\varepsilon^{-1} t \log^2 n)$. □

LEMMA 5.6. *Given $\varepsilon \in (0, 1]$, and a set $U \subseteq [\![n]\!]$, one can compute a $(1 \pm \varepsilon)$-approximation for $m(U)$. The algorithm uses at most $O(\varepsilon^{-4} \log^5 n + \varepsilon^{-1} \sqrt{m(U)} \log^2 n)$ IS queries and succeeds with probability $1 - 1/n^{O(1)}$.*

PROOF. The algorithm starts by checking if the number of edges in $m(U)$ is at most $L_{\text{base}} = O(\varepsilon^{-4} \log^4 n)$ using the algorithm of Lemma 5.4. Otherwise, in the i^{th} iteration, the algorithm sets $t_i = \sqrt{2} t_{i-1}$, where $t_0 = \sqrt{L_{\text{base}}}$, and invokes the algorithm of Lemma 5.5 for t_i as the threshold parameter. If the algorithm succeeds in approximating the right size, we are done. Otherwise, we continue to the next iteration. Taking a union bound over the iterations, we have that the algorithm stops with high probability before $t_\alpha > 4\sqrt{m(U)}$. Let α be the minimum value for which this holds. The number of IS queries performed by the algorithm is $O(\sum_{i=1}^{\alpha} t_i \varepsilon^{-1} \log^2 n) = O(\varepsilon^{-1} \sqrt{m(U)} \log^2 n)$, since this is a geometric sum. □

5.2.2 Shrinking Search. We are given a graph $G = ([\![n]\!], E)$ and a set $U \subseteq [\![n]\!]$. The task at hand is to approximate $m(U)$. Let $\mathcal{N} = |U|$.

Given an oracle that can answer IS queries, we can decide if a specific edge uv exists in the set $E(U)$ by performing an IS query on $\{u, v\}$. We can treat such IS queries as membership oracle queries in the set E of edges in the graph, where the ground set is the set of all possible edges $Z = \binom{U}{2} = \{ij \mid i < j \text{ and } i, j \in U\}$, where $|Z| = \mathcal{N}(\mathcal{N} - 1)/2$. Invoking the algorithm of Lemma 2.8 in this case, with $\gamma = 1/n^{O(1)}$, implies a $(1 \pm \varepsilon)$-approximation to $m(U)$ using $O((\mathcal{N}^2/m(U))\varepsilon^{-2} \log n)$ IS queries. For our purposes, however, we need a budgeted version of this.

Lemma 5.7. *Given parameters $t > 0$, $\xi \in (0, 1]$, and a set $U \subseteq [\![n]\!]$, with $\mathcal{N} = |U|$, an algorithm can either (a) return $m(U) \leq \mathcal{N}^2/(2t)$ or (b) return $(1 \pm \xi)$-approximation to $m(U)$. The algorithm uses $O(t \log n)$ IS queries in case (a) and $O(t\xi^{-2} \log n)$ in case (b). The returned result is correct with high probability.*

Proof. The idea is to use the sampling as done in Lemma 2.7, with $g = \mathcal{N}^2/(16t)$ and $\varepsilon = 1/2$ on the sets of edges $E(U) \subseteq \binom{U}{2}$. The sample R used is of size $O((\mathcal{N}^2/g) \log n) = O(t \log n)$, and we check for each one of the sampled edges if it is in the graph by using an IS query. If the returned estimate is at most $g/2$, then the algorithm returns that it is in case (a).

Otherwise, we invoke the algorithm of Lemma 2.7 again, with $\varepsilon = \xi$, to get the desired approximation, which is case (b). □

5.2.3 The Overall IS Search Algorithm.

Theorem 5.8. *We are given a graph $G = ([\![n]\!], E)$, with access to the edges of the graph via an IS oracle. Let $m = |E|$ be the number of edges in G. The quantity m can be $(1 \pm \varepsilon)$-approximated by an algorithm that uses*

$$O\left(\varepsilon^{-4} \log^5 n + \min(\sqrt{m}, n^2/m)\varepsilon^{-2} \log^2 n\right)$$

IS queries, and it succeeds with probability $\geq 1 - 1/n^{O(1)}$.

Proof. Let $t_0 = \lceil c_1\varepsilon^{-4} \log^4 n\rceil$, for some constant c_1. Using the algorithm of Lemma 5.4, we can decide if $m \leq t_0$, and if so return (the just computed) m.

