Sublinear-Time Algorithms for Counting Star Subgraphs via Edge Sampling

Noname manuscript No.
(will be inserted by the editor)

Sublinear-Time Algorithms for Counting Star Subgraphs via Edge Sampling

Maryam Aliakbarpour · Amartya Shankha Biswas · Themistoklis Gouleakis · John Peebles · Ronitt Rubinfeld · Anak Yodpinyanee

Received: date / Accepted: date

Abstract We study the problem of estimating the value of sums of the form $S_p \triangleq \sum \binom{x_i}{p}$ when one has the ability to sample $x_i \geq 0$ with probability proportional to its magnitude. When $p = 2$, this problem is equivalent to estimating the selectivity of a self-join query in database systems when one can sample rows randomly. We also study the special case when $\{x_i\}$ is the degree sequence of a graph, which corresponds to counting the number of p-stars in a graph when one has the ability to sample edges randomly.

Our algorithm for a $(1 \pm \varepsilon)$-multiplicative approximation of S_p has query and time complexities $\mathrm{O}(\frac{m \log\log n}{\epsilon^2 S_p^{1/p}})$. Here, $m = \sum x_i/2$ is the number of edges in the graph, or equivalently, half the number of records in the database table. Similarly, n is the number of vertices in the graph and the number of unique values in the database table. We also provide tight lower bounds (up to poly-

Maryam Aliakbarpour
CSAIL, MIT, Cambridge MA 02139
E-mail: maryama@mit.edu

Amartya Shankha Biswas
MIT, Cambridge MA 02139
E-mail: asbiswas@mit.edu

Themistoklis Gouleakis
CSAIL, MIT, Cambridge MA 02139
E-mail: tgoule@mit.edu

John Peebles
CSAIL, MIT, Cambridge MA 02139
E-mail: johnpeeb@gmail.com

Ronitt Rubinfeld
CSAIL, MIT, Cambridge MA 02139 and the Blavatnik School of Computer Science, Tel Aviv University
E-mail: ronitt@csail.mit.edu

Anak Yodpinyanee
CSAIL, MIT, Cambridge MA 02139
E-mail: anak@csail.mit.edu

logarithmic factors) in almost all cases, even when $\{x_i\}$ is a degree sequence and one is allowed to use the structure of the graph to try to get a better estimate. We are not aware of any prior lower bounds on the problem of join selectivity estimation.

For the graph problem, prior work which assumed the ability to sample only *vertices* uniformly gave algorithms with matching lower bounds [Gonen, Ron, and Shavitt. *SIAM J. Comput.*, 25 (2011), pp. 1365-1411]. With the ability to sample edges randomly, we show that one can achieve faster algorithms for approximating the number of star subgraphs, bypassing the lower bounds in this prior work. For example, in the regime where $S_p \leq n$, and $p = 2$, our upper bound is $\tilde{O}(n/S_p^{1/2})$, in contrast to their $\Omega(n/S_p^{1/3})$ lower bound when no random edge queries are available.

In addition, we consider the problem of counting the number of directed paths of length two when the graph is directed. This problem is equivalent to estimating the selectivity of a join query between two distinct tables. We prove that the general version of this problem cannot be solved in sublinear time. However, when the ratio between in-degree and out-degree is bounded—or equivalently, when the ratio between the number of occurrences of values in the two columns being joined is bounded—we give a sublinear time algorithm via a reduction to the undirected case.

Keywords subgraphs, approximate counting, randomized algorithms, sublinear-time algorithms

1 Introduction

We study the problem of approximately estimating $S_p \triangleq \sum_{i=1}^{n} \binom{x_i}{p}$ when one has the ability to sample $x_i \geq 0$ with probability proportional to its magnitude. To solve this problem we design *sublinear-time algorithms* (as long as S_p is sufficiently large), which compute such an approximation while only looking at an extremely tiny fraction of the input, rather than having to scan the entire data set in order to determine this value.

We consider two primary motivations for this problem. The first is that in undirected graphs, if x_i is the degree of vertex i then S_p counts the number of p-stars in the graph. Thus, estimating S_p when one has the ability to sample x_i with probability proportional to its magnitude corresponds to estimating the number of p-stars when one has the ability to sample vertices with probability proportional to their degrees (which is equivalent to having the ability to sample edges uniformly). This problem is an instance of the more general *subgraph counting problem* in which one wishes to estimate the number of occurrences of a subgraph H in a graph G. The subgraph counting problem has applications in many different fields, including the study of biological, internet and database systems. For example, detecting and counting subgraphs in protein interaction networks is used to study molecular pathways and cellular processes across species [48].

The second application of interest is that the problem of estimating S_2 corresponds to estimating the selectivity of join and self-join operations in databases when one has the ability to sample rows of the tables uniformly. For example, note that if we set x_i as the number of occurrences of value i in the column being joined, then S_2 is precisely the number of records in the join of the table with itself on that column. When performing a query in a database, a program called a *query optimizer* is used to determine the most efficient way of performing the database query. In order to make this determination, it is useful for the query optimizer to know basic statistics about the database and about the query being performed. For example, queries that return a larger number of records are usually serviced most efficiently by doing simple linear scans over the data, whereas queries that return a smaller number of records may be better serviced by using an index [27]. As such, being able to estimate *selectivity* (number of records returned compared to the maximum possible number) of a query can be useful information for a query optimizer to have. In the more general case of estimating the selectivity of a join between two different tables (which can be modeled with a directed graph), the query optimizer can use this information to decide on the most efficient order to execute a sequence of joins which is a common task.

In the "typical" regime in which we wish to estimate S_2 given that $n \leq S_2 \leq n^2$, our algorithm has a running time of $\tilde{O}(\sqrt{n})$ which is very small compared to the total amount of data. Furthermore, in the case of selectivity estimation, this number can be much less than the number of distinct values in the column being joined on, which results in an even smaller number of queries than would be necessary if one were using an index to compute the selectivity.

We believe that our query-based framework can be realized in many systems. One possible way to implement random edge queries is as follows: 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. Random edge queries allow us to implement a source of *weighted vertex samples*, where a vertex is output with probability proportional to its weight (magnitude). Weighted sampling is used in [41,9] to find sublinear algorithms for approximating the sum of n numbers (allowing only uniform sampling, results in a linear lower bound). We later use this as a subroutine in our algorithm.

Throughout the rest of the paper, we will mostly use graph terminology when discussing this problem: our goal is to estimate S_p on simple graphs for any constant integer $p \geq 2$. However, we emphasize that all our results are fully general and apply to the problem of estimating S_p even when one does not assume that the input is a graph.

1.1 Our Contribution

Prior theoretical work on this problem only considered the version of this problem on graphs and assumed the ability to sample vertices uniformly rather than edges. Specifically, prior studies of sublinear-time algorithms for graph problems usually consider the model where the algorithm is allowed to query the adjacency list representation of the graph: it may make *neighbor queries* (by asking "what is the i^{th} neighbor of a vertex v") and *degree queries* (by asking "what is the degree of vertex v").

We propose a stronger model of sublinear-time algorithms for graph problems which allows random edge queries. Next, for undirected graphs, we construct an algorithm which uses only degree queries and random edge queries. This algorithm and its analysis is discussed in Section 3. For the problem of computing an approximation $\hat{S}_p$ satisfying $(1-\epsilon)S_p \leq \hat{S}_p \leq (1+\epsilon)S_p$, our algorithm has query and time complexities $\mathrm{O}(m \log\log n \,/\, \epsilon^2 S_p^{1/p})$. Although our algorithm is described in terms of graphs, it also applies to the more general case when one wants to estimate $S_p = \sum_i \binom{x_i}{p}$ without any assumptions about graph structure. Thus, it also applies to the problem of self-join selectivity estimation.

We then establish some relationships between m and other parameters so that we may compare the performance of this algorithm to a related work by Gonen et al. more directly ([24]). We also provide lower bounds for our proposed model in Section 4, which are mostly tight up to polylogarithmic factors. This comparison is given in Table 1. We emphasize that even though these lower bounds are stated for graphs, they also apply to the problem of self-join selectivity estimation.

To understand this table, first note that these algorithms require more samples when S_p is small (i.e., stars are rare). As S_p increases, the complexity of each algorithm decreases until—at some point—the number of required samples drops to $\tilde{\mathrm{O}}(n^{1-1/p})$. Our algorithm is able to obtain this better complexity of $\tilde{\mathrm{O}}(n^{1-1/p})$ for a larger range of values of S_p than that of the algorithm given in [24]. Specifically, our algorithm is more efficient for $S_p \leq n^{1+1/p}$, and has the same asymptotic bound for S_p up to n^p. Once $S_p > n^p$, it is unknown whether the degree and random edge queries alone can provide the same query complexity. Nonetheless, if we have access to all three types of queries, we may combine the two algorithms to obtain the best of both cases as illustrated in the last column. Additionally, due to the simplicity of our algorithm, we remark that the dependence on ϵ of our query complexity is only $1/\epsilon^2$ when m is known in advance (or still only $1/\epsilon^3$ otherwise) for any value of S_p, while that of their algorithm is as large as $1/\epsilon^{10}$ in certain cases. This dependence on ϵ may be of interest to some applications, especially when stars are rare whilst an accurate approximation of S_p is crucial.

We also consider a variant of the counting stars problem on directed graphs in Section 5. If one only needs to count "stars" where all edges are either pointing into or away from the center, this is essentially still the undirected case. We then consider counting directed paths of length two, and discover

Table 1: Summary of the query and time complexities for counting p-stars on undirected graphs, given a different set of allowed queries. ϵ is assumed to be constant in the table; however, the dependence on ϵ of our algorithm (for degree and random edge queries) is always $1/\epsilon^2$. Adjacent cells in the same column with the same contents have been merged.

range of S_p	permitted types of queries		
	neighbor, degree ([24])	degree, random edge (this paper)	all types of queries (this paper)
$S_p \leq n$	$\widetilde{\Theta}\left(\frac{n}{S_p^{1/(p+1)}}\right)$	$\widetilde{\Theta}\left(\frac{n}{S_p^{1/p}}\right)$	$\widetilde{\Theta}\left(\frac{n}{S_p^{1/p}}\right)$
$n < S_p \leq n^{1+1/p}$		$\widetilde{\Theta}\left(n^{1-1/p}\right)$	$\widetilde{\Theta}\left(n^{1-1/p}\right)$
$n^{1+1/p} < S_p \leq n^p$	$\widetilde{\Theta}\left(n^{1-1/p}\right)$		
$n^p < S_p$	$\widetilde{\Theta}\left(\frac{n^{p-1/p}}{S_p^{1-1/p}}\right)$	$\Omega\left(\frac{n^{p-1/p}}{S_p^{1-1/p}}\right), \widetilde{\mathrm{O}}\left(n^{1-1/p}\right)$	$\widetilde{\Theta}\left(\frac{n^{p-1/p}}{S_p^{1-1/p}}\right)$

that allowing random edge queries does not provide an efficient algorithm in this case. In particular, we show that any constant factor multiplicative approximation of S_p requires $\Omega(n)$ queries even when all three types of queries are allowed. However, when the ratio between the in-degree and the out-degree on every vertex v (denoted $\deg^-(v)$ and $\deg^+(v)$, respectively) is bounded, we solve this special case in sublinear time via a reduction to the undirected case where degree queries and random edge queries are allowed.

This variant of the counting stars problem can also be used for approximating join selectivity. For a directed graph, we aim at estimating the quantity $\sum_{v \in V(G)} \deg^-(v) \cdot \deg^+(v)$. On the other hand in the database context, we wish to compute the quantity $\sum_{i=1}^{n} x_i \cdot y_i$, where x_i and y_i denote the number of occurrences of a label i in the column we join on, from the first and the second table, respectively. Thus, applying simple changes in variables, the algorithms from Section 5 can be applied to the problem of estimating join selectivity as well.

