UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

PhD thesis

Randomized Algorithms for Large Datasets

Graphs, Streams, and Sums: The Sublinear Trilogy

Jakub Tětek

Advisor: Mikkel Thorup

Submitted: 31.07.2024

This thesis has been submitted to the PhD School of The Faculty of Science, University of Copenhagen

Abstract

This thesis addresses several challenges in efficiently processing large datasets and in privacy-preserving computation. Specifically, in this thesis, we look into the following problems.

First, we focus on the problem of estimating the sum of numbers under the assumption that we may sample the numbers proportionally to their values.

Second, we focus on several related problems in sublinear-time computation on graphs. We investigate several different models which formalize the setting where we have a very large graph and we want to perform some computation on it without even reading all of it. We consider several problems, namely sampling edges uniformly, counting edges, and counting triangles.

Finally, we consider the well-known Misra-Gries sketch for the problem of approximately finding frequent elements in a stream. We make this sketch differentially private while losing only a minimal amount of accuracy.

Resumé

Denne afhandling omhandler forskellige udfordringer inden for effektiv behandling af store datasæt samt privatlivsbevarende beregninger. Specifikt undersøger vi følgende problemer. Først ser vi på estimering af summen af en talmængde under antagelse af at hvert tal fra mængden kan udvælges med sandsynlighed proportionel til dets værdi. Dernæst fokuserer vi på en række relaterede grafproblemer som kan beregnes i sublineær tid. Vi undersøger forskellige modeller til at formalisere situationer hvor vi ønsker at udregne egenskaber af en meget stor graf uden at skulle indlæse hele grafen først. Vi beskæftiger os med problemer omhandlende at implementere uniform udvælgelse af kanter, at tælle kanter, samt at tælle trekanter. Slutteligt betragter vi den velkendte Misra-Gries skitse, som bruges til at finde de oftest forekommende elementer i en strøm. Vi gør skitsen differentielt privat med minimal indvirkning på dens nøjagtighed.

Acknowledgements

I would like to thank my parents, without whom I would not be writing this thesis, especially my mother who has done for my education more than anyone else. I would like to thank everyone who inspired me on my way to becoming a researcher in computer science, most notably Jirka Benc, Martin Mareš, and Tomáš Gavenčiak.

I am also very grateful to the people who have hosted me during the numerous research stays that I completed over the course of my PhD: Ronnit Rubinfeld at MIT, Talya Eden at Bar-Ilan University, Robert Krauthgamer at Weizmann Institute, Pavel Veselý at Charles University, Badih Ghazi at Google, and Olek Łukaszewicz at the University of Wrocław.

I would like to thank Rasmus Pagh. While he was not officially my supervisor, I was very often in his office telling him about an exciting idea I had the previous evening. I have learned a lot from these conversations and I am very grateful that Rasmus's doors were always open.

However, first and foremost, I would like to thank my supervisor Mikkel Thorup. I could not have wished for a better supervisor: He provided steadfast encouragement when I needed it most, helped with his wisdom and advice when I encountered obstacles, and gave me the autonomy to explore ideas independently. His invaluable advice was crucial for my development as a scientist. I am truly grateful for the opportunity to work under his mentorship.

Contents

Chapter 1

Introduction

With the continued increase in the amount of available data, the question of how to process it efficiently is more important than ever. Unfortunately, due to a lack of time and effort, I did not quite manage to solve this problem. Instead, in this thesis, I solve several specific problems of this nature.

In this thesis, we will look at how to perform computation on a graph without even reading all of it. We also focus on estimating sums of numbers under the assumption that we may sample the numbers from a specific natural distribution. We also focus on streaming algorithms for discovering the most common elements – heavy hitters – in a stream, while protecting users' privacy by relying on the differential privacy framework.

This thesis is a compilation thesis and it consists of four papers together with their summaries which now follow. Specifically, the papers in this thesis are the following:

- In the first paper (Beretta and Tětek [2024]), we consider estimating a sum of numbers in the setting where we may sample the numbers proportionally their value. We show that if we have n numbers, we may estimate their sum in roughly $O(\sqrt{n})$ samples.

- In the second of these papers (Tětek and Thorup [2022a]), we focus on sublinear-time computation on graphs. Specifically, we focus on the following problems: sampling an edge in sublinear time given "vertex access" to the graph, counting the number of edges, as well as counting the number of triangles. We consider these problems in four different natural settings which formalize the notion of "vertex access".

- In the third paper (Eden et al. [2023]), we improve upon the edge sampling procedure from Tětek and Thorup [2022a]. This method allows us to sample an edge exactly uniformly in sublinear time in complexity in which our previous results could only achieve constant-factor approximation of uniform.

- The last paper[1] (Janos Lebeda and Tetek [2024]) focuses on differential privacy in the streaming setting. Specifically, in that paper, we show how to make the Misra-Gries sketch – a commonly used sketch for finding frequent items in a stream – differentially private without significant deterioration of performance.

Another paper that I wanted to include as it would fit nicely, but couldn't, is the paper (Tětek [2022]) on approximately counting the number of triangles in a graph by combining sampling and fast matrix multiplication. However, then I recalled that I already handed this paper in as my Master's thesis, and this thesis will sadly have to do without it. However, a reader who enjoys the papers in this thesis will likely also be interested in that paper. But now, without further ado, let me introduce the four papers that I was able to include.

Better Sum Estimation via Weighted Sampling

In this paper, we considered the problem of estimating the sum of numbers – or rather estimating the sum of weights of items – under the assumption that we may sample the items either *(1)* proportionally to their values, or *(2)* that we may sample both proportionally and uniformly. Formally, we assume that we have a set U of items $a \in U$ that each has a value $w(a)$. Let $n = |U|$. The goal is to estimate $W = \sum_{a \in U} w(a)$, where we may sample the items with the probability of sampling a being $w(a)/W$. We assume that if we sample two items, we may check whether both samples are the same item or not (note that there may be multiple items with the same weight). We call this the proportional setting and the setting where we may also sample uniformly we call hybrid.

