A Simple Sublinear-Time Algorithm for Counting Arbitrary Subgraphs via Edge Sampling

Sepehr Assadi[1]
Department of Computer and Information Science, University of Pennsylvania
Philadelphia, PA, US.
sassadi@cis.upenn.edu

Michael Kapralov[2]
School of Computer and Communication Sciences, EPFL
Lausanne, Switzerland
michael.kapralov@epfl.ch

Sanjeev Khanna[3]
Department of Computer and Information Science, University of Pennsylvania
Philadelphia, PA, US.
sanjeev@cis.upenn.edu

Abstract

In the subgraph counting problem, we are given a (large) input graph $G(V, E)$ and a (small) target graph H (e.g., a triangle); the goal is to estimate the number of occurrences of H in G. Our focus here is on designing *sublinear-time* algorithms for approximately computing number of occurrences of H in G in the setting where the algorithm is given *query access* to G. This problem has been studied in several recent papers which primarily focused on specific families of graphs H such as triangles, cliques, and stars. However, not much is known about approximate counting of arbitrary graphs H in the literature. This is in sharp contrast to the closely related subgraph enumeration problem that has received significant attention in the database community as the database join problem. The AGM bound shows that the maximum number of occurrences of any arbitrary subgraph H in a graph G with m edges is $O(m^{\rho(H)})$, where $\rho(H)$ is the *fractional edge-cover* of H, and enumeration algorithms with matching runtime are known for any H.

We bridge this gap between subgraph counting and subgraph enumeration by designing a simple sublinear-time algorithm that can estimate the number of occurrences of any arbitrary graph H in G, denoted by $\#H$, to within a $(1 \pm \varepsilon)$-approximation with high probability in $O(\frac{m^{\rho(H)}}{\#H}) \cdot \text{poly}(\log n, 1/\varepsilon)$ time. Our algorithm is allowed the standard set of queries for general graphs, namely degree queries, pair queries and neighbor queries, plus an additional edge-sample query that returns an edge chosen uniformly at random. The performance of our algorithm matches those of Eden *et al.* [FOCS 2015, STOC 2018] for counting triangles and cliques and extend them to all choices of subgraph H under the additional assumption of edge-sample queries.

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

Keywords and phrases Sublinear-time algorithms, Subgraph counting, AGM bound

Digital Object Identifier 10.4230/LIPIcs.ITCS.2019.4

Related Version A full version is available on arXiv [4].

[1] Supported in part by the National Science Foundation grant CCF-1617851.
[2] Supported in part by ERC Starting Grant 759471.
[3] Supported in part by the National Science Foundation grants CCF-1617851 and CCF-1763514.

1 Introduction

Counting (small) subgraphs in massive graphs is a fundamental algorithmic problem, with a wide range of applications in bioinformatics, social network analysis, spam detection and graph databases (see, e.g. [36, 8, 11]). In social network analysis, the ratio of the number of triangles in a network to the number of length 2 paths (known as the clustering coefficient) as well as subgraph counts for larger subgraphs H have been proposed as important metrics for analyzing massive networks [42]. Similarly, motif counting are popular method for analyzing protein-protein interaction networks in bioinformatics (e.g., [36]). In this paper we consider designing efficient algorithms for this task.

Formally, we consider the following problem: Given a (large) graph $G(V, E)$ with m edges and a (small) subgraph $H(V_H, E_H)$ (e.g., a triangle) and a precision parameter $\varepsilon \in (0, 1)$, output a $(1 \pm \varepsilon)$-approximation to the number of occurrences of H in G. Our goal is to design an algorithm that runs in time *sublinear* in the number m of edges of G, and in particular makes a sublinear number of the following types of queries to the graph G:

- **Degree query** v**:** the degree d_v of any vertex $v \in V$;
- **Neighbor query** (v, i)**:** what vertex is the i-th neighbor of the vertex $v \in V$ for $i \leq d_v$;
- **Pair query** (u, v)**:** test for pair of vertices $u, v \in V$, whether or not (u, v) belongs to E.
- **Edge-sample query:** sample an edge e uniformly at random from E.

The first three queries are the standard baseline queries (see Chapter 10 of Goldreich's book [23]) assumed by nearly all sublinear time algorithms for counting small subgraphs such as triangles or cliques [16, 18] (see [25] for the necessity of using pair queries for counting subgraphs beside stars). The last query is somewhat less standard but is also considered in the literature prior to our work, for example in [2] for counting stars in sublinear time, and in [19] in the context of lower bounds for subgraph counting problems.

1.1 Our Contributions

For the sake of clarity, we suppress any dependencies on the approximation parameter ε, $\log n$-terms, and the size of graph H, using the $O^*(\cdot)$ notation. Our results are parameterized by the *fractional edge-cover number* of the subgraph H (see Section 3 for the formal definition). Our goal in this paper is to approximately compute the number of occurrences $\#H$ of H in G, formally defined as number of subgraphs H' of G such that H and H' are isomorphic.

▶ Theorem 1. There exists a randomized algorithm that given $\varepsilon \in (0, 1)$, a subgraph H, and a query access to the input graph G, with high probability outputs a $(1 \pm \varepsilon)$ approximation to the number of occurrences of H in G, denoted by $\#H$, using:

$$O^*\left(\min\left\{m, \frac{m^{\rho(H)}}{\#H}\right\}\right) \text{ queries and } O^*\left(\frac{m^{\rho(H)}}{\#H}\right) \text{ time.}$$

The algorithm uses degree, neighbor, pair, and edge-sample queries.

Since the fractional edge-cover number of any k-clique K_k is $k/2$, as a corollary of Theorem 1, we obtain sublinear algorithms for counting triangles, and in general k-cliques using

$$O^*\left(\min\left\{m, \frac{m\sqrt{m}}{\#K_3}\right\}\right) \text{ and } O^*\left(\min\left\{m, \frac{m^{k/2}}{\#K_k}\right\}\right),$$

queries respectively. These bounds match the previous results of Eden *et al.* [16, 18] modulo an additive term of $O^*(\frac{n}{(\#K_3)^{1/3}})$ for triangles in [16] and $O^*(\frac{n}{(\#K_k)^{1/k}})$ for arbitrary cliques in [18] which is needed in the absence of edge-sample queries that are not used by [16, 18]. Our bounds settle a conjecture of Eden and Rosenbaum [19] in the affirmative by showing that one can avoid the aforementioned additive terms depending on n in query complexity by allowing edge-sample queries. We now elaborate more on different aspects of Theorem 1.

AGM Bound and Database Joins. The problem of *enumerating* all occurrences of a graph H in a graph G and, more generally, the database join problem, has been considered extensively in the literature. A fundamental question here is that given a database with m entries (e.g. a graph G with m edges) and a conjunctive query H (e.g. a small graph H), what is the maximum possible size of the output of the query (e.g., number of occurrences of H in G)? The AGM bound of Atserias, Grohe and Marx [5] provides a tight upper bound of $m^{\rho(H)}$ (up to constant factors), where $\rho(H)$ is the fractional edge cover of H. The AGM bound applies to databases with several relations, and the fractional edge cover in question is weighted according to the sizes of the different relations. A similar bound on the number of occurrences of a hypergraph H inside a hypergraph G with m hyperedges was proved earlier by Friedgut and Kahn [22], and the bound for graphs is due to Alon [3]. Recent work of Ngo et al. [37] gave algorithms for evaluating database joins in time bounded by worst case output size for a database with the same number of entries. When instantied for the subgraph enumeration problem, their result gives an algorithm for enumerating all occurrences of H in a graph G with m edges in time $O(m^{\rho(H)})$.

Our Theorem 1 is directly motivated by the connection between subgraph counting and subgraph enumeration problems and the AGM bound. In particular, Theorem 1 provides a natural analogue of AGM bound for estimation algorithms by stating that if the number of occurrences H is $\#H \leq m^{\rho(H)}$, then a $(1 \pm \varepsilon)$-approximation to $\#H$ can be obtained in $O^*(\frac{m^{\rho(H)}}{\#H})$ time. Additionally, as we show in Section 4.3, Theorem 1 can be easily extended to the more general problem of database join size estimation (for binary relations). This problem corresponds to a subgraph counting problem in which the graphs G and H are both *edge-colored* and our goal is to count the number of copies of H in G with the same colors on edges. Our algorithm can solve this problem also in $O^*(\frac{m^{\rho(H)}}{\#H_c})$ time where $\#H_c$ denotes the number of copies of H with the same colors in G.

Optimality of Our Bounds. Our algorithm in Theorem 1 is optimal from different points of view. Firstly, by a lower bound of [19] (building on [16, 18]), the bounds achieved by our algorithm when H is any k-clique is optimal among all algorithms with the same query access (including the edge-sample query). In Theorem 14, we further prove a lower bound showing that for *odd cycles* as well, the bounds achieved by Theorem 1 are optimal. These results hence suggest that Theorem 1 is *existentially optimal*: there exists several natural choices for H such that Theorem 1 achieves the optimal bounds. However, there also exist choices of H for which the bounds in Theorem 1 are suboptimal. In particular, Aliakbarpour *et al.* [2] presented an algorithm for estimating occurrences of any star S_ℓ for $\ell \geq 1$ using $O^*(\frac{m}{(\#S_\ell)^{1/\ell}})$ queries in our query model (including edge-sample queries) which is always at least as good as our bound in Theorem 1, but potentially can be better. On the other hand, in the full version of the paper [4], we show that our current algorithm, with almost no further modification, in fact achieves this stronger bound *using a different analysis*.

Additionally, as we pointed out before, our algorithm can solve the more general database join size estimation for binary relations, or equivalently the subgraph counting problem with colors on edges. In Theorem 15, we prove that for this more general problem, our algorithm in Theorem 1 indeed achieves optimal bounds for *all choices* of the subgraph H.

Edge-Sample Queries. The edge-sample query that we assume is not part of the standard access model for sublinear algorithms, namely the "general graph" query model (see, e.g. [32]). Nonetheless, we find allowing for this query "natural" owing to the following factors:

Theoretical implementation. Edge sampling queries can be implemented with an $\widetilde{O}(n/\sqrt{m})$ multiplicative overhead in query and time using the recent result of [20], or with an $O(n)$ additive preprocessing time (which is still sublinear in m) by querying degrees of all vertices. Hence, we can focus on designing algorithms by allowing these queries and later replacing them by either of the above implementations in a black-box way at a certain additional cost.