1.2 Our Approaches

In order to approximate the number of stars in the undirected case, we convert the random edge queries into weighted vertex sampling, where the probability of sampling a particular vertex is proportional to its degree. We then construct an unbiased estimator that approximates the number of stars using the degree of the sampled vertex as a parameter. The analysis of this part is roughly based on the variance bounding method used in [5], which aims to approximate the frequency moment in a streaming model. The number of samples required by this algorithm depends on S_p, which is not known in advance. Thus we create a guessed value of S_p and iteratively update this parameter until it becomes accurate.

To demonstrate lower bounds in the undirected case, we construct new instances to prove tight bounds for the case in which our model is more powerful

than the traditional model. In other cases, we provide a new proof to show that the ability to sample uniformly random edges does not necessarily allow better performance in counting stars. Our proof is based on applying Yao's principle and providing an explicit construction of the hard instances, which unifies multiple cases together and greatly simplifies the approach of [24].[1]

For the directed case, we prove the lower bound using a standard construction and Yao's principle. As for the upper bound when the in-degree and out-degree ratios are bounded, we use rejection sampling to adjust the sampling probabilities so that we may apply the unbiased estimator method from the undirected case.

1.3 Related Work

Motivated by applications in a variety of areas, the subgraph detection and counting problem and its variations have been studied in many different works, often under different terminology such as network motif counting or pathway querying (e.g., [40,47,51,48,49,26,31,29,2]). As this problem is NP-hard in general, many approaches have been developed to efficiently count subgraphs more efficiently for certain families of subgraphs or input graphs (e.g., [16,6, 19,34,2,4,53,52,25,35,1,7,20]). As for applications to database systems, the problem of approximating the size of the resulting table of a join query or a self-join query in various contexts has been studied in [50,28,3]. Selectivity and query optimization have been considered, e.g., in [46,36,21,38,27].

Other works that study sublinear-time algorithms for counting stars are [24] that aims to approximate the number of stars, and [18,23] that aim to approximate the number of edges (or equivalently, the average degree). Note that [24] also shows impossibility results for approximating triangles and paths of length three in sublinear time when given uniform edge sampling, limiting us from studying more sophisticated subgraphs. Recent work by Eden, Levi, Ron and Seshadhri ([17]) provides sublinear time algorithms to approximate the number of triangles in a graph. However, their model uses adjacency matrix queries and neighbor queries. The problem of counting subgraphs has also been studied in the streaming model (e.g., [8,13,10,37,33,39]). There is also

[1] One useful technique for giving lower bounds on sublinear time algorithms, pioneered by [12], is to make use of a connection between lower bounds in communication complexity and lower bounds on sublinear time algorithms. More specifically, by giving a reduction from a communication complexity problem to the problem we want to solve, a lower bound on the communication complexity problem yields a lower bound on our problem. In the past, this approach has led to simpler and cleaner sublinear time lower bounds for many problems. Attempts at such an approach for reducing the set-disjointness problem in communication complexity to our estimation problem on graphs run into the following difficulties: First, as explained in [22], the straightforward reduction adds a logarithmic overhead, thereby weakening the lower bound by the same factor. Second, the reduction seems to work only in the case of sparse graphs. Although it is not clear if these difficulties are insurmountable, it seems that it will not give a simpler argument than the approach that we present in this work.

a body of work on sublinear-time algorithms for approximating various graph parameters (e.g., [45,43,54,30,44]).

Abstracting away the graphical context of counting stars, we may view our problem as finding a parameter of a distribution: edge or vertex sampling can be treated as sampling according to some distribution. In vertex sampling, we have a uniform distribution and in edge sampling, the probabilities are proportional to the degree. The number of stars can be written as a function of the degrees. Aside from our work, there are a number of other studies that make use of combined query types for estimating a parameter of a distribution. Weighted and uniform sampling are considered in [41,9]. Their algorithms may be adapted to approximate the number of edges in the context of approximating graph parameters when given weighted vertex sampling, which we will later use in this paper. A closely related problem in the context of distributions, is the task of approximating frequency moments, mainly studied in the streaming model (e.g., [5,15,32,11]). On the other hand, the combination of weighted sampling and probability distribution queries is also considered (e.g., [14]).

2 Preliminaries

In this paper, we construct algorithms to approximate the number of stars in a graph under different types of query access to the input graph. As we focus on the case of simple undirected graphs, we explain this model here and defer the description for the directed case to Section 5.

2.1 Graph Specification

Let $G = (V, E)$ be the input graph, assumed to be simple and undirected. Let n and m denote the number of vertices and edges, respectively. The value n is known to the algorithm. Each vertex $v \in V$ is associated with a unique ID from $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. Let $\deg(v)$ denote the degree of v.

Let $p \geq 2$ be a constant integer. A *p-star* is a subgraph of size $p+1$, where one vertex, called the *center*, is adjacent to the other p vertices. For example, a 2-star is an undirected path of length 2. Note that a vertex may be a center for many stars, and a set of $p+1$ vertices may form multiple stars. Let S_p denote the number of occurrences of distinct stars in the graph.

Our goal is to construct a randomized algorithm that outputs a value that is within a $(1 \pm \epsilon)$-multiplicative factor of the actual number of stars S_p. More specifically, given a parameter $\epsilon > 0$, the algorithm must give an approximated value $\widehat{S}_p$ satisfying the inequality $(1-\epsilon)S_p \leq \widehat{S}_p \leq (1+\epsilon)S_p$ with success probability at least $2/3$.

2.2 Query Access

The algorithm may access the input graph by querying the *graph oracle*, which answers for the following types of queries. First, the *neighbor queries*: given a vertex $v \in V$ and an index $1 \leq i < n$, the i^{th} neighbor of v is returned if $i \leq \deg(v)$; otherwise, $\perp$ is returned. Second, the *degree queries*: given a vertex $v \in V$, its degree $\deg(v)$ is returned. Lastly, the *random edge queries*: a uniformly random edge $\{u, v\} \in E$ is returned. The *query complexity* of an algorithm is the total number of queries of any type that the algorithm makes throughout the process of computing its answer.

Combining these queries, we may implement various useful sampling processes. We may perform a *uniform edge sampling* using a random edge query, and a *uniform vertex sampling* by simply picking a random index from $[n]$. We may also perform a *weighted vertex sampling* where each vertex is obtained with probability proportional to its degree as follows: uniformly sample a random edge, then randomly choose one of the endpoints with probability 1/2 each. Since any vertex v is incident with $\deg(v)$ edges, then the probability that v is chosen is exactly $\deg(v)/2m$, as desired.

2.3 Queries in the Database Model

Now we explain how the above queries in our graph model have direct interpretations in the database model. Consider the column we wish to join on. For each valid label i, let x_i be the number of rows containing this label. We assume the ability to sample rows uniformly at random. This gives us a label i with probability proportional to x_i, which is a weighted sample from the distribution of labels. We also assume that we can also quickly compute the number of other rows sharing the same label with a given row (analogous to making a degree query). For example, this could be done quickly using an index on the column. Note that if one has an index that is augmented with appropriate information, one can compute the selectivity of a self-join query exactly in time roughly $O(k \log n)$ where k is the number of distinct elements in the column. However, our methods can give runtimes that are asymptotically much smaller than this.

3 Upper Bounds for Counting Stars in Undirected Graphs

In this section we establish an algorithm for approximating the number of stars, S_p, of an undirected input graph. We focus on the case where only degree queries and random edge queries are allowed. This illustrates that even without utilizing the underlying structure of the input graph, we are still able to construct a sublinear approximation algorithm that outperforms other algorithms under the traditional model in certain cases.

3.1 Unbiased Estimator Subroutine

Our algorithm uses weighted vertex sampling to find stars. Intuitively, the number of samples required by the algorithms should be larger when stars are rare because it takes more queries to find them. While the query complexity of the algorithm depends on the actual value of S_p, our algorithm does not know this value in advance. In order to overcome this issue, we devise a subroutine which—given a guess $\widetilde{S}_p$ for the value of S_p—will give a $(1\pm\epsilon)$ approximation of S_p if $\widetilde{S}_p$ is close enough to S_p or tell us that $\widetilde{S}_p$ is much larger than S_p. Then, we start with the maximum possible value of S_p and guess multiplicatively smaller and smaller values for it until we find one that is close enough to S_p, so that our subroutine is able to correctly output a $(1 \pm \epsilon)$ approximation.

Our subroutine works by computing the average value of an unbiased estimator to S_p after drawing enough weighted vertex samples. To construct the unbiased estimator, notice first that the number of p-stars centered at a vertex v is $\binom{\deg(v)}{p}$.[2] Thus, $S_p = \sum_{v\in V}\binom{\deg(v)}{p}$.

Next, we define the unbiased estimator and give the corresponding algorithm. First, let X be the random variable representing the degree of a random vertex obtained through weighted vertex sampling, as explained in Section 2.2. Recall that a vertex v is sampled with probability $\deg(v)/2m$. We define the random variable $Y = \frac{2m}{X}\binom{X}{p}$ so that Y is an unbiased estimator for S_p; that is,

$$E[Y] = \sum_{v\in V}\frac{\deg(v)}{2m}\left(\frac{2m}{\deg(v)}\binom{\deg(v)}{p}\right) = \sum_{v\in V}\binom{\deg(v)}{p} = S_p.$$

Algorithm 1 Subroutine for Computing S_p given $\widetilde{S}_p$ with success probability 2/3

```
1: procedure UNBIASED-ESTIMATE(S̃_p, ε)
2:     k ← 36m / ε² S̃_p^{1/p}
3:     for i = 1 to k do
4:         v ← weighted sampled vertex obtained from a random edge query
5:         d ← deg(v) obtained from a degree query
6:         Y_i ← (2m/d) C(d, p)
7:     Ȳ ← (1/k) Σ_{i=1}^{k} Y_i
8:     return Ȳ
```

Clearly, the output $\bar{Y}$ of Algorithm 1 satisfies $E[\bar{Y}] = S_p$. We claim that the number of samples k in Algorithm 1 is sufficient to provide two desired properties: the algorithm returns an $(1\pm\epsilon)$-approximation of S_p if $\widetilde{S}_p$ is in the correct range; or, if $\widetilde{S}_p$ is too large, the anomaly will be evident as the output $\bar{Y}$ will be much smaller than $\widetilde{S}_p$. In particular, we may distinguish between

[2] For our counting purpose, if $x < y$ then we define $\binom{x}{y} = 0$.

these two cases by comparing $\bar{Y}$ against $(1-\epsilon)\tilde{S}_p$, as specified through the following lemma.