The algorithm now loops for $i = 1, 2, 3, \ldots$, n. In the i^{th} iteration, it does the following:

(A) If $i = 1$, then let $t_1 = \sqrt{t_0}$, and otherwise set $t_i = 2t_{i-1}$.
(B) Using the algorithm of Lemma 5.5, decide if $m \leq 2t_i^2$, and if so it returns the desired $(1 \pm \varepsilon)$-approximation to m. This uses $O(t_i\varepsilon^{-1} \log^2 n)$ IS queries.
(C) Using the algorithm of Lemma 5.7, decide if $m \leq n^2/(2t_i)$, and if so continue to the next iteration. This uses $O(t_i \log n)$ IS queries.
    Otherwise, the algorithm of Lemma 5.7 returned the desired $(1 \pm \varepsilon)$-approximation, using $O(t_i\varepsilon^{-2} \log n)$ IS queries.

Combining the two bounds on the IS queries, we get that the i^{th} iteration used $O(t_i\varepsilon^{-2} \log^2 n)$ IS queries.

The algorithm stopped in the i^{th} iteration if $t_i \geq \sqrt{m/2}$ or $t_i \geq n^2/m$. In particular, for the stopping iteration I, we have $t_I = O(\min(\sqrt{m}, n^2/m))$. As such, the total number of IS queries in all iterations except the last one is bounded by $O(\sum_{i=1}^{I} t_i\varepsilon^{-2} \log^2 n) = O(t_I\varepsilon^{-2} \log^2 n)$. The stopping iteration uses $O(t_I\varepsilon^{-2} \log^2 n)$ IS queries. Each bound holds with high probability, and a union bound implies the same for the final result. □

Corollary 5.9. *For a graph $G = ([\![n]\!], E)$, with an access to G via IS queries, and a parameter $\varepsilon > 0$, one can $(1 \pm \varepsilon)$-approximate m using $O(\varepsilon^{-4} \log^5 n + n^{2/3}\varepsilon^{-2} \log^2 n)$ IS queries.*

Proof. Follows readily as $\min(\sqrt{m}, n^2/m) \leq n^{2/3}$, for any value of m between 0 and n^2. □

5.3 Limitations of IS Queries

In this section, we discuss several ways in which IS queries seem more restricted than BIS queries.

Simulating degree queries with IS queries. A degree query can be simulated by $O(\log n)$ BIS queries (see Lemma 4.10). In contrast, here we provide a graph instance where $\Omega(n/\deg(v))$ IS queries are needed to simulate a degree query. In particular, we show that IS queries may be no better than edge existence queries for the task of degree estimation. Since it is easy to see that

$\Omega(n/\deg(v))$ edge existence queries are needed to estimate $\deg(v)$, this lower bound also applies to IS queries.

For the lower bound instance, consider a graph that is a clique along with a separate vertex v whose neighbors are a subset of the clique. We claim that IS queries involving v are essentially equivalent to edge existence queries. Any edge existence query can be simulated by an IS query. However, any IS query on the union of v and at least two clique vertices will always detect a clique edge. Thus, the only informative IS queries involve exactly two vertices.

Coarse estimator with IS queries. It is natural to wonder if it is possible to replace the coarse estimator (Lemma 4.8) with an analogous algorithm that makes $\text{polylog}(n)$ IS queries. This would immediately imply an algorithm making $\text{polylog}(n)/\varepsilon^4$ IS queries that estimates the number of edges. We do not know if this is possible, but one barrier is a graph consisting of a clique U on $O(\sqrt{m})$ vertices along with a set V of $n - O(\sqrt{m})$ isolated vertices. We claim that for this graph, the algorithm CoarseEstimator(U, V) from Section 4.2, using IS queries instead of BIS queries, will output an estimate $\widetilde{m}$ that differs from m by a factor of $\Theta(n^{1/3})$. Consider the execution of CheckEstimate$(U, V, \widetilde{e})$ from Algorithm 4.1. A natural way to simulate this with IS queries would be to use an IS query on $U' \cup V'$ instead of a BIS query on (U', V'). Assume for the sake of argument that $m = n^{4/3}$ and $|U| = \sqrt{m} = n^{2/3}$. Consider when the estimate $\widetilde{e}$ satisfies $\widetilde{e} = cn^{5/3}$ for a small constant c. In the CheckEstimate execution, there will be a value $i = \Theta(\log n)$ such that with constant probability, $U' \subseteq U$ will contain at least two vertices and $V' \subseteq V$ will contain at least one vertex. In this case, $m(U' \cup V') \neq 0$ even though $m(U', V') = 0$. Thus, using IS queries will lead to incorrectly accepting on such a sample, and this would lead to the CoarseEstimator outputting the estimate $\widetilde{e} = \Theta(n^{5/3})$ even though the true number of edges is $m = n^{4/3}$.