In the paper, we talk about a few examples of this setting, but here I mention that the famous Metropolis-Hastings algorithm allows precisely for this kind of proportional sampling.

This problem has previously been considered by Motwani et al. [2007]. Under the assumption that they know n, they give an algorithm with sample complexity $\tilde{O}(\sqrt{n}/\varepsilon^{7/2})$ in the proportional setting and $\tilde{O}(\sqrt[3]{n}/\varepsilon^{9/2})$ in hybrid, and they prove this is near-optimal in terms of the dependency on n. In our paper, we improve this to $O(\sqrt{n}/\varepsilon)$ and $O(\sqrt[3]{n}/\varepsilon^{4/3})$. We give matching and near-matching lower bounds, including the dependency on ε. We also consider the case when n is not known.

Another advantage of our approach is that some of our algorithms are much simpler than those in the original paper. We now sketch the algorithm for the proportional setting with known n. The algorithm works as follows.

[1] We refer here to the conference version of the paper. However, the version in this thesis corresponds to a journal version which is currently under review.

We let $S = \{a_1, \cdots, a_{|S|}\}$ be the multiset of $\Theta(\sqrt{n}/\varepsilon)$ samples and for each $s \in U$, we define c_s to be the number of times it has been sampled. We then define

$$\hat{W} = \binom{m}{2} \cdot \left(\sum_{s \in S} \frac{\binom{c_s}{2}}{w(s)} \right)^{-1} .$$

I do not show the full analysis here, but I at least try to give some intuition. Instead of working directly with the random variable $\hat{W}$, we consider and analyze the value $1/\hat{W}$ and show that it is a good approximation of $1/W$. This then implies that $\hat{W}$ is a good approximation of W. We now show that $E[1/\hat{W}] = 1/W$. The analysis of the variance can be found in the paper.

We have

$$\hat{W}^{-1} = \binom{m}{2}^{-1} \cdot \sum_{s \in S} \frac{\binom{c_s}{2}}{w(s)} .$$

We now look at the sum. For each $s \in S$, we add to the sum $\binom{c_s}{2}/w(s)$. We consider the collisions between sampled items (that is, the pairs of samples that are equal). The term $\binom{c_s}{2}$ is then the number of collisions that happen on the item s. We define $Y_{i,j} = 1/w(a_i)$ if $a_i = a_j$ and we let $Y_{i,j} = 0$ otherwise. We then thus have

$$\begin{aligned} E\left[\binom{m}{2}^{-1} \cdot \sum_{s \in S} \frac{\binom{c_s}{2}}{w(s)}\right] &= E\left[\binom{m}{2}^{-1} \cdot \sum_{i \neq j \in [|S|]} Y_{i,j}\right] \\ &= \binom{m}{2}^{-1} \cdot \sum_{i \neq j \in [|S|]} E[Y_{i,j}] = E[Y_{1,2}] , \end{aligned}$$

where the last step uses that all variables $Y_{i,j}$ are identically distributed. But the expectation of $Y_{i,j}$ is easy to compute:

$$E[Y_{i,j}] = \sum_{a \in U} P[a_i = a_j = a]/w(a) = \sum_{a \in U} \frac{w(a)^2}{W^2}/w(a) = \sum_{a \in U} \frac{w(a)}{W^2} = 1/W .$$

This gives us that

$$E[1/\hat{W}] = 1/W .$$

This looks reasonable – if we could prove a bound on the variance (as we do in the paper), this would imply that the estimator gives a good relative approximation with good probability.

Edge Sampling and Graph Parameter Estimation via Vertex Neighborhood Accesses

In this paper, we investigate various graph problems in various settings of sublinear computation. The setting most commonly used in sublinear-time graph algorithms allows the following operations: (1) given an index i, we may request the i-th vertex of the graph, (2) given a vertex and an index j, we may request its j-th neighbor, and (3) given a vertex, we may request its degree. We call this the indexed neighbor model. While this is a reasonable model, it is far from the only one that one may wish to use. We suggest two other access models and explore the complexity of several problems in them. Namely, we consider the problem of estimating the number of edges in a graph, sampling edges, and estimating the number of triangles in a graph. Now, we discuss the models that we use and show how the complexities of the above problems compare between the different settings.

Hash-ordered access

One possible addition to the indexed neighbor model is that we may assume the neighbors of a vertex are not ordered adversarially, but they are ordered with respect to a fully random hash function. That is, we have a hash function $h : V \to [0,1]$ and each neighborhood is ordered increasingly with respect to h. While this may seem rather arbitrary, the motivation is that this interface can be efficiently implemented in practice and it allows for more efficient algorithms.

In this setting, we consider the problem of estimating the number of edges in a graph. An algorithm with complexity $O(\frac{n}{\varepsilon^2\sqrt{m}})$ was known (Seshadhri [2015]), as well as a lower bound of $\Omega(\frac{n}{\sqrt{\varepsilon m}})$ Goldreich and Ron [2008]. We improve the lower bound to $\Omega(\frac{n}{\varepsilon\sqrt{m}})$ as long as ε is not too small, and this lower bound works even in the stronger "full neighborhood access" model (see below). We then give a more efficient algorithm in the hash-ordered access setting with complexity $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}} + \frac{1}{\varepsilon^2})$. Note that this is near-optimal for ε being not too small, thanks to our improved lower bound. This is in contrast with the previous results, which only had (near-)optimal complexity in terms of n and m.