Lemma 1 *For $0 < \epsilon \leq 1/2$, with probability at least 2/3:*

1. *If $\frac{1}{2}S_p \leq \widetilde{S}_p \leq 6S_p$, then Algorithm 1 outputs $\bar{Y}$ such that $(1-\epsilon)S_p \leq \bar{Y} \leq (1+\epsilon)S_p$;*
   moreover, if $S_p > \tilde{S}_p$ then $\bar{Y} \geq (1-\epsilon)\widetilde{S}_p$.
2. *If $\widetilde{S}_p > 6S_p$, then Algorithm 1 outputs $\bar{Y}$ such that $\bar{Y} < \frac{1}{2}\widetilde{S}_p \leq (1-\epsilon)\widetilde{S}_p$.*

Proof: Let us first consider the first item. Since $\mathrm{Var}[Y] \leq \mathrm{E}[Y^2]$, we will focus on establishing an upper bound of $\mathrm{E}[Y^2]$. We compute

$$\mathrm{E}[Y^2] = \sum_{v\in V} \frac{\deg(v)}{2m}\left(\frac{2m}{\deg(v)}\binom{\deg(v)}{p}\right)^2 = 2m\sum_{v\in V}\frac{1}{\deg(v)}\binom{\deg(v)}{p}^2$$
$$\leq 2m\sum_{v\in V}\binom{\deg(v)}{p}^{2-1/p} \leq 2m\left(\sum_{v\in V}\binom{\deg(v)}{p}\right)^{2-1/p} = 2mS_p^{2-1/p},$$

where the first inequality holds because $(\deg(v))^p \geq \binom{\deg(v)}{p}$. Rearranging the terms, we have the following relationship:

$$\frac{\mathrm{E}[Y^2]}{S_p^2} \leq \frac{2m}{S_p^{1/p}}.$$

Now let us consider our average $\bar{Y}$. Since Y_i are identically distributed, we have

$$\mathrm{Var}[\bar{Y}] = \mathrm{Var}\left[\frac{1}{k}\sum_{i=1}^{k} Y_i\right] = \frac{1}{k^2}\mathrm{Var}\left[\sum_{i=1}^{k} Y_i\right] = \frac{1}{k}\mathrm{Var}[Y] \leq \frac{1}{k}\mathrm{E}[Y^2].$$

By Chebyshev's inequality (Theorem 10), we have

$$\Pr[|\bar{Y} - \mathrm{E}[\bar{Y}]| \geq \epsilon S_p] \leq \frac{\mathrm{Var}[\bar{Y}]}{\epsilon^2 S_p^2} \leq \frac{1}{k}\cdot\frac{2m}{\epsilon^2 S_p^{1/p}}.$$

In order to achieve the desired value $\bar{Y}$ such that $(1-\epsilon)S_p \leq \bar{Y} \leq (1+\epsilon)S_p$ with error probability $1/3$, it is sufficient to take $6m\,/\,\epsilon^2 S_p^{1/p}$ samples. Recall the assumption that $\widetilde{S}_p$ satisfying $\frac{1}{2}S_p \leq \widetilde{S}_p \leq 6S_p$. Thus, the number of required samples to achieve such bound with probability $1/3$ is

$$k = \frac{36m}{\epsilon^2 \widetilde{S}_p^{1/p}}.$$

For the second item, we apply Markov's Inequality (Theorem 11) to the given condition to obtain

$$\mathrm{P}\left[\overline{Y} \geq \frac{1}{2}\widetilde{S}_p\right] \leq \frac{\mathrm{E}[\overline{Y}]}{\frac{1}{2}\widetilde{S}_p} = \frac{S_p}{\frac{1}{2}\widetilde{S}_p} < \frac{\frac{1}{6}\widetilde{S}_p}{\frac{1}{2}\widetilde{S}_p} = \frac{1}{3},$$

implying the desired success probability.

Lastly, we substitute $\epsilon < 1/2$ to obtain the relationship between $\bar{Y}$ and $(1-\epsilon)\tilde{S}_p$, which establishes the condition for deciding whether the given $\tilde{S}_p$ is much larger than S_p, as desired. □

3.2 Full Algorithm

Our full algorithm proceeds by first setting $\widetilde{S}_p$ to $n\binom{n-1}{p}$, the maximum possible value of S_p given by the complete graph. We then use Algorithm 1 to check if $\widetilde{S}_p > 6S_p$; if this is the case, we reduce $\widetilde{S}_p$ then proceed to the next iteration. Otherwise, Algorithm 1 should already give an $(1 \pm \epsilon)$-approximation to S_p (with constant probability). We note that if $\epsilon > 1/2$, we may replace it with $1/2$ without increasing the asymptotic complexity.

Since the process above may take up to $\mathrm{O}(\log n)$ iterations, we must amplify the success probability of Algorithm 1 so that the overall success probability is still at least $2/3$. To do so, we simply make $\ell = \mathrm{O}(\log p + \log\log n)$ multiple calls to Algorithm 1 then take the median of the returned values. Our full algorithm can be described as Algorithm 2 below.

Algorithm 2 Algorithm for Approximating S_p

```
1: procedure COUNT-STARS(ε)
2:     S̃_p ← n·C(n−1, p), ℓ ← 80(log p + log log n)
3:     loop
4:         for i = 1 to ℓ do
5:             Z_i ← UNBIASED-ESTIMATE(S̃_p, ε)
6:         Z ← median{Z_1, ··· , Z_ℓ}
7:         if Z ≥ (1 − ε)S̃_p then
8:             Ŝ_p ← Z
9:             return Ŝ_p
10:        S̃_p ← S̃_p/2
```

Theorem 1 *Algorithm 2 outputs $\hat{S}_p$ such that $(1-\epsilon)S_p \le \hat{S}_p \le (1+\epsilon)S_p$ with probability at least $2/3$. The query complexity of Algorithm 2 is $\mathrm{O}\left(\frac{m \log\log n}{\epsilon^2 S_p^{1/p}}\right)$ for any constant integer $p \ge 2$.*

Proof: If we assume that the events from Lemma 1 hold, then the algorithm will take at most $\lceil \log\left(n\binom{n-1}{p}\right) \rceil \le (p+1)\log n$ iterations. By choosing $\ell = 80(\log p + \log\log n)$ and running Algorithm 1 independently ℓ times, the number of returned values of Algorithm 1 satisfying the desired property is stochastically lower-bounded by $\mathrm{B}(\ell, 2/3)$, the Binomial random variable with ℓ experiments and success probability $2/3$. The Chernoff bound (Theorem 12) implies that for $p \ge 2$,

$$\Pr\left[\mathrm{B}\left(\ell, \frac{2}{3}\right) < \frac{1}{2}\right] < e^{-\frac{1}{2}(\frac{1}{4})^2(\frac{2}{3})(80(\log p + \log\log n))} < \frac{2}{9p\log n} \le \frac{1}{3(p+1)\log n}.$$

That is, with probability $1-1/(3(p+1)\log n)$, more than half of the returned values of Algorithm 1 satisfy the conditions in Lemma 1, guaranteeing that the median Z also satisfies these conditions. By the union bound, the total failure probability is at most $1/3$. Now it is safe to assume that the events from the two lemmas hold. In case $\widetilde{S}_p > 6S_p$, our algorithm will detect this event because $Z < (1-\epsilon)\widetilde{S}_p$, implying that we never stop and return an inaccurate approximation. On the other hand, if $\widetilde{S}_p < S_p$, our algorithm computes $Z \geq (1-\epsilon)\widetilde{S}_p$ and must terminate. Since we only halve $\widetilde{S}_p$ on each iteration, when $\widetilde{S}_p < S_p$ first occurs, we have $\widetilde{S}_p \geq \frac{1}{2}S_p$. As a result, our algorithm must terminate with the desired approximation before the value $\widetilde{S}_p$ is halved again. Thus, Algorithm 2 returns $\hat{S}_p$ satisfying $(1-\epsilon)S_p \leq \hat{S}_p \leq (1+\epsilon)S_p$ with probability at least $2/3$, as desired.

Observe that the number of samples required by Algorithm 1 increases by a factor of $2^{1/p}$ when $\widetilde{S}_p$ is halved in each iteration. Thus the total number of queries required is less than a factor of $\sum_{i=0}^{\infty}(2^{-1/p})^i = \frac{1}{1-2^{-1/p}} = \Theta(p)$ (for $p \geq 2$) of the number of queries required by the last iteration. As $\widetilde{S}_p = \Theta(S_p)$ in the last iteration of Algorithm 2, our algorithm requires $\mathrm{O}(mp(\log p + \log\log n)\,/\,\epsilon^2 S_p^{1/p})$ samples, achieving the claimed query complexity for constant p. □

3.3 Removing the Dependence on m

As described above, Algorithm 1 picks the value k and defines the unbiased estimator based on m, the number of edges. Nonetheless, it is possible to remove this assumption of having prior knowledge of m by instead computing its approximation. Furthermore, we will bound m in terms of n and S_p, so that we can also relate the performance of our algorithm to previous studies on this problem such as [24], as done in Table 1.

3.3.1 Approximating m

We briefly discuss how to apply our algorithm when m is unknown by first computing an approximation of m. Using weighted vertex sampling, we may simulate the algorithm from [41] or [9] that computes an $(1\pm\epsilon)$-approximation to the sum of degrees using $\tilde{\mathrm{O}}(\sqrt{n}/\epsilon^3)$ weighted samples. More specifically, we cite the following theorem:

Theorem 2 *([9]) Let $x_1, \ldots, x_n$ be n variables, and define a distribution $\mathcal{D}$ that returns (i, x_i) with probability $x_i / \sum_{i=1}^{n} x_j$. There exists an algorithm that computes a $(1 \pm \epsilon)$-approximation of $S = \sum_{i=1}^{n} x_i$ using $\tilde{\mathrm{O}}(\sqrt{n}/\epsilon^3)$ samples from $\mathcal{D}$.*

Thus, we simulate the sampling process from $\mathcal{D}$ by drawing a weighted vertex sample v, querying its degree, and feeding $(v, \deg(v))$ to this algorithm. We will need to decrease ϵ used in this algorithm and our algorithm by a

constant factor to account for the additional error. Below we show that our complexities are at least $\tilde{\mathrm{O}}(n^{1-1/p}/\epsilon^2)$ which is already $\tilde{\mathrm{O}}(\sqrt{n}/\epsilon^2)$ for $p = 2$, and thus this extra step does not affect our algorithm's performance asymptotically in terms of n. Unfortunately, the dependence on ϵ of their algorithm is $\tilde{\mathrm{O}}(1/\epsilon^3)$, which carries over to our algorithm's complexities when m is not known in advance.

3.3.2 Comparing m to n and S_p

For comparison of performances, we will now show some bounds relating m to n and S_p. Notice that the function $\binom{\deg(v)}{p}$ is convex with respect to $\deg(v)$.[3] Then by applying Jensen's inequality (Theorem 13) to this function, we obtain

$$S_p = \sum_{v \in V} \binom{\deg(v)}{p} \geq n \binom{\sum_{v \in V} \deg(v)/n}{p} = n \binom{2m/n}{p}.$$

First, let us consider the case where the stars are very rare, namely when $S_p \leq n$. The inequality above implies that $m \leq np/2$. Substituting this formula back into the bound from Theorem 1 yields the query complexity $\tilde{\mathrm{O}}(n \,/\, \epsilon^2 S_p^{1/p})$.

Now we consider the remaining case where $S_p > n$. If $m < np/2 = \mathrm{O}(n)$, then the query complexity from Theorem 1 becomes $\tilde{\mathrm{O}}(n^{1-1/p} \,/\, \epsilon^2)$. Otherwise we have $2m/n \geq p$, which allows us to apply the following bound on our binomial coefficient:

$$S_p \geq n \binom{2m/n}{p} \geq n \left(\frac{2m}{np}\right)^p.$$

This inequality implies that $m \leq pn^{1-1/p} S_p^{1/p}/2$, also yielding the query complexity $\tilde{\mathrm{O}}(n^{1-1/p} \,/\, \epsilon^2)$.

Compared to [24], our algorithm achieves a better query complexity when $S_p \leq n^{1+1/p}$, where the rare stars are more likely to be found via edge sampling rather than uniform vertex sampling or traversing the graph. Our algorithm also performs no worse than their algorithm does for any S_p as large as n^p. Moreover, the dependence on ϵ of our query complexity is $1/\epsilon^2$ for any value of S_p, which is an improvement over the $1/\epsilon^{10}$ dependence required in certain cases of their algorithm.

3.4 Allowing Neighbor Queries

We now briefly discuss how we may improve our algorithm when neighbor queries are allowed (in addition to degree queries and random edge queries).

[3] We may use the binomial coefficients $\binom{x}{y}$ for non-integral value x in the inequalities. These can be interpreted through alternative formulations of binomial coefficients using falling factorials or analytic functions.