Practical implementation. Edge sampling is a common practice in analyzing social networks [34, 33] or biological networks [1]. Another scenario when random edge sampling is possible is when we can access a random location of the memory that is used to store the graph. To quote [2]: "because edges normally take most of the space for storing graphs, an access to a random memory location where the adjacency list is stored, would readily give a random edge." Hence, assuming edge sampling queries can be considered valid in many scenarios.

Understanding the power of random edge queries. Edge sampling is a critical component of various sublinear time algorithms for graph estimation [16, 17, 2, 18, 20]. However, except for [2] that also assumed edge-sample queries, all these other algorithms employ different workarounds to these queries. As we show in this paper, decoupling these workarounds from the rest of the algorithm by allowing edge-sample queries results in considerably simpler and more general algorithms for subgraph counting and is hence worth studying on its own. We also mention that studying the power of edge-sample queries has been cast as an open question in [19] as well.

Applications to Streaming Algorithms. Subgraph counting is also one of the most studied problems in the graph streaming model (see, e.g. [6, 28, 10, 31, 9, 27, 41, 35, 13, 7] and references therein). In this model, the edges of the input graph are presented one by one in a stream; the algorithm makes a single or a small number of passes over the stream and outputs the answer after the last pass. The goal here is to minimize the memory used by the algorithm (similar-in-spirit to minimizing the query complexity in the query model).

Our algorithm in Theorem 1 can be directly adapted to the streaming model, resulting in an algorithm for subgraph counting that makes $O(1)$ passes over the stream and uses a memory of size $O^*\left(\min\left\{m, \frac{m^{\rho(H)}}{\#H}\right\}\right)$. For the case of counting triangles and cliques, the space complexity of our algorithm matches the best known algorithms of McGregor *et al.* [35] and Bera and Chakrabarti [7] which are known to be optimal [7]. To the best of our knowledge, the only previous streaming algorithms for counting arbitrary subgraphs H are those of Kane *et al.* [31] and Bera and Chakrabarti [7] that use, respectively, one pass and $O^*(\frac{m^{2\cdot|E(H)|}}{(\#H)^2})$ space, and two passes and $O^*(\frac{m^{\beta(H)}}{\#H})$ space, where $\beta(H)$ is the *integral* edge-cover number of H. As $\rho(H) \leq \beta(H) \leq |E(H)|$ by definition and $\#H \leq m^{\rho(H)}$ by AGM bound, the space complexity of our algorithm is always at least as good as the ones in [31, 7] but potentially can be much smaller.

1.2 Main Ideas in Our Algorithm

Our starting point is the AGM bound which implies that the number of "potential copies" of H in G is at most $m^{\rho(H)}$. Our goal of estimating $\#H$ then translates to counting how many of these potential copies form an actual copy of H in G. A standard approach at this point is the *Monte Carlo method:* sample a potential copy of H in G *uniformly at random*

and check whether it forms an actual copy of H or not; a simple exercise in concentration inequalities then implies that we only need $O(\frac{m^{\rho(H)}}{\#H})$ many *independent* samples to get a good estimate of $\#H$.

This approach however immediately runs into a technical difficulty. Given only a query access to G, it is not at all clear how to sample a potential copy of H from the list of all potential copies. Our first task is then to design a procedure for sampling potential copies of H from G. In order to do so, we again consider the AGM bound and the optimal fractional edge-cover that is used to derive this bound. We first prove a simple structural result that states that an optimal fractional edge-cover of H can be supported only on edges that form a disjoint union of *odd cycles* and *stars* (in H). This allows us to decompose H into a collection of odd cycles and stars and treat any arbitrary subgraph H as a collection of these simpler subgraphs that are suitably connected together.

The above decomposition reduces the task of sampling a potential copy of H to sampling a collection of odd cycles and stars. Sampling an odd cycle C_{2k+1} on $2k+1$ edges is as follows: sample k edges $e_1, \ldots, e_k$ uniformly at random from G; pick one of the endpoints of e_1 and sample a vertex v from the neighborhood of this endpoint uniformly at random. With some additional care, one can show that the tuple $(e_1, \ldots, e_k, v)$ sampled here is enough to identify an odd cycle of length $2k+1$ uniquely. To sample a star C_ℓ with ℓ petals, we sample a vertex v from G with probability proportional to its degree (by sampling a random edge and picking one of the two endpoints uniformly), and then sample ℓ vertices $w_1, \ldots, w_\ell$ from the neighborhood of v. Again, with some care, this allows us to sample a potential copy of a star S_ℓ. We remark that these sampling procedures are related to sampling triangles in [16] and stars in [2]. Finally, to sample a potential copy of H, we simply sample all its odd cycles and stars in the decomposition using the method above. We should note right away that this however does *not* result in a uniformly at random sample of potential copies of H as various parameters of the graph G, in particular degrees of vertices, alter the probability of sampling each potential copy.

The next and paramount step is then how to use the samples above to estimate the value of $\#H$. Obtaining an unbiased estimator of $\#H$ based on these samples is not hard as we can identify the probability each potential copy is sampled with in this process (which is a function of degrees of vertices of the potential copy in G) and reweigh each sample accordingly. Nevertheless, the variance of a vanilla variant of this sampling and reweighing approach is quite large for our purpose. To fix this, we use an idea similar to that of [16] for counting triangles: sample a "partial" potential copy of H first and fix it; sample *multiple* "extensions" of this partial potential copy to a complete potential copy and use the average of estimates based on each extension to reduce the variance. More concretely, this translates to sampling multiple copies of the first cycle for the decomposition and for each sampled cycle, recursively sampling multiple copies of the remainder of H as specified by the decomposition. A careful analysis of this recursive process—which is the main technical part of the paper—allows us to bound the variance of the estimator by $O(m^{\rho(H)}) \cdot (\#H)$. Repeating such an estimator $O(\frac{m^{\rho(H)}}{\#H})$ times independently and taking the average value then gives us a $(1 \pm \varepsilon)$-approximation to $\#H$ by a simple application of Chebyshev's inequality.

1.3 Further Related Work

In addition to the previous work in [16, 18, 2] that are already discussed above, sublinear-time algorithms for estimating subgraph counts and related parameters such as average degree and degree distribution moments have also been studied in [21, 24, 25, 17]. Similarly, sublinear-time algorithms are also studied for estimating other graph parameters such as weight of the

minimum spanning tree [15, 12, 14] or size of a maximum matching or a minimum vertex cover [40, 38, 43, 26, 39] (this is by no means a comprehensive summary of previous results).

Subgraph counting has also been studied extensively in the graph streaming model (see, e.g. [6, 28, 10, 31, 9, 27, 41, 35, 13, 7, 30, 29] and references therein). In this model, the edges of the input graph are presented one by one in a stream; the algorithm makes a single or a small number of passes over the stream and outputs the answer after the last pass. The goal in this model is to minimize the memory used by the algorithm similar-in-spirit to minimizing the query complexity in our query model. However, the streaming algorithms typically require reading the entire graph in the stream which is different from our goal in sublinear-time algorithms.

2 Preliminaries

Notation. For any integer $t \geq 1$, we let $[t] := \{1, \ldots, t\}$. For any event $\mathcal{E}$, $\mathbb{I}(\mathcal{E}) \in \{0, 1\}$ is an indicator denoting whether $\mathcal{E}$ happened or not. For a graph $G(V, E)$, $V(G) := V$ denotes the vertices and $E(G) := E$ denotes the edges. For a vertex $v \in V$, $N(v)$ denotes the neighbors of v, and $d_v := |N(v)|$ denotes the degree of v.

To any edge $e = \{u, v\}$ in G, we assign two directed edges $\vec{e}_1 = (u, v)$ and $\vec{e}_2 = (v, u)$ called the directed copies of e and let $\vec{E}$ be the set of all these directed edges. We also fix a total ordering $\prec$ on vertices whereby for any two vertices $u, v \in V$, $u \prec v$ iff $d_u < d_v$, or $d_u = d_v$ and u appears before v in the lexicographic order. To avoid confusion, we use letters a, b and c to denote the vertices in the subgraph H, and letters u, v and w to denote the vertices of G.

We use the following standard variant of Chebyshev's inequality.

▶ Proposition 1. For any random variable X and integer $t \geq 1$, $\Pr\left(|X - \mathbb{E}[X]| \geq t\right) \leq \frac{\mathbb{V}\mathrm{ar}[X]}{t^2}$.

We also recall the law of total variance that states the for two random variables X and Y,

$$\mathbb{V}\mathrm{ar}[Y] = \mathop{\mathbb{E}}_{x}\left(\mathbb{V}\mathrm{ar}[Y \mid X = x]\right) + \mathop{\mathbb{V}\mathrm{ar}}_{x}\left[\mathbb{E}[Y \mid X = x]\right]. \tag{1}$$

Assumption on Size of Subgraph H. Throughout the paper, we assume that the size of the subgraph H is a fixed constant independent of the size of the graph G and hence we suppress the dependency on size of H in various bounds in our analysis using O-notation.

3 A Graph Decomposition Using Fractional Edge-Covers

In this section, we give a simple decomposition of the subgraph H using fractional edge-covers. We start by defining fractional edge-covers formally (see also Figure 1).

▶ **Definition 2** (Fractional Edge-Cover Number)**.** A fractional edge-cover of $H(V_H, E_H)$ is a mapping $\psi : E_H \to [0, 1]$ such that for each vertex $a \in V_H$, $\sum_{e \in E_H, a \in e} \psi(e) \geq 1$. The fractional edge-cover number $\rho(H)$ of H is the minimum value of $\sum_{e \in E_H} \psi(e)$ among all fractional edge-covers ψ.

The fractional edge-cover number of a graph can be computed by the following LP:

$$\begin{aligned} \rho(H) \quad = \quad & \text{minimize} \quad && \sum_{e \in E(H)} x_e \\ & \text{subject to} \quad && \sum_{e \in E_H : a \in e} x_e \geq 1 \text{ for all vertices } a \in V(H). \end{aligned} \tag{2}$$

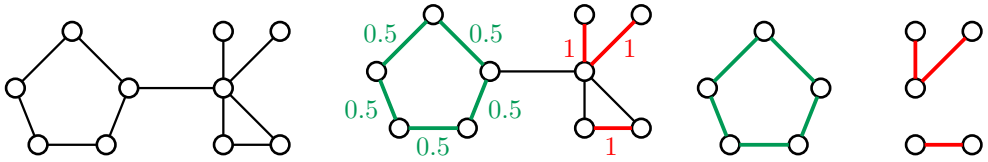
The following lemma is the key to our decomposition. The proof is based on standard ideas in linear programming and is postponed to the full version of the paper [4].