Inspired by this algorithm, we also give an algorithm that uses the more standard assumption that, in addition to the three standard queries (1)-(3), we may also use a query that answers the question "are vertices u and v adjacent" where u and v can be any two vertices that we specify. This algorithm runs in time $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}} + \frac{1}{\varepsilon^4})$ which is again near-optimal when ε is not too small.

At the core of our edge-counting algorithm in the hash-ordered setting is an algorithm for uniformly sampling s edges in the same setting. This algorithms returns s edges sampled independently uniformly in time $\tilde{O}(\sqrt{s}\frac{n}{\sqrt{m}} + s)$. This is the same complexity in which the previous work (Eden et al. [2021b]) in the indexed neighbor model achieved ε-approximate sampling for constant ε. As a side-note, we also improve the algorithm for ε-approximately sampling one edge in the indexed neighbor model from $O(\frac{n}{\sqrt{\varepsilon m}})$ (Eden and Rosenbaum [2018c]) to $O(\frac{n}{\sqrt{m}} \log \varepsilon^{-1})$.

Full neighborhood access

Another natural model choice is to assume that when we query a vertex, we get the (unique identifiers of) all adjacent vertices. While this may not seem very realistic, we argue that in many settings, this is quite a reasonable assumption, apart from also being a very simple and clean model. For example, if the graph is stored on a hard drive, accessing one bit of information does not cost much less than accessing several megabytes. However, vertices rarely have such a high degree in practice that its neighborhood could not be stored in that amount of space.

In this setting, we give a better algorithm for estimating the number of triangles in a graph. In the indexed neighbor setting, an algorithm was known (Eden et al. [2017b]) that ran in time $\tilde{O}(\frac{n}{\varepsilon^{10/3}T^{1/3}} + \frac{m^{3/2}}{\varepsilon^3 T})$, where T is the number of triangles and this was known to be optimal in terms of the dependency on n, m, T. Using the power of the full neighborhood access setting, we managed to improve this significantly to $\tilde{O}(\frac{n}{\varepsilon T^{1/3}} + \frac{\sqrt{nm}}{\varepsilon^2 T})$. We also proved that this is near-optimal in terms of n, m, T.

Sampling an Edge in Sublinear Time Exactly and Optimally

In this paper, we improve upon the algorithm in the indexed neighbor setting for sampling one edge that we gave with my supervisor Mikkel Thorup in the previous paper. Namely, instead of being able to sample pointwise $1+\varepsilon$ close to uniform, we sample *exactly* uniformly in the optimal expected complexity $O(n/\sqrt{m})$. I now try to sketch the basic technique.

The previous work (Eden and Rosenbaum [2018c]) shows that if the degrees are bounded by Δ, then one may sample an edge in time $O(\Delta)$. They then used this technique for the vertices with small degrees (light vertices) while using an alternative approach for the high-degree vertices (heavy vertices) – this "alternative approach" is the crux of the problem. It can be easily seen that the problem of sampling an edge uniformly is equivalent to sampling a vertex proportionally to its degree; that is, sampling w with probability $\frac{d(w)}{2m}$. Specifically, the forward implication (the one we need) holds for the following reason: if we sample a vertex from this distribution and return a random incident edge, this edge is picked uniformly at random from the set of all edges of the graph.

A simple counting argument shows that if we set $\Delta = 100\sqrt{m}$, then any heavy vertex can only have a small constant fraction of its neighbors that are also heavy. If we uniformly sample an edge uv (in the analysis, we do not know how to do this yet!), the probability that u is light and $v = w$ for a fixed heavy vertex w is equal to $c \cdot \frac{d(w)}{2m}$ where c is the proportion of neighbors of w that are light. Using this fact, we use the approach for sampling edges incident to light vertices, in order to sample any heavy vertex w with this probability. If we knew c, we could easily sample an edge uniformly – we would return the sampled vertex with probability, say, $1/(2c)$ which results in sampling the vertex w with probability $\frac{1}{2} \cdot \frac{d(w)}{2m}$ as we wanted (in the framework of Eden and Rosenbaum [2018c], one can easily adjust the sampling probabilities by a fixed constant factor so the factor $\frac{1}{2}$ does not cause issues).

While this is a nice observation, it may not seem very useful, since we do not know c. However, we manage to work around this issue. The main idea is that we do not need to know c, we just need to be able to somehow generate a Bernoulli trial with probability $1/(2c)$. At the same time, note that it is easy to sample a Bernoulli trial with probability c – we just sample a neighbor of w and we check whether it is light. Luckily for us, there is a general result that states that under mild technical assumptions, if we have access to i.i.d. samples from $Bern(p)$, then we may simulate a trial from $Bern(f(p))$ for a function f in expected $O(1)$ samples from $Bern(p)$ (Nacu and Peres [2006]). But this is exactly what we needed – this allows us to generate samples from the desired distribution $Bern(1/(2c))$ just by relying on access to samples $Bern(c)$ which we can easily provide. Using this with the above approach gives us the desired algorithm.

Better Differentially Private Approximate Histograms and Heavy Hitters using the Misra-Gries Sketch

In this paper, we have shown how the commonly used Misra-Gries sketch (Misra and Gries [1982]) for the problem of heavy hitters can be made differentially private. First, I introduce the notion of differential privacy and the Misra-Gries sketch. Then I explain our contribution.

Preliminaries on differential privacy

Differential privacy (Dwork et al. [2006a]) concerns the setting where we want to publicly release some function of a dataset that contains private information. The worry is that especially if we release multiple such values, an attacker might reconstruct parts of the private dataset based on the published information. Indeed, there have been examples of this happening Narayanan and Shmatikov [2006], Barth-Jones [2012]. Differential privacy is a privacy notion that gives strong probabilistic guarantees on the impossibility of such attacks.