For the case when $S_p > n^p$, it is unknown whether our algorithm alone achieves better performance than [24] (see table 1). However, their algorithm has the same basic framework as ours, namely that it also starts by setting $\widetilde{S}_p$ to the maximum possible number of stars, then iteratively halves this value until it is in the correct range, allowing the subroutine to correctly compute a $(1 \pm \epsilon)$-approximation of S_p. As a result, we may achieve the same performance as them in this regime by simply letting Algorithm 2 call the subroutine from [24] when $S_p \geq n^p$. We will later show tight lower bounds (up to polylogarithmic factors) to the case where all three types of queries are allowed, which is a stronger model than the one previously studied in their work.

4 Lower Bounds for Counting Stars in Undirected Graphs

In this section, we establish the lower bounds summarized in the last two columns of Table 1. We give lower bounds that apply even when the algorithm is permitted to sample random edges. Our first lower bound is proved in Section 4.1; While this is the simplest case, it provides useful intuition for the proofs of subsequent bounds. In order to overcome the new obstacle of powerful queries in our model, for larger values of S_p we create an explicit scheme for constructing families of graphs that are hard to distinguish by any algorithm even when these queries are present. Using this construction scheme, our approach obtains the bounds for all remaining ranges for S_p as special cases of a more general bound, and the general bound is proved via the straightforward application of Yao's principle and a coupling argument. Our lower bounds are tight (up to polylogarithmic factors) for all cases except for the bottom middle cell in Table 1.

4.1 Lower Bound for $S_p \leq n$

We first remark that the following construction given in Theorem 3 applies to any $S_p \leq n^p$. However, we will later show a stronger lower bound for $S_p > n$ in Section 4.2.

Theorem 3 *For any constant $p \geq 2$, any (randomized) algorithm for approximating S_p to a multiplicative factor via neighbor queries, degree queries and random edge queries with probability of success at least $2/3$ requires $\Omega(n/S_p^{1/p})$ total number of queries for any $S_p \leq n^p$.*

Proof: We now construct two families of graphs, namely $\mathcal{F}_1$ and $\mathcal{F}_2$, such that any G_1 and G_2 drawn from each respective family satisfy $S_p(G_1) = 0$ and $S_p(G_2) = \Theta(s)$ for some parameter $s > (p+1)^p = O(1)$. We construct G_1 as follows: for a subset $S \subseteq V$ of size $\lceil s^{1/p} \rceil + 1$, we create a union of a $(p-1)$-regular graph on S and a $(p-1)$-regular graph on $V \setminus S$, and add the resulting graph G_1 to $\mathcal{F}_1$. To construct all graphs in $\mathcal{F}_1$, we repeat this process for every subset S of size $\lceil s^{1/p} \rceil + 1$. $\mathcal{F}_2$ is constructed a little differently: rather

than using a $(p-1)$-regular graph on S, we use a star of size $\lceil s^{1/p} \rceil$ on this set instead. We add a union between a star on S and a $(p-1)$-regular graph on $V \setminus S$ of any possible combination to $\mathcal{F}_2$.

By construction, every $G_1 \in \mathcal{F}_1$ contains no p-stars, whereas every $G_2 \in \mathcal{F}_2$ has $\binom{\mathrm{O}(s^{1/p})}{p} = \Theta(s)$ p-stars. For any algorithm to distinguish between $\mathcal{F}_1$ and $\mathcal{F}_2$, when given a graph $G_2 \in \mathcal{F}_2$, it must be able to detect some vertex in S with probability at least 2/3. Otherwise, if we randomly generate a small induced subgraph according to the uniform distribution in $\mathcal{F}_2$ conditional on not having any vertex or edge in S, the distribution would be identical to the uniform in $\mathcal{F}_1$. Furthermore, notice that S cannot be reached via traversal using neighbor queries as it is disconnected from $V \setminus S$. The probability of sampling such vertex or edge from each query is $\mathrm{O}(s^{1/p}/n)$. Thus, $\Omega(n/s^{1/p})$ samples are required to achieve a constant factor approximation with probability 2/3. □

4.2 Overview of the Lower Bound Proof for $S_p > n$

Since graphs with large S_p contain many edges, we must modify our approach above to allow graphs from the first family to contain stars. We construct two families of graphs $\mathcal{F}_1$ and $\mathcal{F}_2$ such that the number of p-star subgraphs (S_p) contained in graphs from these families differ by some multiplicative factor $c > 1$; any algorithm aiming to approximate S_p within a multiplicative factor of $\sqrt{c}$ must distinguish between these two families with probability at least 2/3. We create *representations* of graphs that explicitly specify their adjacency list structure. Each $G_1 \in \mathcal{F}_1$ contains n_1 vertices of degree d_1, while the remaining $n_2 = n - n_1$ vertices are isolated. For each $G_2 \in \mathcal{F}_2$, we modify our representation from $\mathcal{F}_1$ by connecting each of the remaining n_2 vertices to $d_2 \gg d_1$ neighbors, so that these vertices contribute a sufficient number of stars to establish the desired difference in S_p. We hide these additional edges in carefully chosen random locations while ensuring minimal disturbance to the original graph representation; our representations are still so similar that any algorithm may not detect them without making sufficiently many queries. Moreover, we define a coupling for answering random edge queries so that the same edges are likely to be returned regardless of the underlying graph.

While the proof of [24] also uses similar families of graphs, our proof analysis greatly deviates from their proof as follows. Firstly, we apply Yao's principle which allows us to prove the lower bounds on randomized algorithms by instead showing the lower bound on deterministic algorithms on our carefully chosen distribution of input instances.[4] Secondly, rather than constructing two families of graphs via random processes, we construct our graphs with adjacency list representations explicitly, satisfying the above conditions for each lower bound we aim to prove. This allows us to avoid the difficulties in [24] regarding the generation of potential multiple edges and self-loops in the input

[4] See e.g., [42] for more information on Yao's principle.

instances. Thirdly, we define the distribution of our instances based on the permutation of the representations of these two graphs, and the location we place the edges in G_2 that are absent in G_1. We also apply the coupling argument, so that the distribution of the permutations we apply on these graphs, as well as the answers to random edge queries, are as similar as possible. As long as the small difference between these graphs is not discovered, the interaction between the algorithm and our oracle must be exactly the same. We show that with probability $1 - o(1)$, the algorithm and our oracle behave in exactly the same way whether the input instance corresponds to G_1 or G_2. Simplifying the arguments from [24], we completely bypass the algorithm's ability to make use of graph structures. Our proof only requires some conditions on the parameters n_1, d_1, n_2, d_2; this allows us to show the lower bounds for multiple ranges of S_p simply by choosing appropriate parameters.

Theorem 4 *For any constant integer $p \geq 2$, any (randomized) algorithm for approximating S_p to a multiplicative factor via neighbor queries, degree queries and random edge queries with probability of success at least $2/3$ requires $\Omega(n^{1-1/p})$ total number of queries for any $S_p = \mathrm{O}(n^p)$.*

Theorem 5 *For any constant integer $p \geq 2$, any (randomized) algorithm for approximating S_p to a multiplicative factor via neighbor queries, degree queries and random edge queries with probability of success at least $2/3$ requires $\Omega\left(\frac{n^{p-1/p}}{S_p^{1-1/p}}\right)$ total number of queries for any $S_p = \Omega(n^p)$.*

Firstly, to properly describe the adjacency list representation of the input graphs, we introduce the notion of graph representation in Section 4.2.1. Next, we state a main lemma (Lemma 2, Section 4.2.2) that establishes the constraints of parameters n_1, d_1, n_2, d_2 that allows us to create hard instances. We then move on to describe our constructions, including both the distribution for applying Yao's principle, and the implementation of the oracle for answering random edge queries in Section 4.2.3. We prove our main lemma for our construction in Section 4.2.4, and lastly, we give the appropriate parameters that complete the proof of our lower bounds in Section 4.2.5.

4.2.1 Graph Representations

Consider the following *representation* L of an adjacency list for a simple undirected graph G. Let us say that each vertex v_i has $\deg(v_i)$ neighbors numbered $1, \ldots, \deg(v_i)$, where the j^{th} neighbor of vertex v_i is identified as a pair (i, j). We use L to define the adjacency list of our graph; that is, if $L(i_1, j_1) = (i_2, j_2)$ then the j_1^{th} neighbor of v_{i_1} is v_{i_2} (and vice versa). L imposes a perfect matching between these neighbors; namely, $L(i_1, j_1) = (i_2, j_2)$ indicates that neighbors (i_1, j_1) and (i_2, j_2) are matched to each other, and this implies $L(i_2, j_2) = (i_1, j_1)$ as well. Each edge e is associated with a unique pair of matched cells. Note that there can be many such representations of G for different orderings of the neighbors.

4.2.2 Main Lemma

Our proof proceeds in two steps. First, we show the following lemma that applies to certain parameters of graphs.

Lemma 2 *Let n_1, d_1, n_2, d_2 be positive parameters satisfying the following properties: d_1 and n_2 are even, $n_2 \leq d_1 \leq 2d_2$ and $d_1 + 2d_2 < n_1$. Let $n = n_1 + n_2$, and define the following two families of graphs on n vertices:*
– $\mathcal{F}_1$: graphs containing n_1 vertices of degree d_1 and n_2 isolated vertices;
– $\mathcal{F}_2$: graphs containing n_1 vertices of degree d_1 and n_2 vertices of degree d_2.
Let $r = \frac{(d_1+d_2)n_2}{d_1 n_1}$ and $q = o(1/r)$. Then, there exists a distribution $\mathcal{D}$ of representations of graphs from $\mathcal{F}_1 \cup \mathcal{F}_2$ such that for any deterministic algorithm $\mathcal{A}$ that makes at most q total neighbor queries, degree queries and random edge queries, on the graph representation randomly drawn from $\mathcal{D}$, $\mathcal{A}$ cannot correctly identify whether the given representation is of a graph from $\mathcal{F}_1$ or $\mathcal{F}_2$ with probability at least $2/3$.

By applying Yao's principle, the following corollary is implied.

Corollary 6 *Let n_1, d_1, n_2, d_2 be parameters satisfying the properties specified in Lemma 2. Let $s_1 = n_1\binom{d_1}{p}$ and $s_2 = n_1\binom{d_1}{p} + n_2\binom{d_2}{p}$. If $s_1 = \Theta(f(n,p))$ and $s_2 \geq c \cdot s_1$ for some constant $c > 1$, then any (randomized) algorithm for approximating S_p to a multiplicative factor via neighbor queries, degree queries and random edge queries with probability of success at least $2/3$ requires $\Omega(q)$ queries for $S_p = \Theta(f(n,p))$.*

As a second step, we propose a few sets of parameters for different ranges of S_p. Applying Corollary 6, this yields lower bounds for the remaining ranges of S_p.

4.2.3 Our Constructions

Construction of $\mathcal{D}$
We prove this lemma by explicitly constructing the distribution $\mathcal{D}$. We note that our distribution $\mathcal{D}$ over all representations of graphs in $\mathcal{F}_1 \cup \mathcal{F}_2$ assigns probability 0 to many representations; i.e., its support does not necessarily include all graphs of $\mathcal{F}_1$ and $\mathcal{F}_2$. In particular, we will only include graphs of the following structure in our support; therefore, we only need to construct representations for graphs of this structure. We describe the structure first at a high level, then later explain the construction of representations in full detail.

– Graphs from $\mathcal{F}_1$: we begin with an empty graph, then add edges between every pair of vertices from $\{v_1, \ldots, v_{n_1}\}$ whose indices differ by at most $d_1/2$ (under mod n_1).[5]

[5] To be consistent with our notation where indices begin at 1, let $x \bmod y = y$ when x is a multiple of y. (Otherwise, $x \bmod y$ still denotes the remainder of $x \div y$.)