▶ **Lemma 3.** *Any subgraph H admits an optimal fractional edge-cover x^* such that the support of x^*, denoted by $\text{SUPP}(x^*)$, is a collection of vertex-disjoint odd cycles and star graphs, and,*

1. *for every odd cycle $C \in \text{SUPP}(x^*)$, $x^*_e = 1/2$ for all $e \in C$;*
2. *for every edge $e \in \text{SUPP}(x^*)$ that does* not *belong to any odd cycle, $x_e = 1$.*

3.1 The Decomposition

We now present the decomposition of H using Lemma 3. Let x^* be an optimal fractional edge-cover in Lemma 3 and let $\mathcal{C}_1, \ldots, \mathcal{C}_o$ be the odd-cycles in the support of x^* and $\mathcal{S}_1, \ldots, \mathcal{S}_s$ be the stars. We define $\mathcal{D}(H) := \{\mathcal{C}_1, \ldots, \mathcal{C}_o, \mathcal{S}_1, \ldots, \mathcal{S}_s\}$ as the decomposition of H (see Figure 1 below for an illustration).



(a) The subgraph H. **(b)** An optimal edge-cover of H with $\rho(H) = 5.5$. **(c)** Decomposition of H.

Figure 1 Illustration of the our decomposition for H based on fractional edge-covers.

For every $i \in [o]$, let the length of the odd cycle $\mathcal{C}_i$ be $2k_i + 1$ (i.e., $\mathcal{C}_i = C_{2k_i+1}$); we define $\rho^C_i := k_i + 1/2$. Similarly, for every $j \in [s]$, let the number of petals in $\mathcal{S}_j$ be ℓ_j (i.e., $\mathcal{S}_j = S_{\ell_j}$); we define $\rho^S_j := \ell_j$. By Lemma 3,

$$\rho(H) = \sum_{i=1}^{o} \rho^C_i + \sum_{j=1}^{s} \rho^S_j. \tag{3}$$

Recall that by AGM bound, the total number of copies of H possible in G is $m^{\rho(H)}$. We also use the following simple lemma which is a direct corollary of the AGM bound.

▶ **Lemma 4.** *Let $I := \{i_1, \ldots, i_o\}$ and $J := \{j_1, \ldots, j_s\}$ be subsets of $[o]$ and $[s]$, respectively. Suppose $\widetilde{H}$ is the subgraph of H on vertices of the odd cycles $\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_o}$ and stars $\mathcal{S}_{j_1}, \ldots, \mathcal{S}_{j_s}$. Then the total number of copies of $\widetilde{H}$ in G is at most $m^{\rho(\widetilde{H})}$ for $\rho(\widetilde{H}) \leq \sum_{i \in I} \rho^C_i + \sum_{j \in J} \rho^S_j$.*

Proof. Let x^* denote the optimal value of LP (2) in $\mathcal{D}(H)$. Define y^* as the projection of x^* to edges present in $\widetilde{H}$. It is easy to see that y^* is a feasible solution for LP (2) of $\widetilde{H}$ with value $\sum_{i \in I} \rho^C_i + \sum_{j \in J} \rho^S_j$. The lemma now follows from AGM bound for $\widetilde{H}$. ◀

3.2 Profiles of Cycles, Stars, and Subgraphs

We conclude this section by specifying the representation of the potential occurrences of the subgraph H in G based on the decomposition $\mathcal{D}(H)$.

Odd cycles: We represent a potential occurrence of an odd cycle C_{2k+1} in G as follows. Let $\boldsymbol{e} = (\vec{e}_1, \ldots, \vec{e}_k) \in \vec{E}^k$ be an ordered tuple of k directed copies of edges in G and suppose $\vec{e}_i := (u_i, v_i)$ for all $i \in [k]$. Define $u^*_{\boldsymbol{e}} = u_1$ and let w be any vertex in $N(u^*_{\boldsymbol{e}})$. We refer to any

such collection $(\boldsymbol{e}, w)$ as a *profile* of C_{2k+1} in G. We say that "the profile $(\boldsymbol{e}, w)$ forms a cycle C_{2k+1} in G" iff (i) u_1 is the smallest vertex on the cycle according to $\prec$, (ii) $v_1 \prec w$, *and* (iii) the edges $(u_1, v_1), (v_1, u_2), \ldots, (u_k, v_k), (v_k, w), (w, u_1)$ all exist in G and hence there is a copy of C_{2k+1} on vertices $\{u_1, v_1, u_2, v_2, \ldots, u_k, v_k, w\}$ in G. Note that under this definition and our definition of $\#C_{2k+1}$, each copy of C_{2k+1} correspond to exactly one profile $(\boldsymbol{e}, w)$ and vice versa. As such,

$$\#C_{2k+1} = \sum_{\boldsymbol{e} \in \vec{E}^k} \sum_{w \in N(u^*_{\boldsymbol{e}})} \mathbb{I}\Big((\boldsymbol{e}, w) \text{ forms a cycle } C_{2k+1} \text{ in } G\Big). \tag{4}$$

Stars: We represent a potential occurrence of a star S_ℓ in G by $(v, \boldsymbol{w})$ where v is the center of the star and $\boldsymbol{w} = (w_1, \ldots, w_\ell)$ are the ℓ petals. We refer to $(v, \boldsymbol{w})$ as a *profile* of S_ℓ in G. We say that "the profile $(v, \boldsymbol{w})$ forms a star S_ℓ in G" iff (i) $|\boldsymbol{w}| > 1$, *or* (ii) $(\ell =) |\boldsymbol{w}| = 1$ and $v \prec w_1$; in both cases there is a copy of S_ℓ on vertices $v, w_1, \ldots, w_\ell$. Under this definition, each copy of S_ℓ corresponds to exactly one profile $(v, \boldsymbol{w})$. As such,

$$\#S_\ell = \sum_{v \in V} \sum_{\boldsymbol{w} \in N(v)^\ell} \mathbb{I}\Big((v, \boldsymbol{w}) \text{ forms a star } S_\ell \text{ in } G\Big). \tag{5}$$

Arbitrary subgraphs: We represent a potential occurrence of H in G by an $(o+s)$-tuple $\boldsymbol{R} := ((\boldsymbol{e}_1, w_1), \ldots, (\boldsymbol{e}_o, w_o), (v_1, \boldsymbol{w}_1), \ldots, (v_s, \boldsymbol{w}_s))$ where $(\boldsymbol{e}_i, w_i)$ is a profile of the cycle $\mathcal{C}_i$ in $\mathcal{D}(H)$ and $(v_j, \boldsymbol{w}_j)$ is a profile of the star $\mathcal{S}_j$. We refer to $\boldsymbol{R}$ as a *profile* of H and say that "the profile $\boldsymbol{R}$ forms a copy of H in G" iff (i) each profile forms a corresponding copy of $\mathcal{C}_i$ or $\mathcal{S}_j$ in $\mathcal{D}(H)$, and (ii) the remaining edges of H between vertices specified by $\boldsymbol{R}$ all are present in G (note that by definition of the decomposition $\mathcal{D}(H)$, all vertices of H are specified by $\boldsymbol{R}$). As such,

$$\#H = \sum_{\boldsymbol{R}} \mathbb{I}\Big(\boldsymbol{R} \text{ forms a copy of } H \text{ in } G\Big) \cdot f(H), \tag{6}$$

for a fixed constant $f(H)$ depending only on H as defined below. Let $\pi : V_H \to V_H$ be an automorphism of H. Let $C_1, \ldots, C_o$, $S_1, \ldots, S_s$ denote the cycles and stars in the decomposition of H. We say that π is decomposition preserving if for every $i = 1, \ldots, o$ cycle C_i is mapped to a cycle of the same length and for every $i = 1, \ldots, s$ star S_i is mapped to a star with the same number of petals. Let the number of decomposition preserving automorphisms of H be denoted by Z, and define $f(H) = 1/Z$. Define the quantity $\widetilde{\#H} := \sum_{\boldsymbol{R}} \mathbb{I}\Big(\boldsymbol{R} \text{ forms a copy of } H \text{ in } G\Big)$ which is equal to $\#H$ modulo the scaling factor of $f(H)$. It is immediate that estimating $\#H$ and $\widetilde{\#H}$ are equivalent to each other and hence in the rest of the paper, with a slight abuse of notation, we use $\#H$ and $\widetilde{\#H}$ interchangeably.

4 A Sublinear-Time Algorithm for Subgraph Counting

We now present our sublinear time algorithm for approximately counting number of any given arbitrary subgraph H in an underlying graph G and prove Theorem 1. The main component of our algorithm is an unbiased estimator random variable for $\#H$ with low variance. The algorithm in Theorem 1 is then obtained by simply repeating this unbiased estimator in parallel enough number of times (based on the variance) and outputting the average value of these estimators.

4.1 A Low-variance Unbiased Estimator for $\#H$

We present a low-variance unbiased estimator for $\#H$ in this section. Our algorithm is a sampling based algorithm. In the following, we first introduce two separate subroutines for sampling odd cycles (`odd-cycle-sampler`) and stars (`star-sampler`), and then use these components in conjunction with the decomposition we introduced in Section 3, to present our full algorithm. We should right away clarify that `odd-cycle-sampler` and `star-sampler` are not exactly sampling a cycle or a star, but rather sampling a set of vertices and edges (in a non-uniform way) that can potentially form a cycle or star in G, i.e., they sample a profile of these subgraphs defined in Section 3.2.

The `odd-cycle-sampler` Algorithm

We start with the following algorithm for sampling an odd cycle C_{2k+1} for some $k \geq 1$. This algorithm outputs a simple data structure, named the *cycle-sampler tree*, that provides a convenient representation of the samples taken by our algorithm (see Definition 5 immediately after the description of the algorithm). This data structure can be easily avoided when designing a cycle counting algorithm, but will be quite useful for reasoning about the recursive structure of our sampling algorithm for general graphs H.

`odd-cycle-sampler`(G, C_{2k+1}).