More formally, we assume we have a symmetric relation $x \sim x'$ of the set of all possible datasets (possible inputs). Intuitively speaking, this usually corresponds to “the datasets x and x' differ in the data of one user” where the database consists of records of users. Then, for a privacy parameter $\varepsilon > 0$, we say that an algorithm A is ε-differentially private if

$$P[A(x) \in S] \leq e^{\varepsilon} P[A(x') \in S]$$

for any measurable set S. Similarly, ε, δ-differential privacy assumes that

$$P[A(x) \in S] \leq e^{\varepsilon} P[A(x') \in S] + \delta \, .$$

We will not need to rely directly on this definition here. Instead, it will suffice to know that if we define the global sensitivity $GS_f = \sup_{x \sim x'} \|f(x) - f(x')\|_1$, then releasing the value $f(x) + \mathit{Laplace}(GS_f/\varepsilon)$ is ε-differentially private.

An interested reader wishing for an introduction to differential privacy will be well-served by the very short introduction by Near and Abuah [2021].

The Misra-Gries sketch

The Misra-Gries sketch is a sketch for finding the frequent items in a stream of items and approximating their frequencies. If the sketch has size k, the error in the estimated frequency of any item is at most $\frac{m}{k+1}$ where m is the length of the stream. This is known to be optimal among deterministic algorithms (Bose et al. [2003]). The sketch works as follows. It stores at most k key-value pairs, which intuitively correspond to the (approximate) most common elements and their estimated counts. Whenever some value is equal to 0, we remove this pair. Inserting an item is performed as follows: if the key is already present, we increment its counter by 1, otherwise we add a new key-value pair with the counter equal to 1. If this resulted in us having $> k$ pairs, we subtract 1 from all counters (note that this necessarily removes the item that we just inserted, so we will again have $\leq k$ key-value pairs).

Our contribution

How do we make the Misra-Gries sketch differentially private? Note that the global sensitivity is k, since subtracting 1 from all counters results in a total change of k. If we naively use the Laplace noise, this will thus result in the rather large error of $O(k/\varepsilon)$. Annoyingly, this increases as the sketch gets bigger, which is exactly the opposite of what we would want (bigger sketches should result in lower error). I now sketch a simple way to get around this, in the paper we give a more direct analysis which results in better constants.

We come up with an alternative representation of the k counters. Namely, we have one more counter c, that stores the total number of decrements. That is, instead of decrementing all the counters like in the original algorithm, we will increase this counter c. When we insert a new item, we sets its value to be c. Then if we subtract c from each counter at the end, we get the exact same sketch that we would get if we used the standard algorithm. (The rule for when keys are removed from the sketch has to be modified accordingly).

The advantage of this representation is that its ℓ_1 sensitivity can be proved to be ≤ 2. This is easy to see if the two neighboring inputs (from the definition of differential privacy) differ in the last item from the input. By induction, we prove that this holds in general. This means that adding Laplace noise of magnitude $2/\varepsilon$ to each counter in the universe (the ones not represented in the sketch are treated implicitly as 0) ensures ε-differential privacy. This algorithm in turn can be seen to be equivalent to using the standard Misra-Gries sketch and adding two noises as follows: independent Laplacian noise to each counter and then one identical Laplacian noise to all counters.

However, this algorithm has to add noise to every item in the universe, which results in a high maximum noise value and is hard to implement efficiently. Note that we cannot just add noise to the elements stored in the sketch as the set of stored items can differ between neighboring inputs, and we cannot afford to add noise to a counter on one input and not on the other, that would violate privacy. We get around this issue by relying on ε, δ-differential privacy. Namely, we set a threshold $\tau = O(\log(n\delta^{-1})/\varepsilon)$ and we throw away all items whose counter after adding noise is $< \tau$. Why does this help? The issue was that it might happen that the set of stored items is different between two neighboring inputs. We prove that if an item is in the sketch for only one of two neighboring inputs, its count has to be 1. But then the probability that we release any of such items is $\leq \delta$ since that would require the noise to have magnitude at least $\tau - 1$ which happens only with probability δ. But, intuitively speaking, if we do not release these items, they also cannot leak any private information, thus guaranteeing ε, δ-differential privacy.

Chapter 2

Better Sum Estimation via Weighted Sampling

Lorenzo Beretta
lorenzo2beretta@gmail.com
BARC, Univ. of Copenhagen

Jakub Tětek
j.tetek@gmail.com
BARC, Univ. of Copenhagen

Abstract

Given a large set U where each item $a \in U$ has weight $w(a)$, we want to estimate the total weight $W = \sum_{a \in U} w(a)$ to within factor of $1 \pm \varepsilon$ with some constant probability $> 1/2$. Since $n = |U|$ is large, we want to do this without looking at the entire set U. In the traditional setting in which we are allowed to sample elements from U uniformly, sampling $\Omega(n)$ items is necessary to provide any non-trivial guarantee on the estimate. Therefore, we investigate this problem in different settings: in the *proportional* setting we can sample items with probabilities proportional to their weights, and in the *hybrid* setting we can sample both proportionally and uniformly. These settings have applications, for example, in sublinear-time algorithms and distribution testing.

Sum estimation in the proportional and hybrid setting has been considered before by Motwani, Panigrahy, and Xu [ICALP, 2007]. In their paper, they give both upper and lower bounds in terms of n. Their bounds are near-matching in terms of n, but not in terms of ε. In this paper, we improve both their upper and lower bounds. Our bounds are matching up to constant factors in both settings, in terms of both n and ε. No lower bounds with dependency on ε were known previously. In the proportional setting, we improve their $\tilde{O}(\sqrt{n}/\varepsilon^{7/2})$ algorithm to $O(\sqrt{n}/\varepsilon)$. In the hybrid setting, we improve $\tilde{O}(\sqrt[3]{n}/\varepsilon^{9/2})$ to $O(\sqrt[3]{n}/\varepsilon^{4/3})$. Our algorithms are also significantly simpler and do not have large constant factors.