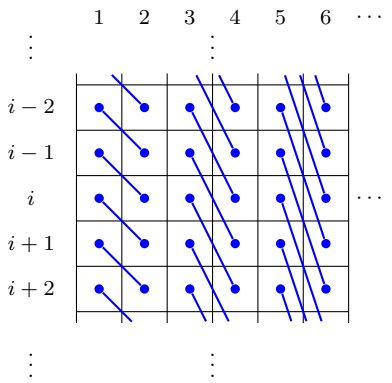


Fig. 1: first few columns of L_1

– Graphs from $\mathcal{F}_2$: we modify the graphs from $\mathcal{F}_1$ constructed above by replacing n_2 neighbors from d_2 vertices from $\{v_1, \ldots, v_{n_1}\}$, with d_2 occurrences of each of the n_2 vertices from $\{v_{n_1+1}, \ldots, v_n\}$ (and fixing the graph appropriately). The choices of vertices and neighbors to be replaced, as well as the modification to the representation, are chosen randomly and systematically to ensure that these changes are hard to detect.

Construction of graph representations for $\mathcal{F}_1$. We now define the representation L_1 for the graph $G_1 \in \mathcal{F}_1$ as follows. We let $v_1, \ldots, v_{n_1}$ be the vertices with degree d_1. Let us refer to the j^{th} pair of consecutive columns (with indices $2j-1$ and $2j$) as the j^{th} slab. Then, in the j^{th} slab, we match each cell on the left column with the cell at distance j below on the right column. Figure 1 illustrates the matching of cells in the first few columns of L_1. More formally, for each integer $i \in [n_1]$ and $j \in [d_1/2]$, we match the cells $(i, 2j-1)$ and $(i+j \bmod n_1, 2j)$ in L_1.

Since d_1 is even, this construction fills the entire table of L_1. We wish to claim that we do not create any parallel edges with this construction. Clearly, this is true within a slab. For different slabs, recall that we map cells in the j^{th} slab with those at vertical distance j away. Thus, it suffices to note that no pair of slabs uses the same distance mod n_1. Equivalently, we can note that as the maximum distance is $d_1/2$ and $d_1/2 < n_1/2$ by our assumption, the set of distances $\{j, n_1 - j\}$ for $j \in [d_1/2]$ are all disjoint. That is, our construction creates no parallel edges or self-loops.

Construction of graph representations for $\mathcal{F}_2$. Next, for each integer $a \in [n_1]$ and $b \in [d_1/2]$, we define a graph $G_2^{a,b}$ with corresponding representation $L_2^{a,b}$ by modifying L_1 as follows. First, recall that we need to add neighbors to the previously isolated vertices $v_{n_1+1}, \ldots, v_n$. These neighbors are represented as a table of size $n_2 \times d_2$ in $L_2^{a,b}$; in Figure 2, it is represented as the green rectangle in Figure $L_2^{a,b}$ (a) which is not present in L_1. We match the cells in this new table to a subtable of size $d_2 \times n_2$, which is shown as the yellow

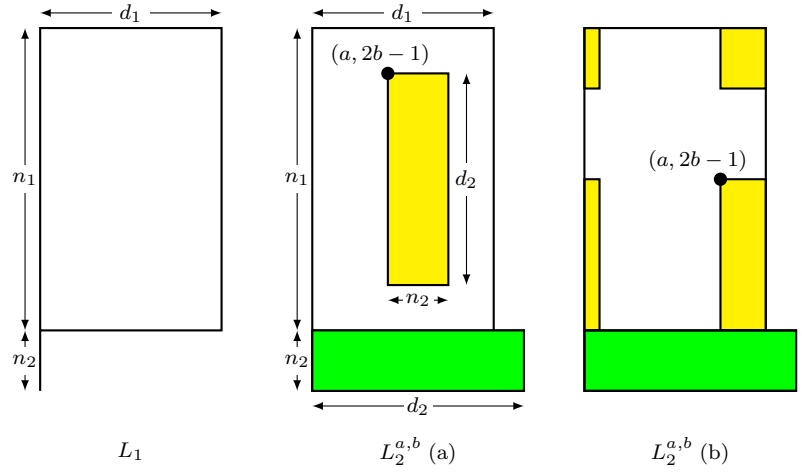


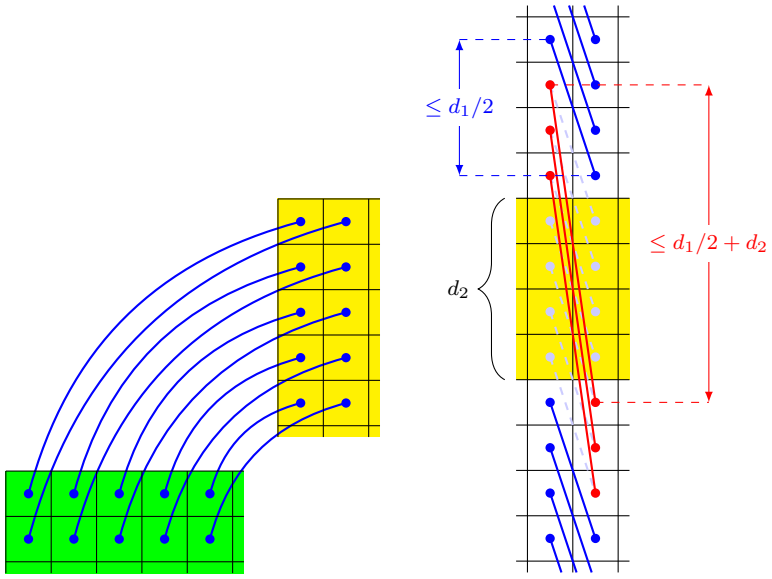
Fig. 2: Comparison between tables L_1 and L_2. $L_2^{a,b}$ (a) and (b) show two different possibilities for $L_2^{a,b}$ depending on the values of a and b.

rectangle in Figure $L_2^{a,b}$ (a). The top-left cell of this subtable corresponds to the index $(a, 2b-1)$ in $L_2^{a,b}$, and note that if $a + d_2 > n_1$ or $2b + n_2 > d_1$, this subtable may wrap around as shown in Figure $L_2^{a,b}$ (b). Since $n_2 \leq d_1$ and $d_2 < n_1$, the dimensions of this yellow rectangle do not exceed the original table in L_1.

Now we explain how we match the cells. Between the yellow and green subtables, we map them in a transposed fashion. That is, the cell with index (i, j) (relative to the green table) is mapped to the yellow cell with index (j, i) (relative to the yellow subtable), as shown in Figure 3 (a). This method guarantees that no two rows contain two pair of matched cells between them. As a result, we do not create any parallel edges or self-loops.

As we place the yellow subtable, some edges originally in L_1 may now have only one endpoint in the yellow subtable. We refer to the cells in the table that correspond to such edges as *unmatched*. Since n_2 is even and we set our offset to $(a, 2b-1)$, then every slab either does not overlap with the yellow subtable, or overlaps in the exact same rows for both columns of the slab. Thus, the only edges that have one endpoint in the yellow subtable are those that go from a cell above it to one in it. Roughly speaking, we still map the cells in the same way but ignore the distance it takes to skip over the yellow subtable. More formally, in the j^{th} slab, we pair each unmatched cell from the left and right respectively that are at vertical distance $j + d_2$ away (instead of j), as shown with the red edges in Figure 3 (b).

Now the set of distances between the cells corresponding to an edge in the j^{th} slab are $\{j, j + d_2, n_1 - j, n_1 - (j + d_2)\}$, since distances can be measured both by going down and by going up and looping around. From our assumption, $d_1/2 \leq d_2$ and $d_1/2 + d_2 < n_1/2$, and thus no distance is shared by



(a) (b)

Fig. 3: matchings in $L_2^{a,b}$

multiple slabs, and thus there are no parallel edges or self-loops.

Permutation of graph representations. Let π be a permutation over $[n]$.[6] Given a graph representation L, we define $\pi(L)$ as a new presentation of the same underlying graph, such that the indices of the vertices are permuted according to π. We may alternatively consider this operation as an interface to the original oracle. Namely, any query made on a vertex index i is translated into a query for index $\pi(i)$ to the original oracle. If a vertex index j is an answer from the oracle, then we return $\pi^{-1}(j)$ instead.

For formality, we include the full subroutines for constructing representations L_1 and $L_2^{a,b}$ as well as the permutation operation in Algorithm 3.

The distribution $\mathcal{D}$. Let $\mathcal{S}_n$ denote the set of all $n!$ permutations over $[n]$. We define $\mathcal{D}$ formally as follows: for any permutation $\pi \in \mathcal{S}_n$, the representation $\pi(L_1)$ corresponding to G_1 is drawn from $\mathcal{D}$ with probability $1/(2n!)$, and each representation $\pi(L_2^{a,b})$ corresponding to $G_2^{a,b}$ is drawn with probability $1/(n_1 d_1 n!)$ for every $(a, b) \in [n_1] \times [d_1/2]$. In other words, to draw a random instance from $\mathcal{D}$, we flip an unbiased coin to choose between families $\mathcal{F}_1$ and $\mathcal{F}_2$. We obtain a representation L_1 if we choose $\mathcal{F}_1$; otherwise we pick a random

[6] A permutation π over $[n]$ is a bijection $\pi : [n] \to [n]$.

Algorithm 3 Subroutines for constructing representations L_1, $L_2^{a,b}$, and their permutations

procedure GENERATE-$L_1(n_1, d_1, n_2, d_2)$
  for $i = 1$ to n_1 **do**
    for $j = 1$ to $d_1/2$ **do**
      $L(i, 2j-1) \leftarrow (i + j \bmod n_1, 2j)$
      $L(i + j \bmod n_1, 2j) \leftarrow (i, 2j-1)$
  return L
procedure GENERATE-$L_2^{a,b}(n_1, d_1, n_2, d_2, a, b)$
  $L \leftarrow$ GENERATE-$L_1(n_1, d_1, n_2, d_2)$
  for $i = 1$ to n_2 **do** ▷ match entries of the two rectangles
    for $j = 1$ to d_2 **do**
      $L(a + j \bmod n_1, (2b-1) + i \bmod d_1) \leftarrow (n_1 + i, j)$
      $L(n_1 + i, j) \leftarrow (a + j \bmod n_1, (2b-1) + i \bmod d_1)$
  for $i = 0$ to $n_2/2 - 1$ **do** ▷ fix unmatched cells
    $j \leftarrow b + i \bmod (d_1/2)$
    for $k = 0$ to $j - 1$ **do**
      $L(a - j + k \bmod n_1, 2j-1) \leftarrow (a + d_2 + k \bmod n_1, 2j)$
      $L(a + d_2 + k \bmod n_1, 2j) \leftarrow (a - j + k \bmod n_1, 2j-1)$
  return L
procedure PERMUTE(L, π)
  for each (i, j) where $L(i, j)$ is defined **do**
    $(i', j') \leftarrow L(i, j)$
    $L'(\pi(i), j) \leftarrow (\pi(i'), j')$
  return L'

representation $L_2^{a,b}$ for $\mathcal{F}_2$. Lastly, we apply a random permutation π to such representation.

Answering Random Edge Queries

Notice that Yao's principle allows us to remove randomness used by the algorithm, but the randomness of the oracle remains for the random edge queries. For any representation we draw from $\mathcal{D}$, the oracle must return an edge uniformly at random for each random edge query. Nonetheless, we may choose our own implementation of the oracle as long as this condition is ensured. We apply a coupling argument that imposes dependencies between the behaviors of our oracle between when the underlying graph is from $\mathcal{F}_1$ or $\mathcal{F}_2$. Let $m_1 = d_1 n_1/2$ and $m_2 = (d_1 n_1 + d_2 n_2)/2$ denote the number of edges of graphs from $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively.

Our oracle works differently depending on which family the graph comes from. The following describes the behavior of our oracle for a single query, and note that all queries should be evaluated independently.

Query to L_1. We simply return an edge chosen uniformly at random. That is, we pick a random matched pair of cells in L_1, and return the vertices corresponding to the rows of those cells.

Query to $L_2^{a,b}$. Let $m_s^{a,b}$ denote the number of edges shared by both L_1 and $L_2^{a,b}$. With probability $m_s^{a,b}/m_2$, we return the same edge we choose for L_1. Otherwise, we return an edge chosen uniformly at random from the set of edges in $L_2^{a,b}$ but not in L_1.