6 CONCLUSION

In this article, we explored the task of using either BIS or IS queries to estimate the number of edges in a graph. We presented randomized algorithms giving a $(1 + \varepsilon)$-approximation using $\text{polylog}(n)/\varepsilon^4$ BIS queries and $\min\{n^2/(\varepsilon^2 m), \sqrt{m}/\varepsilon\}\, \text{polylog}(n)$IS queries. Our algorithms estimate the number of edges by first sparsifying the original graph and then exactly counting edges spanning certain bipartite subgraphs. Next we describe a few open directions for future research.

6.1 Open Directions

Open questions include using a polylogarithmic number of BIS queries to estimate the number of cliques in a graph (see the work of Eden et al. [20] for an algorithm using degree, neighbor, and edge existence queries) or to sample a uniformly random edge (see the work of Eden and Rosenbaum [21] for an algorithm using degree, neighbor, and edge existence queries). In general, any graph estimation problems may benefit from BIS or IS queries, possibly in combination with standard queries (e.g., neighbor queries). Finally, it would be interesting to know what other oracles, besides subset queries, enable estimating graph parameters with a polylogarithmic number of queries.

ACKNOWLEDGMENTS

We thank the anonymous referees for helpful comments about improving the presentation of our article and for pointing out relevant references.

REFERENCES

[1] Maryam Aliakbarpour, Amartya Shankha Biswas, Themis Gouleakis, John Peebles, Ronitt Rubinfeld, and Anak Yodpinyanee. 2018. Sublinear-time algorithms for counting star subgraphs via edge sampling. *Algorithmica* 80, 2 (2018), 668–697. DOI: https://doi.org/10.1007/s00453-017-0287-3

[2] Noga Alon and Vera Asodi. 2005. Learning a hidden subgraph. *SIAM Journal on Discrete Mathematics* 18, 4 (2005), 697–712.

[3] Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. 2004. Learning a hidden matching. *SIAM J. Comput.* 33, 2 (2004), 487–501.

[4] Dana Angluin and Jiang Chen. 2006. Learning a hidden hypergraph. *Journal of Machine Learning Research* 7, Oct. (2006), 2215–2236.

[5] Dana Angluin and Jiang Chen. 2008. Learning a hidden graph using $O(\log n)$ queries per edge. *J. Comput. Syst. Sci.* 74, 4 (June 2008), 546–556. DOI: https://doi.org/10.1016/j.jcss.2007.06.006

[6] B. Aronov and S. Har-Peled. 2008. On approximating the depth and related problems. *SIAM Journal on Computing* 38, 3 (2008), 899–921. DOI: https://doi.org/10.1137/060669474

[7] P. Beame, S. Har-Peled, S. Natarajan Ramamoorthy, C. Rashtchian, and M. Sinha. 2018. Edge estimation with independent set oracles. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, A. R. Karlin (Ed.). Leibniz International Proceedings in Informatics (LIPIcs), Vol. 94. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 38:1–38:21. DOI: https://doi.org/10.4230/LIPIcs.ITCS.2018.38

[8] I. Ben-Eliezer, T. Kaufman, M. Krivelevich, and D. Ron. 2013. Comparing the strength of query types in property testing: The case of k-colorability. *Computational Complexity* 22, 1 (2013), 89–135. DOI: https://doi.org/10.1007/s00037-012-0048-2

[9] Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. 2019. Hyperedge estimation using polylogarithmic subset queries. arxiv:1908.04196.

[10] Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. 2019. Triangle estimation using tripartite independent set queries. In 30th International Symposium on Algorithms and Computations (ISAAC 2019), P. Lu and G. Zhang (Eds.). Leibniz International Proceedings in Informatics (LIPIcs), Vol. 149. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 19:1–19:17. DOI: https://doi.org/10.4230/LIPIcs.ISAAC.2019.19

[11] S. Cabello and M. Jejčič. 2015. Shortest paths in intersection graphs of unit disks. *Computational Geometry* 48, 4 (2015), 360–367. DOI: https://doi.org/10.1016/j.comgeo.2014.12.003

[12] C. L. Chen and W. H. Swallow. 1990. Using group testing to estimate a proportion, and to test the binomial model. *Biometrics* 46, 4 (Dec. 1990), 1035–1046. DOI: https://doi.org/10.2307/2532446

[13] Xi Chen, Amit Levi, and Erik Waingarten. 2020. Nearly optimal edge estimation with independent set queries. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*. 2916–2935. DOI: https://doi.org/10.1137/1.9781611975994.177

[14] F. Chung and L. Lu. 2006. Concentration inequalities and martingale inequalities: A survey. *Internet Mathematics* 3, 1 (2006), 79–127. https://projecteuclid.org:443/euclid.im/1175266369

[15] Holger Dell and John Lapinskas. 2018. Fine-grained reductions from approximate counting to decision. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC'18)*. ACM, New York, NY, 281–288. DOI: https://doi.org/10.1145/3188745.3188920

[16] Holger Dell, John Lapinskas, and Kitty Meeks. 2020. Approximately counting and sampling small witnesses using a colourful decision oracle. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*. 2201–2211. DOI: https://doi.org/10.1137/1.9781611975994.135

[17] R. Dorfman. 1943. The detection of defective members of large populations. *Annals of Mathematical Statistics* 14, 4 (Dec. 1943), 436–440. DOI: https://doi.org/10.1214/aoms/1177731363

[18] D. P. Dubhashi and A. Panconesi. 2009. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press. http://www.cambridge.org/gb/knowledge/isbn/item2327542/.