We then investigate the previously unexplored scenario in which n is not known to the algorithm. In this case, we obtain a $O(\sqrt{n}/\varepsilon + \log n/\varepsilon^2)$ algorithm for the proportional setting, and a $O(\sqrt{n}/\varepsilon)$ algorithm for the hybrid setting. This means that in the proportional setting, we may

remove the need for advice without greatly increasing the complexity of the problem, while there is a major difference in the hybrid setting. We prove that this difference in the hybrid setting is necessary, by showing a matching lower bound.

Our algorithms have applications in the area of sublinear-time graph algorithms. Consider a large graph $G = (V, E)$ and the task of $(1 \pm \varepsilon)$-approximating $|E|$. We consider the (standard) settings where we can sample uniformly from E or from both E and V. This relates to sum estimation as follows: we set $U = V$ and the weights to be equal to the degrees. Uniform sampling then corresponds to sampling vertices uniformly. Proportional sampling can be simulated by taking a random edge and picking one of its endpoints at random. If we can only sample uniformly from E, then our results immediately give a $O(\sqrt{|V|}/\varepsilon)$ algorithm. When we may sample both from E and V, our results imply an algorithm with complexity $O(\sqrt[3]{|V|}/\varepsilon^{4/3})$. Surprisingly, one of our subroutines provides an $(1 \pm \varepsilon)$-approximation of $|E|$ using $\tilde{O}(d/\varepsilon^2)$ expected samples, where d is the average degree, under the mild assumption that at least a constant fraction of vertices are non-isolated. This subroutine works in the setting where we can sample uniformly from both V and E. We find this remarkable since it is $O(1/\varepsilon^2)$ for sparse graphs.

Introduction.

Suppose we have a large set U, a weight function $w : U \to [0, \infty)$ and we want to compute a $(1 \pm \varepsilon)$-approximation of the sum of all weights $W = \sum_{a \in U} w(a)$. Since $n = |U|$ is very large, we want to estimate W by sampling as few elements as possible. In the traditional setting in which we are allowed to sample elements from U uniformly, sampling $o(n)$ items cannot provide any non-trivial guarantee on the approximation as we may miss an element with a very large weight. This led Motwani et al. [2007] to study this problem when we are allowed to sample elements proportionally to their weights (i.e., sample a with probability $w(a)/W$). In particular, they studied two settings: the *proportional* setting, where we can sample items proportionally to their weights, and the *hybrid* setting where both proportional and uniform sampling is possible. In this paper, we revisit these two settings and get both improved lower and upper bounds. We also extend the results to more general settings and show how our techniques imply new results for counting edges in sublinear time.

Motwani et al. [2007] give upper and lower bounds for both proportional and hybrid settings. Their bounds are matching up to polylogarithmic factors in terms of n, but not in terms of ε. In this paper, we improve both their upper and lower bounds. Our bounds are matching up to polylogarithmic factors in both settings, in terms of both n and ε. Moreover, for wide parameter regimes our bounds are matching up to *constant* factors. No lower bounds with dependency on ε were known previously.

In the proportional setting, we improve their $\tilde{O}(\sqrt{n}/\varepsilon^{7/2})$ algorithm to $O(\sqrt{n}/\varepsilon)$. In the hybrid setting, we improve $\tilde{O}(\sqrt[3]{n}/\varepsilon^{9/2})$ to $O(\sqrt[3]{n}/\varepsilon^{4/3})$. Our algorithms are also significantly simpler. In the same paper, Motwani et al. [2007] write: "to efficiently derive the sum from [proportional] samples does not seem straightforward". We would like to disagree and give a formula that outputs an estimate of W from a proportional sample. This formula is not only optimal in terms of sample complexity but also very simple. Our other algorithms, although not as simple, do not have large hidden constants and we believe they are both practical and less involved than their predecessors.

In their work, Motwani et al. [2007] always assume to know the size of the universe $n = |U|$. We extend these results to the case of unknown n. In this case, we obtain a $O(\sqrt{n}/\varepsilon + \log n/\epsilon^2)$ algorithm for the proportional setting, and a $O(\sqrt{n}/\varepsilon)$ algorithm for the hybrid setting. This means that in the proportional setting we may remove the need for advice without significantly impacting the complexity of the problem, while there is a major difference in the hybrid setting. We prove that this difference in the hybrid setting is necessary, by showing a matching lower bound.

We give lower bounds for all our estimation problems, both for proportional and hybrid settings, and when n is either known or unknown. This is the most technically challenging part of the paper. We prove lower bounds for n known and unknown as well as proportional and hybrid settings; all our lower and upper bounds are matching up to a constant factor. See Table 2.1 for a summary of our results.

Our algorithms have particularly interesting applications in the area of sublinear-time graph algorithms, which are explained in detail in Chapter 2.

The paper is structured as follows. In what is left of Chapter 2 we explain applications and related work, give an overview of the techniques employed, and provide the reader with formal definitions of problems and notation. Chapter 2 contains our algorithms for proportional setting, while Line 8 contains algorithms for hybrid setting. In Line 3 we prove all our lower bounds. In Line 3 we show how to apply our algorithms to the problem of counting the number of edges in a graph in sublinear time. In Line 3 we raise several open problems.

Related Work and Applications.

In this section, we show that our algorithms can be applied to get sublinear-time graph algorithms and distribution testing and discuss how this relates to previous work in these areas. We also discuss here how the widely used Metropolis-Hastings algorithm in fact implements the proportional sampling. We suggest that this could be an application domain worth investigating.