1. Sample k directed edges $\boldsymbol{e} := (\vec{e}_1, \ldots, \vec{e}_k)$ uniformly at random (with replacement) from G with the constraint that for $\vec{e}_1 = (u_1, v_1)$, $u_1 \prec v_1$.
2. Let $u^*_{\boldsymbol{e}} := u_1$ and let $d^*_{\boldsymbol{e}} := d_{u^*_{\boldsymbol{e}}}$.
3. For $i = 1$ to $t_{\boldsymbol{e}} := \lceil d^*_{\boldsymbol{e}}/\sqrt{m} \rceil$: Sample a vertex w_i uniformly at random from $N(u^*_{\boldsymbol{e}})$.
4. Let $\boldsymbol{w} := (w_1, \ldots, w_{t_{\boldsymbol{e}}})$. Return the cycle-sampler tree $\mathcal{T}(\boldsymbol{e}, \boldsymbol{w})$ (see Definition 5).

▶ **Definition 5** (Cycle-Sampler Tree)**.** The cycle-sampler tree $\mathcal{T}(\boldsymbol{e}, \boldsymbol{w})$ for the tuple $(\boldsymbol{e}, \boldsymbol{w})$ sampled by `odd-cycle-sampler`(G, C_{2k+1}) is the following 2-level tree $\mathcal{T}$:

- Each node α of the tree contains two attributes: $\mathsf{label}[\alpha]$ which consists of some of the edges and vertices in $(\boldsymbol{e}, \boldsymbol{w})$, and an integer $\mathsf{value}[\alpha]$.
- For the root α_r of $\mathcal{T}$, $\mathsf{label}[\alpha_r] := \boldsymbol{e}$ and $\mathsf{value}[\alpha_r] := (2m)^k/2$.
  ($\mathsf{value}[\alpha_r]$ is equal to the inverse of the probability that $\boldsymbol{e}$ is sampled by `odd-cycle-sampler`).
- The root α_r has $t_{\boldsymbol{e}}$ child-nodes in $\mathcal{T}$ for a parameter $t_{\boldsymbol{e}} = \lceil d^*_{\boldsymbol{e}}/\sqrt{m} \rceil$ (consistent with line 3 of `odd-cycle-sampler`(G, C_{2k+1}) above).
- For the i-th child-node α_i of root, $i \in [t_{\boldsymbol{e}}]$, $\mathsf{label}[\alpha_i] := w_i$ and $\mathsf{value}[\alpha_i] := d^*_{\boldsymbol{e}}$
  ($\mathsf{value}[\alpha_i]$ is equal to the inverse of the probability that w_i is sampled by `odd-cycle-sampler`, conditioned on $\boldsymbol{e}$ being sampled).

Moreover, for each root-to-leaf path $\mathcal{P}_i := (\alpha_r, \alpha_i)$ (for $i \in [t_{\boldsymbol{e}}]$), define $\mathsf{label}[\mathcal{P}_i] := \mathsf{label}[\alpha_r] \cup \mathsf{label}[\alpha_i]$ and $\mathsf{value}[\mathcal{P}_i] := \mathsf{value}[\alpha_r] \cdot \mathsf{value}[\alpha_i]$ *($\mathsf{label}[\mathcal{P}_i]$ is a profile of the cycle C_{2k+1} as defined in Section 3.2).*

`odd-cycle-sampler` can be implemented in our query model by using k edge-sample queries (and picking the correct direction for e_1 based on $\prec$ and one of the two directions uniformly at random for the other edges) in Line (1), two degree queries in Line (2), and one neighbor query in Line (3). This results in $O(k)$ queries in total for one iteration of the for-loop in Line (3). As such, the total query complexity of `odd-cycle-sampler` is $O(t_{\boldsymbol{e}})$ (recall that k is a constant). It is also straightforward to verify that we can compute the

cycle-sampler tree $\mathcal{T}$ of an execution of `odd-cycle-sampler` with no further queries and in $O(t_{\boldsymbol{e}})$ time. We bound the query complexity of this algorithm by bounding the expected number of iterations in the for-loop. The proof is postponed to the full version [4].

▶ **Lemma 6.** *For the parameter $t_{\boldsymbol{e}}$ in Line (3) of* `odd-cycle-sampler`*, $\mathbb{E}[t_{\boldsymbol{e}}] = O(1)$.*

We now define a process for estimating the number of odd cycles in a graph using the information stored in the cycle-sampler tree and the `odd-cycle-sampler` algorithm. While we do not use this process in a black-box way in our main algorithm, abstracting it out makes the analysis of our main algorithm simpler to follow and more transparent, and serves as a warm-up for our main algorithm.

Warm-up: An Estimator for Odd Cycles. Let $\mathcal{T} := \texttt{odd-cycle-sampler}(G, C_{2k+1})$ be the output of an invocation of `odd-cycle-sampler`. Note that the cycle-sampler tree $\mathcal{T}$ is a random variable depending on the randomness of `odd-cycle-sampler`. We define the random variable X_i such that $X_i := \mathsf{label}[\mathcal{P}_i]$ for the i-th root-to-leaf path iff $\mathsf{label}[\mathcal{P}_i]$ forms a copy of C_{2k+1} in G and otherwise $X_i := 0$ (according to the definition of Section 3). We further define $Y := \frac{1}{t_{\boldsymbol{e}}} \cdot \sum_{i=1}^{t_{\boldsymbol{e}}} X_i$ (note that $t_{\boldsymbol{e}}$ is also a random variable). Our estimator algorithm can compute the value of these random variables using the information stored in the tree $\mathcal{T}$ plus additional $O(k) = O(1)$ queries for each of the $t_{\boldsymbol{e}}$ root-to-leaf path $\mathcal{P}_i$ to detect whether $(\boldsymbol{e}, w_i)$ forms a copy of H or not. Thus, the query complexity and runtime of the estimator is still $O(t_{\boldsymbol{e}})$ (which in expectation is $O(1)$ by Lemma 6). The expectation and variance of the estimator can be bounded as follows (the proof is in the full version [4]).

▶ **Lemma 7.** *For the random variable Y associated with* `odd-cycle-sampler`(G, C_{2k+1})*,*

$$\mathbb{E}[Y] = (\#C_{2k+1}), \qquad \mathbb{V}\mathrm{ar}[Y] \leq (2m)^k \sqrt{m} \cdot \mathbb{E}[Y].$$

The `star-sampler` Algorithm

We now give an algorithm for sampling a star S_ℓ with ℓ petals. Similar to `odd-cycle-sampler`, this algorithm also outputs a simple data structure, named the *star-sampler tree*, that provides a convenient representation of the samples taken by our algorithm (see Definition 8, immediately after the description of the algorithm). This data structure can be easily avoided when designing a star counting algorithm, but will be quite useful for reasoning about the recursive structure of our sampling algorithm for general graphs H.

`star-sampler`(G, S_ℓ).

1. Sample a vertex $v \in V$ chosen with probability proportional to its degree in G (i.e., for any vertex $u \in V$, $\Pr(u \text{ is chosen as the vertex } v) = d_u/2m$).
2. Sample ℓ vertices $\boldsymbol{w} := (w_1, \ldots, w_\ell)$ from $N(v)$ uniformly at random (without replacement).
3. Return the star-sampler tree $\mathcal{T}(v, \boldsymbol{w})$ (see Definition 8).

▶ **Definition 8** (Star-Sampler Tree)**.** The star-sampler tree $\mathcal{T}(v, \boldsymbol{w})$ for the tuple $(v, \boldsymbol{w})$ sampled by `star-sampler`(G, S_ℓ) is the following 2-level tree $\mathcal{T}$ (with the same attributes as in Definition 5) with only two nodes:

- For the root α_r of $\mathcal{T}$, $\mathsf{label}[\alpha_r] := v$ and $\mathsf{value}[\alpha_r] := 2m/d_v$.
  ($\mathsf{value}[\alpha_r]$ is equal to the inverse of the probability that v is sampled by `star-sampler`*).*

- The root α_r has exactly one child-node α_l in $\mathcal{T}$ with $\mathsf{label}[\alpha_l] = \boldsymbol{w} = (w_1, \ldots, w_\ell)$ and $\mathsf{value}[\alpha_l] = \binom{d_v}{\ell}$.
  ($\mathsf{value}[\alpha_l]$ is equal to the inverse of the probability that $\boldsymbol{w}$ is sampled by `star-sampler`, conditioned on v being sampled*)*.

Moreover, for the root-to-leaf path $\mathcal{P} := (\alpha_r, \alpha_l)$, we define $\mathsf{label}[\mathcal{P}] := \mathsf{label}[\alpha_r] \cup \mathsf{label}[\alpha_l]$ and $\mathsf{value}[\mathcal{P}] := \mathsf{value}[\alpha_r] \cdot \mathsf{value}[\alpha_l]$. *(*$\mathsf{label}[\mathcal{P}]$ is a representation of the star S_ℓ as defined in Section 3.2*)*.

`star-sampler` can be implemented in our query model by using one edge-sample query in Line (1) and then picking one of the endpoints uniformly at random, a degree query to determine the degree of v, and ℓ neighbor queries in Line (2), resulting in $O(\ell)$ queries in total. It is also straightforward to verify that we can compute the star-sampler tree $\mathcal{T}$ of an execution of `star-sampler` with no further queries and in $O(1)$ time.

We again define a process for estimating the number of stars in a graph using the information stored in the star-sampler tree and the `star-sampler` algorithm, as a warm-up to our main result in the next section.

Warm-up: An Estimator for Stars. The star-sampler tree $\mathcal{T}$ is a random variable depending on the randomness of `star-sampler`. We define the random variable X such that $X := \mathsf{value}[\mathcal{P}]$ for the root-to-leaf path of $\mathcal{T}$ iff $\mathsf{label}[\mathcal{P}]$ forms a copy of S_ℓ in G and otherwise $X := 0$. Our estimator algorithm can compute the value of this random variable using only the information stored in the tree $\mathcal{T}$ with no further queries to the graph (by simply checking if all w_i's in $\boldsymbol{w}$ are distinct). As such, the query complexity and runtime of the estimator algorithm is still $O(1)$. The proof of the following lemma is postponed to the full version [4].