Our oracle clearly returns an edge chosen uniformly at random from the corresponding representation. The benefit of using this coupling oracle is that we increase the probability that the same edge is returned to $m_s^{a,b}/m_2$. By our construction, the cells in L_1 that are modified to obtain $L_2^{a,b}$ are fully contained within the subtable of size $(d_1+d_2)n_2$ obtained by extending the yellow subtable to include $d_1/2$ more rows above and below. $m_s^{a,b} \geq (d_1n_1 - (d_1 + d_2)n_2)/2$. Thus, our oracle may only return a different edge with probability

$$1 - \frac{m_s^{a,b}}{m_2} \leq 1 - \frac{d_1n_1 - (d_1+d_2)n_2}{d_1n_1 + d_2n_2} = \frac{d_1n_2}{d_1n_1 + d_2n_2} \leq r.$$

4.2.4 Proof of Lemma 2

Recall that we consider a deterministic algorithm $\mathcal{A}$ that makes at most $q = o(1/r)$ queries. We may describe the behavior between $\mathcal{A}$ and the oracle with its *query-answer history*. Notice that since $\mathcal{A}$ is deterministic, if every answer that $\mathcal{A}$ receives from the oracle is the same, then $\mathcal{A}$ must return the same answer, regardless of the underlying graph. Our general approach is to show that for most permutations π, running $\mathcal{A}$ with instance $\pi(L_1)$ will result in the same query-answer history as running with $\pi(L_2^{a,b})$ for most random parameters π and (a, b). If these histories are equivalent, then $\mathcal{A}$ may answer correctly for only roughly half of the distribution.

Throughout this section, we refer to our indices before applying π to the representation. We bound the probability that the query-answer histories are different using an inductive argument as follows. Suppose that at some point during the execution of $\mathcal{A}$, the history only contains vertices of indices from $[n_1]$, and all cells in the history are matched in the same way in both L_1 and $L_2^{a,b}$. This inductive hypothesis restricts the possible parameters π and (a, b) to those that yield same history up to this point. We now consider the probability that the next query-answer pair differs, and aim to bound this probability by $O(r)$.

Firstly, we consider a degree query. By our hypothesis, for a vertex of index outside $[n_1]$ to be queried, $\mathcal{A}$ must specify a vertex it has not chosen before. Notice that $\mathcal{A}$ may learn about up to 2 vertices from each query-answer pair, so at least $n - 2q$ vertices have never appeared in the history. Since we pick a random permutation π for our construction, the probability that the queried vertex has index outside $[n_1]$ is $n_2/(n - 2q)$. As $r \geq n_2/n_1 \geq 1/n_1$, we have $q = o(n_1)$ and our probability simplifies to at most

$$\frac{n_2}{n-2q} = \frac{n_2}{(n_1+n_2) - 2 \cdot o(n_1)} \leq \frac{n_2}{n_1(1-o(1))} = O(r).$$

Next, we consider a neighbor query. From the argument above, with probability $1-O(r)$, the queried vertex given by $\mathcal{A}$ has an index from $[n_1]$. Similarly,

$\mathcal{A}$ may learn about up to 2 cells from each query-answer pair. Notice that there are $(d_1 + d_2)n_2$ different possible (a, b) for which each of these cells could be located in the yellow subtable or the two $(d_1/2) \times n_2$ strips above and below it. As a result, out of $d_1 n_1 - ((d_1 + d_2)n_2)q$ remaining possible locations for the yellow subtable, the queried cell and the corresponding answer may be in at most $2(d_1 + d_2)n_2$ of them. As (a, b) is randomly chosen, the probability that this next query-answer pair is different is at most

$$\frac{2(d_1 + d_2)n_2}{d_1 n_1 - ((d_1 + d_2)n_2)q} = \frac{2r}{1 - rq} = \frac{2r}{1 - o(1)} = O(r).$$

Lastly, we consider a random edge query. From the construction in Section 4.2.3 above, the probability that the returned random edge differs is $O(r)$, regardless of the parameters.

From this inductive argument, the probability that the history differs at each step is at most $O(r)$. As $\mathcal{A}$ only make q queries, the probability that the history differs is at most $q \cdot O(r) = o(1)$. Thus with probability $1 - o(1)$, it is impossible for $\mathcal{A}$ to distinguish whether the underlying graph is from $\mathcal{F}_1$ or $\mathcal{F}_2$. Since each family is included in $\mathcal{D}$ with probability density $1/2$, as $\mathcal{A}$ is deterministic, the answer given by $\mathcal{A}$ for these cases is correct for only half of them. Thus, the probability of $\mathcal{A}$ correctly distinguish between the two graph families is only $1 - \frac{1}{2}(1 - o(1)) = \frac{1}{2} + o(1)$, as required.

4.2.5 Establishing Lower Bounds

Now we propose the feasible asymptotic parameters according to Lemma 2 and Lemma 6 in order to establish our lower bounds through the following claim.

Claim 3 *There exists parameters n_1, d_1, n_2, d_2 satisfying the properties specified in Lemma 2, yielding values s_1, s_2 satisfying the properties in Lemma 6 for constant $p \geq 2$ in each of the following cases:*

1. $n_1 = \Theta(n), d_1 = \Theta((s/n)^{1/p}), n_2 = \Theta(1), d_2 = \Theta(s^{1/p})$ *for* $s = f(n, p) = \mathrm{O}(n^p)$
2. $n_1 = \Theta(n), d_1 = \Theta((s/n)^{1/p}), n_2 = \Theta(s/n^p), d_2 = \Theta(n)$ *for* $s = f(n, p) = \Omega(n^p)$

Proof: Consider the first case. For any sufficiently large n, $n_2 \leq d_1 \leq 2d_2$ since $\mathrm{O}(1) \subseteq \mathrm{O}((s/p)^{1/p}) \subseteq \mathrm{O}(s^{1/p})$ for $s = \mathrm{O}(n^p)$ when we pick $s \geq p$. Next, there exists d_1, d_2, n_1 satisfying $d_1 + 2d_2 < n_1$ because $\mathrm{O}((s/n)^{1/p} + s^{1/p}) = \mathrm{O}((n^p)^{1/p}) = \mathrm{O}(n)$. Further, $n_1\binom{d_1}{p} = \Theta(n \cdot ((s/n)^{1/p})^p) = \Theta(s)$ and $n_2\binom{d_2}{p} = \Theta(1 \cdot (s^{1/p})^p) = \Theta(s)$, so $s_2 \geq c \cdot s_1$ for some $c > 1$, as desired. Without modifying any asymptotic bounds, we may double the variables to ensure that d_1 and n_2 are even.

Now consider the second case. For any sufficiently large n, $n_2 \leq d_1 \leq 2d_2$ and $d_1 + 2d_2 < n_1$, since $\mathrm{O}(s/n^p) \subseteq \mathrm{O}((s/p)^{1/p}) \subseteq \mathrm{O}(n)$ and $\mathrm{O}((s/n)^{1/p} + n) \subseteq \mathrm{O}((n^p)^{1/p}) \subseteq \mathrm{O}(n)$, respectively, because $s = O(n^{p+1})$ holds for every simple

graph. Further, $n_1\binom{d_1}{p} = \Theta(n\cdot((s/n)^{1/p})^p) = \Theta(s)$ and $n_2\binom{d_2}{p} = \Theta(s/n^p\cdot n^p) = \Theta(s)$, so $s_2 \geq c\cdot s_1$ for some $c > 1$, as desired. We may again double the variables if necessary. □

We remark that in the first case,

$$r = \frac{(d_1+d_2)n_2}{d_1 n_1} = \Theta\left(\frac{((s/n)^{1/p} + s^{1/p})\cdot 1}{(s/n)^{1/p}\cdot n}\right) = \Theta\left(\frac{1}{n^{1-1/p}}\right),$$

and in the second case,

$$r = \frac{(d_1+d_2)n_2}{d_1 n_1} = \Theta\left(\frac{((s/n)^{1/p} + n)\cdot s/n^p}{(s/n)^{1/p}\cdot n}\right) = \Theta\left(\frac{s^{1-1/p}}{n^{p-1/p}}\right).$$

By applying Corollary 6 to the value of r computed above in each case, we obtain Theorem 4 and Theorem 5, respectively.

5 Extension to Directed Graphs

In this section, we extend our model to the directed graph case. First, we formally give the specification of this new model. Since most of the specification from the undirected graph model given in Section 2 still applies to the directed case, we only explain the differences between these models. We assume separate adjacency lists for in-neighbors and out-neighbors, allowing for a vertex to ask about either type of neighbor. Similarly, a degree query may ask for either of the in-degree or the out-degree (denoted $\deg^-(v)$ and $\deg^+(v)$ for a vertex v, respectively). Random edge queries now return directed edges (u, v); the algorithm knows both the endpoints and the direction. Moreover, we can view u as a random vertex drawn with probability proportional to the out-degree of u. Similarly, v is a random vertex drawn with probability proportional to its in-degree. Also, if we return u and v each with probability $1/2$, the return vertex can be viewed as a random vertex drawn with probability proportional to its total degree.

Notice the number of stars where all edges point the same direction (inward or outward) can be computed easily by modifying the weighted vertex sampling to sample using in-degree or out-degree respectively and then applying the algorithm from Section 3. Thus, the main difficulty is to handle the case when the star subgraph has edges pointing different directions. In the following, we focus on the simplest case of stars with edges pointing both inward and outward, namely when the in-degree and out-degree are both one. This is exactly the problem of approximately counting the number of directed paths of length two.

5.1 Lower Bound

By constructing hard instances similar to those of Lemma 4.1, we obtain a lower bound of $\Omega(n)$. More formally, letting $L(G)$ denote the number of paths of length two in the directed graph G, we prove the following theorem.

Theorem 7 *Any (randomized) algorithm for approximating $L(G)$ to a multiplicative factor via neighbor queries, degree queries and random edge queries requires $\Omega(n)$ total number of queries. In particular, this number of queries is necessary to distinguish the case where $L(G) = 0$ and the case where $L(G) = n$ with probability 2/3.*

Proof: Without loss of generality, we assume n is even. Now, we partition the vertex set V into S and T such that $|S| = |T| = n/2$. Let $\mathcal{G}_1$ be the family of graphs that contains only G_1, the complete bipartite graph where every vertex in S has an edge pointing to every vertex in T. Let $\mathcal{G}_2$ be the family of graphs $G_{(t,s)}$ constructed by taking the graph from $\mathcal{G}_1$ and adding one extra back edge $(t, s) \in T \times S$. Notice that there can be many adjacency list representations of each graph, and this affect the answers to neighbor queries. We associate each possible adjacency list representation to each graph, and include all possible such representations in the family.

Clearly, $L(G_1) = 0$, whereas $L(G_{(t,s)}) = n$ for every $G_{(t,s)} \in \mathcal{G}_2$. For any algorithm to distinguish between $\mathcal{G}_1$ and $\mathcal{G}_2$, when given a graph $G_{(t,s)}$ from $\mathcal{G}_2$, it must be able to detect the vertex s or t, the endpoints of the extra edge, with probability at least 2/3. Otherwise, if neither s nor t is discovered, the subgraph induced by vertices that the algorithm sees from both families would be exactly the same. The probability of sampling vertices s or t from a vertex sampling, as well as their incident edges from an edge sampling, is $\mathrm{O}(1/n)$. Similarly, in order to reach s or t from one of their neighbors, the algorithm must provide the index of s or t in order to make such neighbor query, which may only succeed with probability $\mathrm{O}(1/n)$. Thus, $\Omega(n)$ samples are required in order to find s or t with probability 2/3, which establishes our lower bound. □

5.2 Upper Bound

For each $v \in V$, define $l(v) = \deg^-(v) \cdot \deg^+(v)$, which represents the number of length two paths whose middle vertex is v. Thus the number of paths of length two, which we aim to approximate, can be written as $L = \sum_{v \in V} l(v)$. Notice that $2n$ degree queries suffice for exactly computing the number of such paths, already matching the lower bound. We explore this problem further by asking whether there is a class of graphs which requires $o(n)$ queries. To this end, we consider the class of directed graphs such that there exists a bound on the ratio of in-degree to out-degree. More specifically, we assume that there exists a given bound $r \geq 1$ such that $\frac{1}{r} \leq \frac{\deg^-(v)}{\deg^+(v)} \leq r$, limiting the ratio between the in-degree and the out-degree of any vertex in G.