[19] T. Eden, A. Levi, D. Ron, and C. Seshadhri. 2017. Approximately counting triangles in sublinear time. *SIAM Journal on Computing* 46, 5 (2017), 1603–1646. DOI: https://doi.org/10.1137/15M1054389

[20] T. Eden, D. Ron, and C. Seshadhri. 2017. On approximating the number of k-cliques in sublinear time. arxiv:cs.DS/1707.04858.

[21] T. Eden and W. Rosenbaum. 2018. On sampling edges almost uniformly. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*, R. Seidel (Ed.). OpenAccess Series in Informatics (OASIcs), Vol. 61. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 7:1–7:9. DOI: https://doi.org/10.4230/OASIcs.SOSA.2018.7

[22] M. Falahatgar, A. Jafarpour, A. Orlitsky, V. Pichapati, and A. T. Suresh. 2016. Estimating the number of defectives with group testing. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT'16)*. IEEE, Los Alamitos, CA, 1376–1380. DOI: https://doi.org/10.1109/ISIT.2016.7541524

[23] U. Feige. 2006. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing* 35, 4 (2006), 964–984. DOI: https://doi.org/10.1137/s0097539704447304

[24] A. V. Fishkin. 2003. Disk graphs: A short survey. In *Proceedings of the 1st International Workshop on Approximation and Online Algorithms (WAOA'03)*. 260–264. DOI: https://doi.org/10.1007/978-3-540-24592-6_23

[25] O. Goldreich and D. Ron. 2008. Approximating average parameters of graphs. *Random Structures & Algorithms* 32, 4 (2008), 473–493. DOI: https://doi.org/10.1002/rsa.v32:4

[26] M. Gonen, D. Ron, and Y. Shavitt. 2011. Counting stars and other small subgraphs in sublinear-time. *SIAM Journal on Discrete Mathematics* 25, 3 (2011), 1365–1411. DOI: https://doi.org/10.1137/100783066

[27] Vladimir Grebinski and Gregory Kucherov. 1998. Reconstructing a Hamiltonian cycle by querying the graph: Application to DNA physical mapping. *Discrete Applied Mathematics* 88, 1–3 (1998), 147–165.

[28] R. J. Klein, C. Zeiss, E. Chew, J.-Y. Tsai, R. S. Sackler, C. Haynes, A. K. Henning, et al. 2005. Complement factor H polymorphism in age-related macular degeneration. *Science* 308, 5720 (2005), 385–389. DOI: https://doi.org/10.1126/science.1109557

[29] K. Onak, D. Ron, M. Rosen, and R. Rubinfeld. 2012. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA'12).* 1123–1131. http://dl.acm.org/citation.cfm?id=2095116.2095204

[30] Lev Reyzin and Nikhil Srivastava. 2007. Learning and verifying graphs using queries with a focus on edge counting. In *International Conference on Algorithmic Learning Theory.* Springer, 285–297.

[31] D. Ron and G. Tsur. 2016. The power of an example: Hidden set size approximation using group queries and conditional sampling. *ACM Transactions on Computation Theory* 8, 4 (2016), Article 15, 19 pages. DOI: https://doi.org/10.1145/2930657

[32] C. Seshadhri. 2015. A simpler sublinear algorithm for approximating the triangle count. arxiv:cs.DS/1505.01927.

[33] L. Stockmeyer. 1983. The complexity of approximate counting (preliminary version). In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC'83).* 118–126. DOI: https://doi.org/10.1145/800061.808740

[34] L. Stockmeyer. 1985. On approximation algorithms for #P. *SIAM Journal on Computing* 14, 4 (1985), 849–861. DOI: https://doi.org/10.1137/0214060

[35] W. H. Swallow. 1985. Group testing for estimating infection rates and probabilities of disease transmission. *Phytopathology* 75, 8 (1985), 882. DOI: https://doi.org/10.1094/phyto-75-882

[36] J. Wang, E. Lo, and M. L. Yiu. 2013. Identifying the most connected vertices in hidden bipartite graphs using group testing. *IEEE Tranactions on Knowledge and Data Engineering* 25, 10 (2013), 2245–2256. DOI: https://doi.org/10.1109/TKDE.2012.178

Received September 2018; revised March 2020; accepted May 2020