▶ **Lemma 9.** *For the random variable X associated with* `star-sampler`(G, S_ℓ)*,*

$$\mathbb{E}[X] = (\#S_\ell), \qquad \mathbb{V}\mathrm{ar}[X] \leq 2m^\ell \cdot \mathbb{E}[X].$$

The Estimator Algorithm for Arbitrary Subgraphs

We now present our main estimator for the number of occurrences of an arbitrary subgraph H in G, denoted by $(\#H)$. Recall the decomposition $\mathcal{D}(H) := \{\mathcal{C}_1, \ldots, \mathcal{C}_o, \mathcal{S}_1, \ldots, \mathcal{S}_s\}$ of H introduced in Section 3. Our algorithm creates a *subgraph-sampler tree* $\mathcal{T}$ (a generalization of cycle-sampler and star-sampler trees in Definitions 5 and 8) and use it to estimate $(\#H)$. We define the subgraph-sampler tree $\mathcal{T}$ and the algorithm `subgraph-sampler`(G, H) that creates it simultaneously:

Subgraph-Sampler Tree. The subgraph-sampler tree $\mathcal{T}$ is a z-level tree for $z := (2o + 2s)$ returned by `subgraph-sampler`(G, H). The algorithm constructs $\mathcal{T}$ as follows.

Sampling Odd Cycles. In `subgraph-sampler`(G, H), we run `odd-cycle-sampler`$(G, \mathcal{C}_1)$ and initiate $\mathcal{T}$ to be its output cycle-sampler tree. For every (current) leaf-node α of $\mathcal{T}$, we run `odd-cycle-sampler`$(G, \mathcal{C}_2)$ *independently* to obtain a cycle-sampler tree $\mathcal{T}_\alpha$ (we say that α *started* the sampling of $\mathcal{T}_\alpha$). We then extend the tree $\mathcal{T}$ with two new layers by connecting each leaf-node α to the root of $\mathcal{T}_\alpha$ that started its sampling. This creates a 4-level tree $\mathcal{T}$. We continue like this for o steps, each time appending the tree obtained by `odd-cycle-sampler`$(G, \mathcal{C}_j)$ for $j \in [o]$, to the (previous) leaf-node that started this sampling. This results in a $(2o)$-level tree. Note that the nodes in the tree $\mathcal{T}$ can have different degrees as the number of leaf-nodes in the cycle-sampler tree is not necessarily the same always (not even for two different trees associated with one single $\mathcal{C}_j$ through different calls to `odd-cycle-sampler`$(G, \mathcal{C}_j)$).

Sampling Stars. Once we iterated over all odd cycles of $\mathcal{D}(H)$, we switch to processing stars $\mathcal{S}_1, \ldots, \mathcal{S}_s$. The approach is identical to the previous part. Let α be a (current) leaf-node of $\mathcal{T}$. We run $\texttt{star-sampler}(G, \mathcal{S}_1)$ to obtain a star-sampler tree $\mathcal{T}_\alpha$ and connect α to $\mathcal{T}_\alpha$ to extend the levels of tree by 2 more. We continue like this for s steps, each time appending the tree obtained by $\texttt{star-sampler}(G, \mathcal{S}_j)$ for $j \in [s]$, to the (former) leaf-node that started this sampling. This results in a z-level tree $\mathcal{T}$. Note that all nodes added when sampling stars have exactly one child-node (except for the leaf-nodes) as by Definition 8, star-sampler trees always contain only two nodes.

Labels and Values. Each node α of $\mathcal{T}$ is again given two attributes, $\mathsf{label}[\alpha]$ and $\mathsf{value}[\alpha]$, which are defined to be exactly the same attributes in the corresponding cycle-sampler or star-sampler tree that was used to define these nodes (recall that each node of $\mathcal{T}$ is "copied" from a node in either a cycle-sampler or a star-sampler tree). Finally, for each root-to-leaf path $\mathcal{P}$ in $\mathcal{T}$, we define $\mathsf{label}[\mathcal{P}] := \bigcup_{\alpha \in \mathcal{P}} \mathsf{label}[\alpha]$ and $\mathsf{value}[\mathcal{P}] := \prod_{\alpha \in \mathcal{P}} \mathsf{value}[\alpha]$. In particular, $\mathsf{label}[\mathcal{P}] := ((\boldsymbol{e}_1, w_1), \ldots, (\boldsymbol{e}_o, w_o), (v_1, \boldsymbol{w}_1), \ldots, (v_s, \boldsymbol{w}_s))$ by definition of labels of cycle-sampler and star-sampler trees. As such $\mathsf{label}[\mathcal{P}]$ is a representation of the subgraph H as defined in Section 3.2. By making $O(1)$ additional pair-queries to query all the remaining edges of this representation of H we determine if $\mathsf{label}[\mathcal{P}]$ forms a copy of H.

This concludes the description of $\texttt{subgraph-sampler}(G, H)$ and its output subgraph-sampler tree $\mathcal{T}$. We bound the query complexity of the algorithm in the following lemma (the proof is postponed to the full version [4]).

▶ **Lemma 10.** *The expected query complexity/ running time of* `subgraph-sampler` *is* $O(1)$.

We are now ready to present our estimator algorithm using `subgraph-sampler` and the subgraph-sampler tree $\mathcal{T}$ it outputs.

An Estimator for Arbitrary Subgraphs. Note that as before the subgraph-sampler tree $\mathcal{T}$ itself is a random variable depending on the randomness of `subgraph-sampler`. For any root-to-leaf path $\mathcal{P}_i := \alpha_1, \ldots, \alpha_z$ of $\mathcal{T}$, we define the random variable X_i such that $X_i := \mathsf{value}[\mathcal{P}_i]$ iff $\mathsf{label}[\mathcal{P}_i]$ forms a copy of H in G and otherwise $X_i := 0$. We further define $Y := (\frac{1}{t} \sum_{i=1}^{t} X_i)$, where t is the number of leaf-nodes of $\mathcal{T}$ (which itself is a random variable). These random variables can all be computed from $\mathcal{T}$ and `subgraph-sampler` with at most $O(1)$ further pair-queries per each root-to-leaf path $\mathcal{P}$ of the tree to determine if indeed $\mathsf{label}[\mathcal{P}]$ forms a copy of H in G or not. As such, query complexity and runtime of this algorithm is proportional to `subgraph-sampler` (which in expectation is $O(1)$ by Lemma 10). In the following two lemmas, we show that Y is a low-variance unbiased estimator of $(\#H)$.

Notation. For any node α in $\mathcal{T}$, we use $\mathcal{T}_\alpha$ to denote the sub-tree of $\mathcal{T}$ rooted at α. For a leaf-node α, we define a random variable Y_α which is $\mathsf{value}[\alpha]$ iff for the root-to-leaf path $\mathcal{P}$ ending in α, $\mathsf{label}[\mathcal{P}]$ forms a copy of H in G and otherwise Y_α is 0. For an internal node α in $\mathcal{T}$ with t child-nodes $\alpha_1, \ldots, \alpha_t$, we define $Y_\alpha = \mathsf{value}[\alpha] \cdot \left(\frac{1}{t} \cdot \sum_{i=1}^{t} Y_i\right)$. It is easy to verify that Y_{α_r} for the root α_r of $\mathcal{T}$ is the same as the estimator random variable Y defined earlier. Furthermore, for a node α in level ℓ of $\mathcal{T}$, we define $\boldsymbol{L}_\alpha := (\mathsf{label}[\alpha_1], \mathsf{label}[\alpha_2], \ldots, \mathsf{label}[\alpha_{\ell-1}])$, where $\alpha_1, \ldots, \alpha_{\ell-1}$ forms the path from the root of $\mathcal{T}$ to the parent of α.

We analyze the expected value and the variance of the estimator.

▶ **Lemma 11.** *For Y in* `subgraph-sampler`(G, H), $\mathbb{E}[Y] = (\#H)$.

Proof. We prove this inductively by showing that for any node α in an *odd layer* of $\mathcal{T}$, $\mathbb{E}[Y_\alpha \mid \boldsymbol{L}_\alpha] = (\#H \mid \boldsymbol{L}_\alpha)$, where $(\#H \mid \boldsymbol{L}_\alpha)$ denotes the number of copies of H in G that contain the vertices and edges specified by $\boldsymbol{L}_\alpha$ (according to the decomposition $\mathcal{D}(H)$).

$\mathbb{E}[Y_\alpha \mid \boldsymbol{L}_\alpha]$ measures the value of Y_α after we fix the rest of the tree $\mathcal{T}$ and let the sub-tree $\mathcal{T}_\alpha$ be chosen randomly as in `subgraph-sampler`.

The base case of the induction, i.e., for vertices in the last odd layer of $\mathcal{T}$ follows exactly as in the proofs of Lemmas 7 and 9 (as will also become evident shortly) and hence we do not repeat it here. We now prove the induction hypothesis. Fix a vertex α in an odd layer ℓ. We consider two cases based on whether $\ell < 2o$ (hence α is root of a cycle-sampler tree) or $\ell > 2o$ (hence α is root of a star-sampler tree).

Case of $\ell < 2o$. In this case, the sub-tree $\mathcal{T}_\alpha$ in the next two levels is a cycle-sampler tree,

$$\mathbb{E}[Y_\alpha \mid \boldsymbol{L}_\alpha] = \sum_{\boldsymbol{e}} \Pr(\mathsf{label}[\alpha] = \boldsymbol{e}) \cdot \mathsf{value}[\alpha] \cdot \left(\frac{1}{t_{\boldsymbol{e}}} \sum_{i=1}^{t_{\boldsymbol{e}}} \mathbb{E}[Y_{\alpha_i} \mid \boldsymbol{L}_\alpha, \boldsymbol{e}]\right)$$

(here, α_i's are child-nodes of α)

$$= \sum_{\boldsymbol{e}} \frac{1}{t_{\boldsymbol{e}}} \sum_{i=1}^{t_{\boldsymbol{e}}} \mathbb{E}[Y_{\alpha_i} \mid \boldsymbol{L}_\alpha, \boldsymbol{e}] \quad (\text{as by definition, } \mathsf{value}[\alpha] = \Pr(\mathsf{label}[\alpha] = \boldsymbol{e})^{-1})$$

Note that each α_i has exactly one child-node, denoted by β_i. As such,

$$\mathbb{E}[Y_\alpha \mid \boldsymbol{L}_\alpha] = \sum_{\boldsymbol{e}} \frac{1}{t_{\boldsymbol{e}}} \sum_{i=1}^{t_{\boldsymbol{e}}} \mathbb{E}[Y_{\alpha_i} \mid \boldsymbol{L}_\alpha, \boldsymbol{e}]$$

$$= \sum_{\boldsymbol{e}} \frac{1}{t_{\boldsymbol{e}}} \sum_{i=1}^{t_{\boldsymbol{e}}} \sum_{w} \Pr(\mathsf{label}[\alpha_i] = w) \cdot \mathsf{value}[\alpha_i] \cdot \mathbb{E}[Y_{\beta_i} \mid \boldsymbol{L}_\alpha, \boldsymbol{e}, w]$$

$$= \sum_{\boldsymbol{e}} \frac{1}{t_{\boldsymbol{e}}} \sum_{i=1}^{t_{\boldsymbol{e}}} \sum_{w} \mathbb{E}[Y_{\beta_i} \mid \boldsymbol{L}_{\beta_i}]$$