Under this additional assumption, we obtain a sublinear time algorithm by reduction to what is essentially the undirected case. Our approach is to modify the weighted vertex sampling process via *rejection sampling* so that the probability of sampling a vertex v becomes proportional to $\sqrt{l(v)}$, bringing the sampling probability of each vertex closer to the number of paths centered

at that vertex by the rejection sampling method. Then we use a modified variation of Algorithm 2 to approximate $\sum_{v\in V}(\sqrt{l(v)})^2$; this modification will be explained in the proof of Theorem 8.

First, we propose a subroutine (Algorithm 4) for drawing a multi-set S of s random vertices, such that each vertex is sampled independently with probability $\sqrt{l(v)}/\sum_{v'\in V}\sqrt{l'(v)}$, with overall success probability $1-\delta$.

Algorithm 4 Subroutine for generating s independent samples according to Claim 4

procedure SAMPLER(s, δ)
  $t \leftarrow 8rs\log 1/\delta$
  $S \leftarrow \emptyset$
  for $i = 1$ to t **do**
    $(u, v) \leftarrow$ an edge drawn from a random edge query
    $X_i \leftarrow$ a (fresh) random indicator variable with $\Pr[X_i = 1] = \frac{1}{\sqrt{r}}\sqrt{\frac{\deg^-(u)}{\deg^+(u)}}$
    if $X_i = 1$ **then**
      $S \leftarrow S \cup \{u\}$
      if $|S| = s$ **then**
        return S
  report FAILURE

Claim 4 *In the directed graph model, given random edge sampling and in-degree and out-degree queries, with probability $1-\delta$, Algorithm 4 generates s random vertices such that each vertex v is returned with probability $\sqrt{l(v)}/\sum_{v'\in V}\sqrt{l(v')}$ by making $\mathrm{O}(rs\log 1/\delta)$ queries, assuming there exists some value r such that $\frac{1}{r} \le \frac{\deg^-(v)}{\deg^+(v)} \le r$ for every $v \in V$.*

Proof: The process to obtain the vertices is explained in Algorithm 4. Each vertex u is chosen from a random edge sampling with probability proportional to its out-degree, $\deg^+(u)$. We only keep u with probability $\frac{1}{\sqrt{r}}\sqrt{\frac{\deg^-(u)}{\deg^+(u)}}$, so the probability that any vertex u is added to S is proportional to $\deg^+(u)\cdot\sqrt{\frac{\deg^-(u)}{\deg^+(u)}} = \sqrt{l(u)}$, as desired.

Observe that since $\frac{1}{r} \le \frac{\deg^-(v)}{\deg^+(v)} \le r$, we have that $\frac{1}{r} \le \frac{1}{\sqrt{r}}\sqrt{\frac{\deg^-(v)}{\deg^+(v)}} \le 1$. In other words, in each iteration of the algorithm with probability at least $1/r$, X_i is one and we add a random vertex to S while spending $\mathrm{O}(1)$ queries. Now, we use the Chernoff bound (Theorem 12) to show that Algorithm 4 generates a set S containing s vertices with probability at least $1-\delta$. The probability that the algorithm fails to return a set S with s samples is upper-bounded by

$$\Pr\left[\sum_{i=1}^{t} X_i < s\right] = \Pr\left[\frac{\sum_{i=1}^{t} X_i}{t} < \frac{s}{t}\right] \le \Pr\left[\frac{\sum_{i=1}^{t} X_i}{t} < \left(1-\frac{1}{2}\right)\cdot\frac{1}{r}\right]$$
$$< e^{-(\frac{1}{2})(\frac{1}{2})^2(\frac{1}{r})(t)} = \delta,$$

which concludes the statement of the claim. □

Theorem 8 *Assuming there exists some value r such that $\frac{1}{r} \leq \frac{\deg^-(v)}{\deg^+(v)} \leq r$ for every $v \in V$ in the directed graph G, then there exists an algorithm that, using degree queries and random edge queries, computes a $(1 \pm \epsilon)$-approximation of the number of directed paths of length two in G with success probability $2/3$ using $\mathrm{O}(r\sqrt{n}/\epsilon^3)$ queries.*

Proof: Define $L' = \sum_{v \in V} \sqrt{l(v)}$. In Algorithm 4 we describe a method for sampling vertices with probability $\sqrt{l(v)}/L'$ via rejection sampling, which increases the time or query complexities only asymptotically by a factor of $O(r)$ (for a constant δ).

For now, assume L' is known. To compute $L = \sum_{v \in V} (\sqrt{l(v)})^2$, we modify Algorithm 1 so that it computes an unbiased estimator for L: we set X as $\sqrt{l(v)}$ and $Y = L'X$ so that

$$E[Y] = \sum_{v \in V} \frac{\sqrt{l(v)}}{L'} \left(L' \sqrt{l(v)} \right) = \sum_{v \in V} (\sqrt{l(v)})^2 = L.$$

This modified algorithm requires $\tilde{\mathrm{O}}(\sqrt{n}/\epsilon^2)$ samples obtained from Algorithm 4. The proof of the variance bound (Lemma 1) can be subsequently modified to obtain $\mathrm{Var}[Y] = \mathrm{O}(\sqrt{n}L^2)$, which implies the desired correctness and success probability for the full algorithm. The proof is essentially a repetition of that for approximating S_p, so we omit it from the paper. In fact, this modified algorithm approximates the second moment of the sequence of vertices, where the number of occurrences of each vertex is proportional to $\sqrt{l(v)}$ – this problem has been extensively studied in [5] in the context of sublinear space, which includes all the required proof.

Now, using $\tilde{\mathrm{O}}(\sqrt{n}/\epsilon^3)$ samples obtained from Algorithm 4, we may then approximate L' using the idea described in Section 3.3.1. Assume $\hat{L}'$ is the estimated value. Then, we have

$$\mathrm{E}[Y] = \frac{\hat{L}'}{L'} L.$$

Thus, an accurate estimate of L' will be sufficient to yield an accurate estimate of L.

As the number of samples required from Algorithm 1 is $s = \tilde{\mathrm{O}}(\sqrt{n}/\epsilon^3)$, to obtain a constant success probability, we need to make $\tilde{\mathrm{O}}(r\sqrt{n}/\epsilon^3)$ combined queries, constituting our query complexity for approximating the number of directed paths of length two. □

Corollary 9 *Assuming that the ratio between the in-degree and the out-degree of every vertex in the directed graph G is bounded above and below by some given constant, then there exists an algorithm that, using degree queries and random edge queries, computes a $(1 \pm \epsilon)$-approximation of the number of directed paths of length two in G with success probability 2/3 using $\mathrm{O}(\sqrt{n}/\epsilon^3)$ queries.*

Compliance with Ethical Standards

Conflict of Interest: The authors declare that they have no conflict of interest.

Acknowledgements Aliakbarpour, Gouleakis, Peebles, Rubinfeld and Yodpinyanee were supported by the National Science Foundation Graduate Research Fellowship under Grant No. CCF-1217423, CCF-1065125 and CCF-1420692. Peebles was also supported by Grant No. CCF-1122374. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

In addition, Rubinfeld was supported by the Israel Science Foundation grant 1536/14, and Yodpinyanee was supported by the Development and Promotion of Science and Technology Talents Project scholarship, Royal Thai Government.

We thank Dana Ron for her valuable contribution to this paper. We thank Peter Haas and Samuel Madden for helpful discussions. We thank anonymous reviewers for their insightful comments on the preliminary version of this paper.

References

1. Ahn, K.J., Guha, S., McGregor, A.: Graph sketches: sparsification, spanners, and subgraphs. In: Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012, pp. 5–14 (2012). DOI 10.1145/2213556.2213560. URL http://doi.acm.org/10.1145/2213556.2213560
2. Alon, N., Dao, P., Hajirasouliha, I., Hormozdiari, F., Sahinalp, S.C.: Biomolecular network motif counting and discovery by color coding. In: Proceedings 16th International Conference on Intelligent Systems for Molecular Biology (ISMB), Toronto, Canada, July 19-23, 2008, pp. 241–249 (2008). DOI 10.1093/bioinformatics/btn163. URL http://dx.doi.org/10.1093/bioinformatics/btn163
3. Alon, N., Gibbons, P.B., Matias, Y., Szegedy, M.: Tracking join and self-join sizes in limited storage. J. Comput. Syst. Sci. **64**(3), 719–747 (2002). DOI 10.1006/jcss.2001.1813. URL http://dx.doi.org/10.1006/jcss.2001.1813
4. Alon, N., Gutner, S.: Balanced hashing, color coding and approximate counting. In: Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers, pp. 1–16 (2009). DOI 10.1007/978-3-642-11269-0_1. URL http://dx.doi.org/10.1007/978-3-642-11269-0_1
5. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. J. Comput. Syst. Sci. **58**(1), 137–147 (1999). DOI 10.1006/jcss.1997.1545. URL http://dx.doi.org/10.1006/jcss.1997.1545
6. Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica **17**(3), 209–223 (1997). DOI 10.1007/BF02523189. URL http://dx.doi.org/10.1007/BF02523189
7. Amini, O., Fomin, F.V., Saurabh, S.: Counting subgraphs via homomorphisms. SIAM J. Discrete Math. **26**(2), 695–717 (2012). DOI 10.1137/100789403. URL http://dx.doi.org/10.1137/100789403
8. Bar-Yossef, Z., Kumar, R., Sivakumar, D.: 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., pp. 623–632 (2002). URL http://dl.acm.org/citation.cfm?id=545381.545464
9. Batu, T., Berenbrink, P., Sohler, C.: A sublinear-time approximation scheme for bin packing. Theor. Comput. Sci. **410**(47-49), 5082–5092 (2009). DOI 10.1016/j.tcs.2009.08.006. URL http://dx.doi.org/10.1016/j.tcs.2009.08.006
10. Becchetti, L., Boldi, P., Castillo, C., Gionis, A.: Efficient semi-streaming algorithms for local triangle counting in massive graphs. In: Proceedings of the 14th ACM SIGKDD