Advice to the algorithm	This paper		Motwani et al. [2007]	
	Proportional	Hybrid	Proportional	Hybrid
n known	$\Theta(\sqrt{n}/\varepsilon)$	$O(\min(\sqrt[3]{n}/\varepsilon^{4/3}, n\log n))$ $\Omega(\min(\sqrt[3]{n}/\varepsilon^{4/3}, n)$	$\tilde{O}(\sqrt{n}/\varepsilon^{7/2})$ $\Omega(\sqrt{n})$	$\tilde{O}(\sqrt[3]{n}/\varepsilon^{9/2})$, $O(\sqrt{n}/\varepsilon^2)$ $\Omega(\sqrt[3]{n})$
Known $\tilde{n} \geq n$	$O(\sqrt{\tilde{n}}/\varepsilon)$ $\Omega(\sqrt{n}/\varepsilon)$	$O(\min(\sqrt{n}/\varepsilon, n\log n))$ $\Omega(\min(\sqrt{n}/\varepsilon, n))$		
No advice	$O(\sqrt{n}/\varepsilon + \log n/\epsilon^2)$ $\Omega(\sqrt{n}/\varepsilon)$	$O(\min(\sqrt{n}/\varepsilon, n\log n))$ $\Omega(\min(\sqrt{n}/\varepsilon, n))$		

Table 2.1: Results of this paper.

Counting edges in sublinear time.

When a graph $G = (V, E)$ is very large, we may want to approximately solve certain tasks without looking at the entire G, thus having a time complexity that is sublinear in the size of G. In particular, estimating global properties of G such $|V|$ or $|E|$ in this setting is an important problem and has been studied in both theoretical and applied communities Feige [2006a], Goldreich and Ron [2006], Seshadhri [2015], Eden et al. [2017a], Tětek and Thorup [2022], Katzir et al. [2011], Dasgupta et al. [2014a]. Since the algorithm does not have the time to pre-process (or even see) the whole graph, it is important to specify how we access G. Several models are employed in the literature. The models differ from each other for the set of queries that the algorithm is allowed to perform. A *random vertex query* returns a random vertex, a *random edge query* returns a random edge, a *pair query* takes two vertices u, v as arguments and returns $(u, v) \in E$, a *neighborhood query* takes a vertex v and an index i as arguments and returns the i-th neighbor of v (or says that $d(v) < i$), a *degree query*, given a vertex v, returns its degree $\deg(v)$. We parameterize the complexities with an approximation parameter ε, $n = |V|$ and $m = |E|$.

The problem of estimating the number of edges in a graph in sublinear time has been first considered by Feige [2006a]. Their algorithm works in the model where only random vertex queries and degree queries are allowed and achieves a $(2+\varepsilon)$-approximation algorithm using $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}})$ time and queries. Their algorithm does not use neighborhood queries and the authors showed that without neighborhood queries, $2-\varepsilon$ approximation requires a linear number of queries. Goldreich and Ron [2006] broke the barrier of factor 2 by using neighborhood queries. Indeed, they showed a $(1+\varepsilon)$-approximation with time and query complexity of $\tilde{O}(\frac{n}{\varepsilon^{9/2}\sqrt{m}})$. Currently, the best known algorithm is the one by Eden et al. [2017a] and has complexity $\tilde{O}(\frac{n}{\varepsilon^2\sqrt{m}})$. If pair queries are allowed, the algorithm of Tětek and Thorup [2022] has complexity $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}} + \frac{1}{\varepsilon^4})$; in the same paper the authors showed a lower bound which is near-matching for $\varepsilon \geq m^{1/6}/n^{1/3}$ as well as an algorithm with complexity $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}} +$

$\frac{1}{\varepsilon^2}$) in what they call the hash-ordered access model. They also show an algorithm in the more standard setting with random vertex and neighborhood queries, that runs in time $\tilde{O}(\sqrt{n}/\varepsilon)$. This is the same complexity (up to a $\log^{O(1)} n$ factor) that we achieve in the setting with random edge queries, as we discuss below. Our techniques are, however, completely different and share no similarity with the techniques used by Tětek and Thorup [2022].

Our algorithms can be applied to solve the edge counting problem when either (i) random edge queries only are allowed, or (ii) both random edge and random vertex queries are allowed. We remark that uniform edge sampling can be (approximately) simulated using $\tilde{O}(n/\sqrt{\varepsilon m})$ random vertex queries, degree queries and neighbor queries (Eden and Rosenbaum [2018c]). While edge counting has not been explicitly considered in these settings before, these settings are established and have been used in several papers (Assadi et al. [2019b], Aliakbarpour et al. [2018a], Fichtenberger et al. [2020a], Biswas et al. [2021b]). We instantiate our algorithm for this graph problem setting $U = V$ and $w(v) = \deg(v)$ for each $v \in V$. Uniform sampling then corresponds to sampling vertices uniformly. Proportional sampling can be simulated by taking a random edge and picking one of its endpoints at random. Degree query allows us to get the weights of sampled vertices. Since these settings have not been explicitly studied before, we compare our results with what follows directly from the known literature.

If we can only sample uniformly from E, the algorithm by Motwani et al. implies an algorithm for this problem that has complexity $\tilde{O}(\sqrt{n}/\varepsilon^{7/2})$. Using an algorithm from Eden et al. [2017a] and standard simulation of random vertex queries using random edge queries, one would get time $\tilde{O}(\sqrt{m}/\varepsilon^2 + \frac{m}{\varepsilon\sqrt{n'}})$ for n' being the number of non-isolated[1] vertices[2]. Our results immediately give a $O(\sqrt{n}/\varepsilon)$ algorithm, or $\tilde{O}(\sqrt{n}/\varepsilon + 1/\varepsilon^2)$ when n is not known.