(by definition $\mathsf{value}[\alpha_i] = \Pr(\mathsf{label}[\alpha_i] = w)^{-1}$ and $\boldsymbol{L}_{\beta_i} = \boldsymbol{L}_\alpha, (\boldsymbol{e}, w)$)

$$= \sum_{\boldsymbol{e}} \frac{1}{t_{\boldsymbol{e}}} \sum_{i=1}^{t_{\boldsymbol{e}}} \sum_{w} (\#H \mid \boldsymbol{L}_{\beta_i}) = \sum_{\boldsymbol{e}} \frac{1}{t_{\boldsymbol{e}}} \sum_{i=1}^{t_{\boldsymbol{e}}} \sum_{w} (\#H \mid \boldsymbol{L}_\alpha, (\boldsymbol{e}, w))$$

(by induction hypothesis for odd-layer nodes β_i's)

$$= \sum_{\boldsymbol{e}} \sum_{w} (\#H \mid \boldsymbol{L}_\alpha, (\boldsymbol{e}, w)) = (\#H \mid \boldsymbol{L}_\alpha).$$

This concludes the proof of induction hypothesis in this case.

Case of $\ell > 2o$. In this case, the sub-tree $\mathcal{T}_\alpha$ in the next two levels is a star-sampler tree. By the same analogy made in the proof of the previous part and Lemma 7, the proof of this part also follows directly from the proof of Lemma 9 for star-sampler trees.

We can now finalize the proof of Lemma 11, by noting that for the root α_r of $\mathcal{T}$, $\boldsymbol{L}_{\alpha_r}$ is the empty-set and hence, $\mathbb{E}[Y] = \mathbb{E}[Y_{\alpha_r} \mid \boldsymbol{L}_{\alpha_r}]$, which by induction is equal to $(\#H)$. ◀

▶ **Lemma 12.** *For Y in* `subgraph-sampler`(G, H), $\mathbb{V}\mathrm{ar}[Y] = O(m^{\rho(H)}) \cdot \mathbb{E}[Y]$.

Proof. We bound $\mathbb{V}\mathrm{ar}[Y]$ using a similar inductive proof as in Lemma 11. Recall the parameters $\rho_1^C, \ldots, \rho_o^C$ and $\rho_1^S, \ldots, \rho_s^S$ associated respectively with the cycles $\mathcal{C}_1, \ldots, \mathcal{C}_o$ and stars $\mathcal{S}_1, \ldots, \mathcal{S}_s$ of the decomposition $\mathcal{D}(H)$. For simplicity of notation, for any $i \in [o+s]$, we define ρ_{i+} as follows:

$$\text{for all } i \leq o,\ \rho_{i+} := \sum_{j=i}^{o} \rho_j^C + \sum_{j=1}^{s} \rho_j^S, \qquad \text{for all } o < i \leq o+s,\ \rho_{i+} := \sum_{j=i-o}^{s} \rho_j^S.$$

We inductively show that, for any node α in an *odd layer* $2\ell-1$ of $\mathcal{T}$,

$$\mathbb{V}\mathrm{ar}\left[Y_\alpha \mid \boldsymbol{L}_\alpha\right] \leq 2^{2z-2\ell} \cdot m^{\rho_{\ell+}} \cdot (\#H \mid \boldsymbol{L}_\alpha),$$

where $(\#H \mid \boldsymbol{L}_\alpha)$ denotes the number of copies of H in G that contain the vertices and edges specified by $\boldsymbol{L}_\alpha$ (according to the decomposition $\mathcal{D}(H)$).

The induction is from the leaf-nodes of the tree to the root. The base case of the induction, i.e., for vertices in the last odd layer of $\mathcal{T}$ follows exactly as in the proofs of Lemmas 7 and 9 (as will also become evident shortly) and hence we do not repeat it here. We now prove the induction hypothesis. Fix a vertex α in an odd layer $2\ell-1$. We consider two cases based on whether $\ell \leq o$ (hence α is root of a cycle-sampler tree) or $\ell > o$ (hence α is root of a star-sampler tree).

Case of $\ell \leq o$. In this case, the sub-tree $\mathcal{T}_\alpha$ in the next two levels is a cycle-sampler tree corresponding to the odd cycle $\mathcal{C}_\ell$ of $\mathcal{D}(H)$. Let the number of edges in C_ℓ be $(2k+1)$ (i.e., $\mathcal{C}_\ell = C_{2k+1}$) Let $\boldsymbol{e}$ denote the label of the α. By the law of total variance in Eq (1)

$$\mathbb{V}\mathrm{ar}\left[Y_\alpha \mid \boldsymbol{L}_\alpha\right] = \mathbb{E}\left[\mathbb{V}\mathrm{ar}\left[Y_\alpha \mid \boldsymbol{e}\right] \mid \boldsymbol{L}_\alpha\right] + \mathbb{V}\mathrm{ar}\left[\mathbb{E}\left[Y_\alpha \mid \boldsymbol{e}\right] \mid \boldsymbol{L}_\alpha\right]. \tag{7}$$

We start by bounding the second term in Eq (7) which is easier. By the inductive proof of Lemma 11, we also have, $\mathbb{E}\left[Y_\alpha \mid \boldsymbol{L}_\alpha, \boldsymbol{e}\right] = (\#H \mid \boldsymbol{L}_\alpha, \boldsymbol{e})$. As such,

$$\begin{aligned}
\mathbb{V}\mathrm{ar}\left[\mathbb{E}\left[Y_\alpha \mid \boldsymbol{e}\right] \mid \boldsymbol{L}_\alpha\right] &= \mathbb{V}\mathrm{ar}\left[(\#H \mid \boldsymbol{L}_\alpha, \boldsymbol{e}) \mid \boldsymbol{L}_\alpha\right] \leq \mathbb{E}\left[(\#H \mid \boldsymbol{L}_\alpha, \boldsymbol{e})^2 \mid \boldsymbol{L}_\alpha\right] \\
&= \sum_{\boldsymbol{e}} \Pr\left(\mathsf{label}[\alpha] = \boldsymbol{e}\right) \cdot (\#H \mid \boldsymbol{L}_\alpha, \boldsymbol{e})^2 = \frac{1}{m^k} \sum_{\boldsymbol{e}} (\#H \mid \boldsymbol{L}_\alpha, \boldsymbol{e})^2 \\
&\qquad \left(\Pr\left(\mathsf{label}[\alpha] = \boldsymbol{e}\right) = 1/m^k \text{ by definition of } \texttt{odd-cycle-sampler}\right) \\
&\leq \frac{1}{m^k} \Big(\sum_{\boldsymbol{e}} (\#H \mid \boldsymbol{L}_\alpha, \boldsymbol{e}) \Big)^2 = \frac{1}{m^k} (\#H \mid \boldsymbol{L}_\alpha)^2 \\
&\leq m^{\rho_{\ell+}} \cdot (\#H \mid \boldsymbol{L}_\alpha).
\end{aligned} \tag{8}$$

The reason behind the last equality is that $(\#H \mid \boldsymbol{L}_\alpha)$ is at most equal to the number of copies of the subgraph of H consisting of $\mathcal{C}_\ell, \ldots, \mathcal{C}_o, \mathcal{S}_1, \ldots, \mathcal{S}_s$, which by Lemma 4 is at most $m^{\rho_{\ell+}}$ by definition of $\rho_{\ell+}$. We now bound the first and the main term in Eq (7),

$$\begin{aligned}
\mathbb{E}\left[\mathbb{V}\mathrm{ar}\left[Y_\alpha \mid \boldsymbol{e}\right] \mid \boldsymbol{L}_\alpha\right] &= \sum_{\boldsymbol{e}} \Pr\left(\mathsf{label}[\alpha] = \boldsymbol{e}\right) \cdot \mathbb{V}\mathrm{ar}\left[Y_\alpha \mid \boldsymbol{e}, \boldsymbol{L}_\alpha\right] \\
&= \sum_{\boldsymbol{e}} \frac{1}{m^k} \cdot m^{2k} \cdot \frac{1}{t_{\boldsymbol{e}}^2} \cdot \sum_{i=1}^{t_{\boldsymbol{e}}} \mathbb{V}\mathrm{ar}\left[Y_{\alpha_i} \mid \boldsymbol{e}, \boldsymbol{L}_\alpha\right], \\
&\qquad \text{(here } \alpha_i\text{'s are child-nodes of } \alpha\text{)}
\end{aligned}$$

where the final equality holds because Y_{α_i}'s are independent conditioned on $\boldsymbol{e}, \boldsymbol{L}_\alpha$ and since Y_α is by definition m^k times the average of Y_{α_i}'s. Moreover, note that distribution of all Y_{α_i}'s are the same. Hence, by canceling the terms,

$$\mathbb{E}\left[\mathbb{V}\mathrm{ar}\left[Y_\alpha \mid \boldsymbol{e}\right] \mid \boldsymbol{L}_\alpha\right] = m^k \cdot \sum_{\boldsymbol{e}} \frac{1}{t_{\boldsymbol{e}}} \cdot \mathbb{V}\mathrm{ar}\left[Y_{\alpha_1} \mid \boldsymbol{e}, \boldsymbol{L}_\alpha\right], \tag{9}$$

We thus only need to bound $\mathbb{V}\mathrm{ar}\left[Y_{\alpha_1} \mid \boldsymbol{e}, \boldsymbol{L}_\alpha\right]$. Recall that α_1 corresponds to a leaf-node in a cycle-sampler tree and hence its label is a vertex w from the neighborhood of $u^*_{\boldsymbol{e}}$ as defined in `odd-cycle-sampler`. We again use the law of total variance in Eq (1) to obtain,

$$\mathbb{V}\mathrm{ar}\left[Y_{\alpha_1} \mid \boldsymbol{e}, \boldsymbol{L}_\alpha\right] = \mathbb{E}\left[\mathbb{V}\mathrm{ar}\left[Y_{\alpha_1} \mid w\right] \mid \boldsymbol{e}, \boldsymbol{L}_\alpha\right] + \mathbb{V}\mathrm{ar}\left[\mathbb{E}\left[Y_{\alpha_1} \mid w\right] \mid \boldsymbol{e}, \boldsymbol{L}_\alpha\right] \tag{10}$$