International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008, pp. 16–24 (2008). DOI 10.1145/1401890.1401898. URL http://doi.acm.org/10.1145/1401890.1401898
11. Bhuvanagiri, L., Ganguly, S., Kesh, D., Saha, C.: Simpler algorithm for estimating frequency moments of data streams. In: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp. 708–713. ACM (2006)
12. Blais, E., Brody, J., Matulef, K.: Property testing lower bounds via communication complexity. Computational Complexity **21**(2), 311–358 (2012). DOI 10.1007/s00037-012-0040-x. URL http://dx.doi.org/10.1007/s00037-012-0040-x
13. Buriol, L.S., Frahling, G., Leonardi, S., Marchetti-Spaccamela, A., Sohler, C.: 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, pp. 253–262 (2006). DOI 10.1145/1142351.1142388. URL http://doi.acm.org/10.1145/1142351.1142388
14. Canonne, C.L., Rubinfeld, R.: Testing probability distributions underlying aggregated data. In: Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I, pp. 283–295 (2014). DOI 10.1007/978-3-662-43948-7_24. URL http://dx.doi.org/10.1007/978-3-662-43948-7_24
15. Coppersmith, D., Kumar, R.: An improved data stream algorithm for frequency moments. In: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004, pp. 151–156 (2004). URL http://dl.acm.org/citation.cfm?id=982792.982815
16. Duke, R.A., Lefmann, H., Rödl, V.: A fast approximation algorithm for computing the frequencies of subgraphs in a given graph. SIAM J. Comput. **24**(3), 598–620 (1995). DOI 10.1137/S0097539793247634. URL http://dx.doi.org/10.1137/S0097539793247634
17. Eden, T., Levi, A., Ron, D., Seshadhri, C.: 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, pp. 614–633 (2015). DOI 10.1109/FOCS.2015.44. URL http://dx.doi.org/10.1109/FOCS.2015.44
18. Feige, U.: On sums of independent random variables with unbounded variance and estimating the average degree in a graph. SIAM J. Comput. **35**(4), 964–984 (2006). DOI 10.1137/S0097539704447304. URL http://dx.doi.org/10.1137/S0097539704447304
19. Flum, J., Grohe, M.: The parameterized complexity of counting problems. SIAM J. Comput. **33**(4), 892–922 (2004). DOI 10.1137/S0097539703427203. URL http://dx.doi.org/10.1137/S0097539703427203
20. Fomin, F.V., Lokshtanov, D., Raman, V., Saurabh, S., Rao, B.V.R.: Faster algorithms for finding and counting subgraphs. J. Comput. Syst. Sci. **78**(3), 698–706 (2012). DOI 10.1016/j.jcss.2011.10.001. URL http://dx.doi.org/10.1016/j.jcss.2011.10.001
21. Getoor, L., Taskar, B., Koller, D.: Selectivity estimation using probabilistic models. In: Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001, pp. 461–472 (2001). DOI 10.1145/375663.375727. URL http://doi.acm.org/10.1145/375663.375727
22. Goldreich, O.: On the communication complexity methodology for proving lower bounds on the query complexity of property testing. Electronic Colloquium on Computational Complexity (ECCC) **20**, 73 (2013). URL http://eccc.hpi-web.de/report/2013/073
23. Goldreich, O., Ron, D.: Approximating average parameters of graphs. Random Struct. Algorithms **32**(4), 473–493 (2008). DOI 10.1002/rsa.20203. URL http://dx.doi.org/10.1002/rsa.20203
24. Gonen, M., Ron, D., Shavitt, Y.: Counting stars and other small subgraphs in sublinear-time. SIAM J. Discrete Math. **25**(3), 1365–1411 (2011). DOI 10.1137/100783066. URL http://dx.doi.org/10.1137/100783066
25. Gonen, M., Shavitt, Y.: Approximating the number of network motifs. Internet Mathematics **6**(3), 349–372 (2009). DOI 10.1080/15427951.2009.10390645. URL http://dx.doi.org/10.1080/15427951.2009.10390645
26. Grochow, J.A., Kellis, M.: Network motif discovery using subgraph enumeration and symmetry-breaking. In: Research in Computational Molecular Biology, 11th Annual International Conference, RECOMB 2007, Oakland, CA, USA, April 21-25,

2007, Proceedings, pp. 92–106 (2007). DOI 10.1007/978-3-540-71681-5_7. URL http://dx.doi.org/10.1007/978-3-540-71681-5_7

27. Haas, P.J., Ilyas, I.F., Lohman, G.M., Markl, V.: Discovering and exploiting statistical properties for query optimization in relational databases: A survey. Statistical Analysis and Data Mining **1**(4), 223–250 (2009). DOI 10.1002/sam.10016. URL http://dx.doi.org/10.1002/sam.10016
28. Haas, P.J., Naughton, J.F., Seshadri, S., Swami, A.N.: Selectivity and cost estimation for joins based on random sampling. J. Comput. Syst. Sci. **52**(3), 550–569 (1996). DOI 10.1006/jcss.1996.0041. URL http://dx.doi.org/10.1006/jcss.1996.0041
29. Hales, D., Arteconi, S.: Motifs in evolving cooperative networks look like protein structure networks. NHM **3**(2), 239–249 (2008). DOI 10.3934/nhm.2008.3.239. URL http://dx.doi.org/10.3934/nhm.2008.3.239
30. Hassidim, A., Kelner, J.A., Nguyen, H.N., Onak, K.: 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, pp. 22–31 (2009). DOI 10.1109/FOCS.2009.77. URL http://dx.doi.org/10.1109/FOCS.2009.77
31. Hormozdiari, F., Berenbrink, P., Przulj, N., Sahinalp, S.C.: Not all scale-free networks are born equal: The role of the seed graph in PPI network evolution. PLoS Computational Biology **3**(7) (2007). DOI 10.1371/journal.pcbi.0030118. URL http://dx.doi.org/10.1371/journal.pcbi.0030118
32. Indyk, P., Woodruff, D.P.: Optimal approximations of the frequency moments of data streams. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005, pp. 202–208 (2005). DOI 10.1145/1060590.1060621. URL http://doi.acm.org/10.1145/1060590.1060621
33. Kane, D.M., Mehlhorn, K., Sauerwald, T., Sun, H.: 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, pp. 598–609 (2012). DOI 10.1007/978-3-642-31585-5_53. URL http://dx.doi.org/10.1007/978-3-642-31585-5_53
34. Kashtan, N., Itzkovitz, S., Milo, R., Alon, U.: Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. Bioinformatics **20**(11), 1746–1758 (2004). DOI 10.1093/bioinformatics/bth163. URL http://dx.doi.org/10.1093/bioinformatics/bth163
35. Kolountzakis, M.N., Miller, G.L., Peng, R., Tsourakakis, C.E.: Efficient triangle counting in large graphs via degree-based vertex partitioning. Internet Mathematics **8**(1-2), 161–185 (2012). DOI 10.1080/15427951.2012.625260. URL http://dx.doi.org/10.1080/15427951.2012.625260
36. Lee, J., Kim, D., Chung, C.: Multi-dimensional selectivity estimation using compressed histogram information. In: SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA., pp. 205–214 (1999). DOI 10.1145/304182.304200. URL http://doi.acm.org/10.1145/304182.304200
37. Manjunath, M., Mehlhorn, K., Panagiotou, K., Sun, H.: Approximate counting of cycles in streams. In: Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings, pp. 677–688 (2011). DOI 10.1007/978-3-642-23719-5_57. URL http://dx.doi.org/10.1007/978-3-642-23719-5_57
38. Markl, V., Haas, P.J., Kutsch, M., Megiddo, N., Srivastava, U., Tran, T.M.: Consistent selectivity estimation via maximum entropy. VLDB J. **16**(1), 55–76 (2007). DOI 10.1007/s00778-006-0030-1. URL http://dx.doi.org/10.1007/s00778-006-0030-1
39. McGregor, A., Vorotnikova, S., Vu, H.T.: 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, pp. 401–411 (2016). DOI 10.1145/2902251.2902283. URL http://doi.acm.org/10.1145/2902251.2902283
40. Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., Alon, U.: Network motifs: simple building blocks of complex networks. Science **298**(5594), 824–827 (2002)

41. Motwani, R., Panigrahy, R., Xu, Y.: Estimating sum by weighted sampling. In: Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings, pp. 53–64 (2007). DOI 10.1007/978-3-540-73420-8_7. URL http://dx.doi.org/10.1007/978-3-540-73420-8_7
42. Motwani, R., Raghavan, P.: Randomized algorithms. Chapman & Hall/CRC (2010)
43. Nguyen, H.N., Onak, K.: 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, pp. 327–336 (2008). DOI 10.1109/FOCS.2008.81. URL http://dx.doi.org/10.1109/FOCS.2008.81
44. Onak, K., Ron, D., Rosen, M., Rubinfeld, R.: 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, pp. 1123–1131 (2012). URL http://portal.acm.org/citation.cfm?id=2095204&CFID=63838676&CFTOKEN=79617016
45. Parnas, M., Ron, D.: Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. Theor. Comput. Sci. **381**(1-3), 183–196 (2007). DOI 10.1016/j.tcs.2007.04.040. URL http://dx.doi.org/10.1016/j.tcs.2007.04.040
46. Poosala, V., Ioannidis, Y.E.: Selectivity estimation without the attribute value independence assumption. In: VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece, pp. 486–495 (1997). URL http://www.vldb.org/conf/1997/P486.PDF
47. Przulj, N., Corneil, D.G., Jurisica, I.: Modeling interactome: scale-free or geometric? Bioinformatics **20**(18), 3508–3515 (2004). DOI 10.1093/bioinformatics/bth436. URL http://dx.doi.org/10.1093/bioinformatics/bth436
48. Scott, J., Ideker, T., Karp, R.M., Sharan, R.: Efficient algorithms for detecting signaling pathways in protein interaction networks. Journal of Computational Biology **13**(2), 133–144 (2006). DOI 10.1089/cmb.2006.13.133. URL http://dx.doi.org/10.1089/cmb.2006.13.133
49. Shlomi, T., Segal, D., Ruppin, E., Sharan, R.: Qpath: a method for querying pathways in a protein-protein interaction network. BMC Bioinformatics **7**, 199 (2006). DOI 10.1186/1471-2105-7-199. URL http://dx.doi.org/10.1186/1471-2105-7-199
50. Swami, A.N., Schiefer, K.B.: On the estimation of join result sizes. In: Advances in Database Technology - EDBT'94. 4th International Conference on Extending Database Technology, Cambridge, United Kingdom, March 28-31, 1994, Proceedings, pp. 287–300 (1994). DOI 10.1007/3-540-57818-8_58. URL http://dx.doi.org/10.1007/3-540-57818-8_58
51. Wernicke, S.: Efficient detection of network motifs. IEEE/ACM Trans. Comput. Biology Bioinform. **3**(4), 347–359 (2006). DOI 10.1109/TCBB.2006.51. URL http://dx.doi.org/10.1109/TCBB.2006.51
52. Williams, R.: Finding paths of length k in $o^*(2^k)$ time. Inf. Process. Lett. **109**(6), 315–318 (2009). DOI 10.1016/j.ipl.2008.11.004. URL http://dx.doi.org/10.1016/j.ipl.2008.11.004
53. Williams, V.V., Williams, R.: Finding, minimizing, and counting weighted subgraphs. SIAM J. Comput. **42**(3), 831–854 (2013). DOI 10.1137/09076619X. URL http://dx.doi.org/10.1137/09076619X
54. Yoshida, Y., Yamamoto, M., Ito, H.: Improved constant-time approximation algorithms for maximum matchings and other optimization problems. SIAM J. Comput. **41**(4), 1074–1093 (2012). DOI 10.1137/110828691. URL http://dx.doi.org/10.1137/110828691

A Useful Inequalities

This section provides standard equalities that we use throughout our paper. These inequalities exist in many variations, but here we only present the formulations which are most convenient for our purposes.

Theorem 10 *(Chebyshev's Inequality) For any random variable X and $a > 0$,*

$$\mathrm{P}[|X - \mathrm{E}[X]| \geq a] \leq \frac{\mathrm{Var}[X]}{a^2}$$

Theorem 11 *(Markov's Inequality) For any non-negative random variable X and $a > 0$,*

$$\mathrm{P}[X \geq a] \leq \frac{\mathrm{E}[X]}{a}.$$

Theorem 12 *(Chernoff Bound) Let $X_1, \cdots, X_n$ be independent Bernoulli random variables such that $\mathrm{P}[X_i = 1] = p$ for all $i \in [n]$, and let $X = \frac{1}{n}\sum_{i=1}^{n} X_i$. Then for any $0 < \delta \leq 1$,*

$$\mathrm{P}[X < (1-\delta)p] < e^{-\delta^2 pn/2}.$$

Theorem 13 *(Jensen's Inequality) For any real convex function f with $x_1, \cdots, x_n$ in its domain,*

$$\sum_{i=1}^{n} f(x_i) \geq nf\left(\sum_{i=1}^{n} x_i\right)$$