When we may sample both from E and V, the algorithms by Motwani et al. imply algorithms with sample complexities of $\tilde{O}(\sqrt[3]{n}/\varepsilon^{9/2})$ and $O(\sqrt{n}/\varepsilon^2)$. We may also use the algorithm of Eden et al. [2017a] that relies on random vertex query only. This has complexity from $O(\frac{n}{\varepsilon^2\sqrt{m}})$. Our results imply an algorithm with complexity $O(\sqrt[3]{n}/\varepsilon^{4/3})$. Surprisingly, one of our subroutines provides an $(1 \pm \varepsilon)$-approximation of $|E|$ using $\tilde{O}(d/\varepsilon^2)$ expected samples, where d is the average degree, under the mild assumption that at least a constant fraction of vertices are non-isolated (in fact, we prove a more complicated complexity which depends on the fraction of vertices that are isolated). This subroutine works in the setting where we can sample uniformly from both V and E. We find this interesting since it is $O(1/\varepsilon^2)$ for sparse graphs.

[1] A vertex is isolated if it has degree zero.

[2] One may simulate uniform sampling from the set of non-isolated vertices at multiplicative overhead of $O(m/n')$ by sampling proportionally and using rejection sampling. We may then use set size estimation by birthday paradox in time $O(\sqrt{n'}/\epsilon)$ to learn n' and the algorithm of Eden et al. [2017a].

Distribution testing.

Consider a model in which we are allowed to sample from a distribution $\mathcal{D}$ on U, and when we do, we receive both $a \in U$ and $P_{\mathcal{D}}(a)$. This model is stronger than the one we considered throughout the paper. In fact, it is a special case of our model obtained by fixing $W = 1$. Similarly, we may consider such stronger variant of the hybrid setting. This model has received attention both in statistics and in theoretical computer science literature. Most notably, Horvitz and Thompson [1952b] in 1952 showed how to estimate the sum $\sum_{a \in U} v(a)$, for value function $v : U \to \mathbb{R}$ when having access to a sample a from $\mathcal{D}$ and $P_{\mathcal{D}}(a)$. More recently, many distribution testing problems have been considered in this and similar models.

First, Canonne and Rubinfeld [2014] considered the model where one can (i) sample $a \in U$ according to $\mathcal{D}$; (ii) query an oracle that, given $a \in U$, returns $P_{\mathcal{D}}(a)$. This allows us to both get the probability of a sampled item, but also the probability of any other item. For every permutation-invariant problem, any sublinear-time algorithm in this model may be transformed so that it only queries the oracle on previously sampled items or on items chosen uniformly at random [3]. This setting is stronger than our hybrid setting. However, the two models become equivalent if we know W.

Second, Onak and Sun [2018] considered exactly the model described above, where one can sample from a distribution $\mathcal{D}$, and both $a \in U$ and $P_{\mathcal{D}}(a)$ are returned. This setting is again stronger than our proportional setting, and it becomes equivalent once we know W.

In both of these papers, the authors solved several distribution testing problems such as: uniformity testing, identity testing, closeness testing, distance to a known distribution, and distance between two unknown distributions in the respective models. Canonne and Rubinfeld [2014] also considered the problem of providing an additive approximation of entropy. Both Canonne and Rubinfeld [2014] and Onak and Sun [2018] proved several lower bounds, showing that many of their algorithms are optimal up to constant factors. These lower bounds also imply a separation between the two models for some of the problems mentioned above.

In both of the papers, the authors make a point that many of their algorithms (all the ones we mentioned) are robust with respect to noise of multiplicative error $(1 \pm \varepsilon/2)$ in the answers of the probability oracles. Our algorithms can then serve as a reduction from the weaker models we consider in this paper to these stronger models where we know the probabilities and not just the weights. The reason is that we can get an approximation of

[3] Since we consider permutation invariant problems, we may randomly permute the elements. Whenever we query an element that has not yet been sampled, we may assume that it is sampled uniformly from the set of not-yet-sampled elements. This can be simulated by sampling elements until getting a not-yet-seen item (sublinearity ensures that the step adds multiplicative $O(1)$ overhead).

W, meaning that we may then approximately implement the above-described models. Since the mentioned algorithms (Canonne and Rubinfeld [2014], Onak and Sun [2018]) are robust, a $(1 \pm \varepsilon/2)$-approximation of W is sufficient to simulate them.

Proportional sampling in practice: Metropolis-Hastings.

Proportional sampling is often implemented in practice using the Metropolis-Hasting algorithm. This algorithm is widely used in statistics and statistical physics, but can also be used to sample combinatorial objects. It can be used to sample from large sets which have complicated structures that make it difficult to use other sampling methods. Just like our algorithms, it is suitable when the set is too large to be stored explicitly, making it impossible to pre-process it for efficient sampling. One of the main appeals of Metropolis-Hasting is that it does not require one to know the exact sampling probabilities, but it is sufficient to know the items' weights like in the proportional setting described in this paper. We thus suggest that our algorithm could find practical applications in combination with the Metropolis-Hasting algorithm.

Overview of employed techniques.

Here we provide a summary of the techniques employed throughout the paper. We denote with $P_{unif}(\cdot)$ and $P_{prop}(\cdot)$ the probabilities computed according to uniform and proportional sampling, respectively. Although often not specified for brevity, all guarantees on the approximation factors of estimates in this section are meant to hold with probability $2/3$. These probability can be amplified to $1 - \delta$ through $O(\log 1/\delta)$ repetitions using the folklore median trick. We do not know whether the dependency w.r.t. δ is optimal and this would be an interesting further question.

Proportional setting with advice $\tilde{n} \geq n$.