For the first term,

$$\begin{aligned}\mathbb{E}\left[\mathbb{V}\mathrm{ar}\left[Y_{\alpha_1} \mid w\right] \mid \boldsymbol{e}, \boldsymbol{L}_\alpha\right] &= \sum_{w \in N(u^*_{\boldsymbol{e}})} \Pr\left(\mathsf{label}[\alpha_1] = w\right) \cdot \mathbb{V}\mathrm{ar}\left[Y_{\alpha_1} \mid w, \boldsymbol{e}, \boldsymbol{L}_\alpha\right] \\ &= \sum_{w} \frac{1}{d^*_{\boldsymbol{e}}} \cdot \left(d^*_{\boldsymbol{e}}\right)^2 \cdot \mathbb{V}\mathrm{ar}\left[Y_{\beta_1} \mid w, \boldsymbol{e}, \boldsymbol{L}_\alpha\right],\end{aligned}$$

where β_1 is the unique child-node of α_1 and so $Y_{\alpha_1} = \mathsf{value}[\alpha_1] \cdot Y_{\beta_1}$, while conditioned on $\boldsymbol{e}$, $\mathsf{value}[\alpha_1] = d^*_{\boldsymbol{e}}$. Moreover, as $\boldsymbol{L}_{\beta_1} = (\boldsymbol{L}_\alpha, \boldsymbol{e}, w)$, and by canceling the terms,

$$\begin{aligned}\mathbb{E}\left[\mathbb{V}\mathrm{ar}\left[Y_{\alpha_1} \mid w\right] \mid \boldsymbol{e}, \boldsymbol{L}_\alpha\right] &= \sum_{w} d^*_{\boldsymbol{e}} \cdot \mathbb{V}\mathrm{ar}\left[Y_{\beta_1} \mid \boldsymbol{L}_{\beta_1}\right] \\ &\leq \sum_{w} d^*_{\boldsymbol{e}} \cdot 2^{2z-2\ell-2} \cdot m^{\rho_{(\ell+1)+}} \cdot (\#H \mid \boldsymbol{L}_{\beta_1}), \qquad (11)\end{aligned}$$

where the inequality is by induction hypothesis for the odd-level node β_1. We now bound the second term in Eq (10) as follows,

$$\begin{aligned}\mathbb{V}\mathrm{ar}\left[\mathbb{E}\left[Y_{\alpha_1} \mid w\right] \mid \boldsymbol{e}, \boldsymbol{L}_\alpha\right] &\leq \mathbb{E}\left[\Big(\mathbb{E}\left[Y_{\alpha_1} \mid w\right]\Big)^2 \mid \boldsymbol{e}, \boldsymbol{L}_\alpha\right] \\ &= \sum_{w} \Pr\left(\mathsf{label}[\alpha_1] = w\right) \cdot \Big(\mathbb{E}\left[Y_{\alpha_1} \mid w, \boldsymbol{e}, \boldsymbol{L}_\alpha\right]\Big)^2 \\ &= \sum_{w} \frac{1}{d^*_{\boldsymbol{e}}} \cdot \left(d^*_{\boldsymbol{e}}\right)^2 \cdot \Big(\mathbb{E}\left[Y_{\beta_1} \mid w, \boldsymbol{e}, \boldsymbol{L}_\alpha\right]\Big)^2 \\ &= \sum_{w} d^*_{\boldsymbol{e}} \cdot \Big(\mathbb{E}\left[Y_{\beta_1} \mid \boldsymbol{L}_{\beta_1}\right]\Big)^2 = \sum_{w} d^*_{\boldsymbol{e}} \cdot (\#H \mid \boldsymbol{L}_{\beta_1})^2 \\ &\leq \sum_{w} d^*_{\boldsymbol{e}} \cdot m^{\rho_{(\ell+1)+}} \cdot (\#H \mid \boldsymbol{L}_{\beta_1}). \qquad (12)\end{aligned}$$

Here, the second to last equality holds by the inductive proof of Lemma 11, and the last equality is because $(\#H \mid \boldsymbol{L}_{\beta_1}) \leq m^{\rho_{(\ell+1)+}}$ by Lemma 4, as $(\#H \mid \boldsymbol{L}_{\beta_1})$ is at most equal to the total number of copies of a subgraph of H on $\mathcal{C}_{\ell+1}, \ldots, \mathcal{C}_o, \mathcal{S}_1, \ldots, \mathcal{S}_s$ (and by definition of $\rho_{(\ell+1)+}$). We now plug in Eq (11) and Eq (12) in Eq (10),

$$\mathbb{V}\mathrm{ar}\left[Y_{\alpha_1} \mid \boldsymbol{e}, \boldsymbol{L}_\alpha\right] \leq \sum_{w} d^*_{\boldsymbol{e}} \cdot \left(2^{2z-2\ell-2} \cdot m^{\rho_{(\ell+1)+}} \cdot (\#H \mid \boldsymbol{L}_{\beta_1}) + m^{\rho_{(\ell+1)+}} \cdot (\#H \mid \boldsymbol{L}_{\beta_1})\right).$$

We now in turn plug this in Eq (9),

$$\begin{aligned}&\mathbb{E}\left[\mathbb{V}\mathrm{ar}\left[Y_\alpha \mid \boldsymbol{e}\right] \mid \boldsymbol{L}_\alpha\right] \\ &\leq m^k \sum_{\boldsymbol{e}} \frac{1}{t_{\boldsymbol{e}}} \sum_{w} d^*_{\boldsymbol{e}} \cdot \left(2^{2z-2\ell-2} \cdot m^{\rho_{(\ell+1)+}} \cdot (\#H \mid \boldsymbol{L}_{\beta_1}) + m^{\rho_{(\ell+1)+}} \cdot (\#H \mid \boldsymbol{L}_{\beta_1})\right) \\ &\leq m^k \sqrt{m} \cdot \sum_{\boldsymbol{e}} \sum_{w} 2^{2z-2\ell-1} \cdot m^{\rho_{(\ell+1)+}} \cdot (\#H \mid \boldsymbol{L}_{\beta_1}) \qquad (\text{as } t_{\boldsymbol{e}} \geq d^*_{\boldsymbol{e}}/\sqrt{m}) \\ &\leq 2^{2z-2\ell-1} \cdot m^{\rho_{\ell+}} \cdot \sum_{\boldsymbol{e}} \sum_{w} (\#H \mid \boldsymbol{L}_{\beta_1}) \\ &\qquad (\text{as } \rho^C_\ell = k + 1/2 \text{ and } \rho_{\ell+} = \rho^C_\ell + \rho_{(\ell+1)+} \text{ by definition}) \\ &= 2^{2z-2\ell-1} \cdot m^{\rho_{\ell+}} \cdot (\#H \mid \boldsymbol{L}_\alpha). \qquad (\text{as } \boldsymbol{L}_{\beta_1} = (\boldsymbol{L}_\alpha, \boldsymbol{e}, w))\end{aligned}$$

Finally, by plugging in this and Eq (8) in Eq (7),

$$\begin{aligned}\mathbb{V}\mathrm{ar}\left[Y_\alpha \mid \boldsymbol{L}_\alpha\right] &= 2^{2z-2\ell-1} \cdot m^{\rho_{\ell+}} \cdot (\#H \mid \boldsymbol{L}_\alpha) + m^{\rho_{\ell+}} \cdot (\#H \mid \boldsymbol{L}_\alpha) \\ &\leq 2^{2z-2\ell} \cdot m^{\rho_{\ell+}} \cdot (\#H \mid \boldsymbol{L}_\alpha),\end{aligned}$$

finalizing the proof of induction step in this case. We again remark that this proof closely followed the proof for the variance of the estimator for cycle-sampler tree in Lemma 7.

Case of $\ell > o$. In this case, the sub-tree $\mathcal{T}_\alpha$ in the next two levels is a star-sampler tree. By the same analogy made in the proof of the previous case and Lemma 7, the proof of this part also follows the proof of Lemma 9 for star-sampler trees. We hence omit the details.

To conclude, we have that $\mathbb{V}\mathrm{ar}[Y] = \mathbb{V}\mathrm{ar}[Y_{\alpha_r} \mid \boldsymbol{L}_{\alpha_r}] = O(m^{\rho(H)}) \cdot (\#H) = O(m^{\rho(H)}) \cdot \mathbb{E}[Y]$ as $Y = Y_{\alpha_r}$ for the root α_r of $\mathcal{T}$, $\boldsymbol{L}_{\alpha_r} = \emptyset$, $(\#H) = \mathbb{E}[Y]$ by Lemma 11, and $z = O(1)$. ◀

4.2 An Algorithm for Estimating Occurrences of Arbitrary Subgraphs

We now use our estimator algorithm from the previous section to design our algorithm for estimating the occurrences of an arbitrary subgraph H in G. In the following theorem, we assume that the algorithm has knowledge of m and also a lower bound on the value of $\#H$; these assumptions can be lifted easily as we describe afterwards.

▶ **Theorem 13.** *There exists a sublinear time algorithm that uses degree, neighbor, pair, and edge sample queries and given a precision parameter $\varepsilon \in (0,1)$, an explicit access to a constant-size graph $H(V_H, E_H)$, a query access to the input graph $G(V,E)$, the number of edges m in G, and a lower bound $h \leq \#H$, with high probability outputs a $(1 \pm \varepsilon)$-approximation to $\#H$ using $O\Big(\min\Big\{m, \frac{m^{\rho(H)}}{h} \cdot \frac{\log n}{\varepsilon^2}\Big\}\Big)$ queries and $O\Big(\frac{m^{\rho(H)}}{h} \cdot \frac{\log n}{\varepsilon^2}\Big)$ time, in the worst-case.*

Proof. Fix a sufficiently large constant $c > 0$. We run `subgraph-sampler`(G,H) for $k := \frac{c \cdot m^{\rho(H)}}{\varepsilon^2 \cdot h}$ time independently in parallel to obtain estimates $Y_1, \ldots, Y_k$ and let $Z := \frac{1}{k}\sum_{i=1}^{k} Y_i$. By Lemma 11, $\mathbb{E}[Z] = (\#H)$. Since Y_i's are independent, we also have

$$\mathbb{V}\mathrm{ar}[Z] = \frac{1}{k^2}\sum_{i=1}^{k} \mathbb{V}\mathrm{ar}[Y_i] \leq \frac{1}{k} \cdot O(m^{\rho(H)}) \cdot \mathbb{E}[Z] \leq \frac{\varepsilon^2}{10} \cdot \mathbb{E}[Z]^2,$$

by Lemma 12, and by choosing the constant c sufficiently larger than the constant in the O-notation of this lemma, together with the fact that $h \leq (\#H) = \mathbb{E}[Z]$. By Chebyshev's inequality (Proposition 1), $\Pr(|Z - \mathbb{E}[Z]| \geq \varepsilon \cdot \mathbb{E}[Z]) \leq \frac{\mathbb{V}\mathrm{ar}[Z]}{\varepsilon^2 \cdot \mathbb{E}[Z]^2} \leq \frac{1}{10}$, by the bound above on the variance. This means that with probability 0.9, this algorithm outputs a $(1 \pm \varepsilon)$-approximation of $\#H$. Moreover, the expected query complexity and running time of this algorithm is $O(k)$ by Lemma 10, which is $O(\frac{m^{\rho(H)}}{\varepsilon^2})$ (if $k \geq m$, we simply query all edges of the graph and solve the problem using an offline enumeration algorithm). To extend this result to a high probability bound and also making the guarantee of query complexity and run-time in the worst-case, we simply run this algorithm $O(\log n)$ times in parallel and stop each execution that uses more than 10 times queries than the expectation. ◀

The algorithm in Theorem 13 assumes the knowledge of h which is a lower bound on $(\#H)$. However, this assumption can be easily removed by making a geometric search on h starting from $m^{\rho(H)}/2$ which is (approximately) the largest value for $(\#H)$ all the way down to 1 in factors of 2, and stopping the search once the estimates returned for a guess of h became consistent with h itself. This only increases the query complexity and runtime of the algorithm by polylog(n) factors. As this part is quite standard, we omit the details and instead refer the interested reader to [16, 18]. This concludes the proof of our main result in Theorem 1 from the introduction.

4.3 Extension to the Database Join Size Estimation Problem

The database join size estimation for binary relations can be modeled by the subgraph estimation problem where the subgraph H and the underlying graph G are additionally *edge-colored* and we are only interested in counting the copies of H in G with matching colors on the edges. In this abstraction, the edges of the graph G correspond to the entries of the database, and the color of edges determine the relation of the entry.

We formalize this variant of the subgraph counting problem in the following. In the *colorful* subgraph estimation problem, we are given a subgraph $H(V_H, E_H)$ with a coloring function $c_H : E_H \to \mathbb{N}$ and query access to a graph $G(V, E)$ along with a coloring function $c_G : E \to \mathbb{N}$. The set of allowed queries to G contains the degree queries, pair queries, neighbor queries, and edge-sample queries as before, with a simple change that whenever we query an edge (through the last three types of queries), the color of the edge according to c_G is also revealed to the algorithm. Our goal is to estimate the number of copies of H in G with matching colors, i.e., the *colorful* copies of H.

It is immediate to verify that our algorithm in this section can be directly applied to the colorful subgraph estimation problem with the only difference that when testing whether a subgraph forms a copy of H in G, we in fact check whether this subgraph forms a colorful copy of H in G instead. The analysis of this new algorithm is exactly as in the case of the original algorithm with the only difference that we switch the parameter $\#H$ to $\#H_c$ that only counts the number of copies of H with the same colors in G. To summarize, we obtain an algorithm with $O^*(\frac{m^{\rho(H)}}{\#H_c})$ query and time complexity for the colorful subgraph counting problem, which can in turn solves the database join size estimation problem for binary relations.

5 Lower Bounds

We present two lower bounds that demonstrate the optimality of Theorem 1 in different scenarios. Our first lower bound establishes tight bounds for counting *odd cycles*.

▶ **Theorem 14.** *For any $k \geq 1$, any algorithm $\mathcal{A}$ that can output any multiplicative-approximation to the number of copies of the odd cycle C_{2k+1} in a given graph $G(V, E)$ with probability at least $2/3$ requires $\Omega(\frac{m^{k+\frac{1}{2}}}{\#C_{2k+1}})$ queries to G.*

Theorem 14 implies that in addition to cliques (that were previously proved [19]; see also [16, 18]), our algorithm in Theorem 1 also achieve optimal bounds for odd cycles.

Our next lower bound targets the more general problem of database join size estimation for which we argued that our Theorem 1 continues to hold. We show that for this more general problem, our algorithm in Theorem 1 is in fact optimal *for all* choices of subgraph H.

▶ **Theorem 15.** *For any subgraph $H(V_H, E_H)$ which contains at least one edge, suppose $\mathcal{A}$ is an algorithm for the colorful subgraph estimation problem that given H, a coloring $c_H : E_H \to \mathbb{N}$, and query access to $G(V, E)$ with m edges and coloring function $c_G : E \to \mathbb{N}$, can output a multiplicative-approximation to the number of colorful copies of H in G with probability at least $2/3$. Then, $\mathcal{A}$ requires $\Omega(\frac{m^{\rho(H)}}{\#H_c})$ queries, where $\#H_c$ is the number of colorful copies of H in G. The lower bound continues to hold even if the number of colors used by c_H and c_G is at most two.*

The proofs of Theorems 14 and 15 are postponed to the full version of the paper [4].

Acknowledgements

We are thankful to the anonymous reviewers of ITCS 2019 for many valuable comments.

References

1 Nesreen K. Ahmed, Jennifer Neville, and Ramana Rao Kompella. Network sampling: From static to streaming graphs. *TKDD*, 8(2):7:1–7:56, 2013.

2 Maryam Aliakbarpour, Amartya Shankha Biswas, Themis Gouleakis, John Peebles, Ronitt Rubinfeld, and Anak Yodpinyanee. Sublinear-time algorithms for counting star subgraphs via edge sampling. *Algorithmica*, 80(2):668–697, 2018.

3 Noga Alon. On the number of subgraphs of prescribed type of graphs with a given number of edges. *Israel Journal of Mathematics*, 1981.

4 Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. *arXiv*, abs/1811.07780, 2018. URL: https://arxiv.org/abs/1811.07780.

5 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 739–748. IEEE Computer Society, 2008. URL: https://doi.org/10.1109/FOCS.2008.43, doi:10.1109/FOCS.2008.43.

6 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 623–632, 2002.

7 Suman K. Bera and Amit Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 11:1–11:14, 2017.

8 E. Bloedorn, N. Rothleder, D. DeBarr, and L. Rosen. Relational Graph Analysis with Real-World Constraints: An Application in IRS Tax Fraud Detection. In *AAAI*, 2005.

9 Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 244–254, 2013.

10 Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 253–262, 2006.

11 S. Burt. Structural Holes and Good Ideas. *The American Journal of Sociology*, 110(2):349–399, 2004. URL: http://dx.doi.org/10.2307/3568221, doi:10.2307/3568221.

12 Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM J. Comput.*, 34(6):1370–1379, 2005.

13 Graham Cormode and Hossein Jowhari. A second look at counting triangles in graph streams (corrected). *Theor. Comput. Sci.*, 683:22–30, 2017.

14 Artur Czumaj, Funda Ergün, Lance Fortnow, Avner Magen, Ilan Newman, Ronitt Rubinfeld, and Christian Sohler. Approximating the weight of the euclidean minimum spanning tree in sublinear time. *SIAM J. Comput.*, 35(1):91–109, 2005.

15 Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear-time. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 175–183, 2004.

16 Talya Eden, Amit Levi, Dana Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 614–633, 2015.

17 Talya Eden, Dana Ron, and C. Seshadhri. Sublinear time estimation of degree distribution moments: The degeneracy connection. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 7:1–7:13, 2017.

18 Talya Eden, Dana Ron, and C. Seshadhri. On approximating the number of k-cliques in sublinear time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 722–734, 2018.

19 Talya Eden and Will Rosenbaum. Lower bounds for approximating graph parameters via communication complexity. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, pages 11:1–11:18, 2018.

20 Talya Eden and Will Rosenbaum. On sampling edges almost uniformly. In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, pages 7:1–7:9, 2018.

21 Uriel Feige. On sums of independent random variables with unbounded variance, and estimating the average degree in a graph. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 594–603, 2004.

22 Ehud Friedgut and Jeff Kahn. On the number of copies of one hypergraph in another. *Israel Journal of Mathematics*, 1998.

23 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.

24 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random Struct. Algorithms*, 32(4):473–493, 2008.

25 Mira Gonen, Dana Ron, and Yuval Shavitt. Counting stars and other small subgraphs in sublinear time. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 99–116, 2010.

26 Avinatan Hassidim, Jonathan A. Kelner, Huy N. Nguyen, and Krzysztof Onak. Local graph partitions for approximation and testing. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 22–31, 2009.

27 Madhav Jha, C. Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 589–597, 2013.

28 Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, pages 710–716, 2005.

29 John Kallaugher, Michael Kapralov, and Eric Price. The sketching complexity of graph and hypergraph counting. *CoRR*, abs/1808.04995. To appear in FOCS 2018., 2018.

30 John Kallaugher and Eric Price. A hybrid sampling scheme for triangle counting. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1778–1797, 2017.

31 Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, pages 598–609, 2012.

32 Tali Kaufman, Michael Krivelevich, and Dana Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM J. Comput.*, 33(6):1441–1483, 2004.

33 Ju-Sung Lee and Jürgen Pfeffer. Estimating centrality statistics for complete and sampled networks: Some approaches and complications. In *48th Hawaii International Conference on System Sciences, HICSS 2015, Kauai, Hawaii, USA, January 5-8, 2015*, pages 1686–1695, 2015.

34 Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 631–636, 2006.

35 Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 401–411, 2016.

36 R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, October 2002.

37 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3):16:1–16:40, 2018.

38 Huy N. Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 327–336, 2008.

39 Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1123–1131, 2012.

40 Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theor. Comput. Sci.*, 381(1-3):183–196, 2007.

41 Olivia Simpson, C. Seshadhri, and Andrew McGregor. Catching the head, tail, and everything in between: A streaming algorithm for the degree distribution. In *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015*, pages 979–984, 2015.

42 Johan Ugander, Lars Backstrom, and Jon Kleinberg. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 1307–1318, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee. URL: http://dl.acm.org/citation.cfm?id=2488388.2488502.

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