Consider sampling two elements $a_1, a_2 \in U$ proportionally and define $Y_{12} = 1/w(a_1)$ if $a_1 = a_2$, and $Y_{12} = 0$ otherwise. It is easy to show that Y_{12} is an unbiased estimator of $1/W$. We could perform this experiment many times and take the average. This would give a good approximation to $1/W$ and taking the inverse value, we would get a good estimate on W. Unfortunately, we would need $\Theta(n/\varepsilon^2)$ repetitions in order to succeed with a constant probability. We can fix this as follows: we take m samples $a_1, \cdots, a_m$ and consider one estimator Y_{ij} for each pair of samples a_i, a_j for $i \neq j$. This allows us to get $\binom{m}{2}$ estimators from m samples. We show that estimators Y_{ij} are uncorrelated. This reduces the needed number of samples from $O(n/\varepsilon^2)$ to $O(\sqrt{n}/\varepsilon)$.

We now describe the estimator formally. Let $S = \{a_1, \cdots, a_m\}$ be the set of sampled items, and for each $s \in S$ define c_s to be the number of times item

s is sampled. Then,

$$\hat{W} = \binom{m}{2} \cdot \left(\sum_{s \in S} \frac{\binom{c_s}{2}}{w(s)} \right)^{-1}$$

is a $(1 \pm \varepsilon)$-approximation of W with probability 2/3 for $m = \Theta(\sqrt{n}/\varepsilon)$. If, instead of knowing n exactly, we know some $\tilde{n} \geq n$ we can achieve the same guarantees by taking $\Theta(\sqrt{\tilde{n}}/\varepsilon)$ samples.

Proportional setting, unknown n.

We partition U into buckets $B_i = \{a \in U \mid w(a) \in [2^i, 2^{i+1})\}$. We choose some $b \in \mathbb{Z}$ and compute two estimates: $\hat{W}_b$ approximates $W_b = \sum_{a \in B_b} w(a)$, and $\hat{P}_b$ approximates $P_b = P_{prop}(a \in B_b) = W_b/W$. We then return $\hat{W}_b/\hat{P}_b$, as an estimate of W. If both estimates are accurate, then the returned value is accurate. To choose b, we sample a_1 and a_2 proportionally and define b so that $\max\{a_1, a_2\} \in [2^b, 2^{b+1})$. We prove that, surprisingly[4], $E[1/P_b] = O(\log n)$. Computing $\hat{P}_b$ is simply a matter of estimating the fraction of proportional samples falling into B_b. This can be done using $O(1/(P_b\varepsilon^2))$ samples, where ε is the approximation parameter. Therefore the expected complexity of computing $\hat{P}_b$ is $O(\log n/\varepsilon^2)$. Computing $\hat{W}_b$ is more involved. First, we design two subroutines to sample from B_b, one for proportional sampling and one for uniform. These subroutines work by sampling $a \in U$ until we get $a \in B_b$; the subroutine for uniform sampling then uses rejection sampling. These subroutines take, in expectation, $O(1/P_b)$ samples to output one sample from B_b. Since we can now sample uniformly from B_b, we sample items uniformly form B_b until we find the first repeated item and use the stopping time to infer $|B_b|$ up to a constant factor. In expectation, we use $O(\sqrt{|B_b|})$ uniform samples and compute $\tilde{n}_b$, such that $|B_b| \leq \tilde{n}_b \leq O(|B_b|)$. Since we can also sample proportionally from B_b, we may use the algorithm for proportional setting with advice $\tilde{n}_b$ to estimate W_b. This yields a total sample complexity of $O(\sqrt{n}/\varepsilon + \log n/\varepsilon^2)$.

Hybrid setting.

We now present the techniques used in the hybrid setting. Before giving a sketch of the main algorithms, we introduce two subroutines.

Coupon-collector-based algorithm. In the hybrid setting, we can sample elements uniformly. If we know $n = |U|$ a well-known result under the name of "coupon collector problem" shows that we can retrieve with high probability all n elements by performing $\Theta(n \log n)$ uniform samples. We extend this

[4] Notice that, if we just define $b = a_1$, then we have $E[1/P_b] = n$ in the worst case where we have n distinct non-empty buckets. Therefore, taking the maximum of two samples entails an exponential advantage.

result to the case of unknown n. We maintain a set of retrieved elements $S \subseteq U$ and keep on sampling uniformly and adding new elements to S until we perform $\Theta(|S| \log |S|)$ samples in a row without updating S. It turns out that this procedure retrieves the whole set U with probability $2/3$ and expected complexity $O(n \log n)$. According to our lower bounds, this algorithm is near-optimal when $\varepsilon = O(1/\sqrt{n})$. Therefore, we can focus on studying hybrid sampling for larger values of ε.

Harmonic mean and estimating W/n with advice $\tilde{\theta} \geq W/n$ in hybrid setting. If we sample $a \in U$ proportionally, then $1/w(a)$ is an unbiased estimator of n/W. Unfortunately, we may have very small values of $w(a)$, which can make the variance of this estimator arbitrarily large. To reduce variance, we can set a threshold ϕ and define an estimator Y_ϕ as $1/w(a)$ if $w(a) \geq \phi$, and 0 otherwise. Set $p = P_{unif}(w(a) \geq \phi)$, then we have $E[Y_\phi] = pn/W$ and $\text{Var}(Y_\phi) \leq pn/(\phi W)$. If we define $\bar{Y}_\phi$ as the average of $\tilde{\theta}/(p\phi\varepsilon^2)$ copies of Y_ϕ, we have $\text{Var}(\bar{Y}_\phi) \leq (\varepsilon E[Y_\phi])^2$. With such a small variance, $\bar{Y}_\phi$ is a good estimate of pn/W. Estimating p is simply a matter of estimating the fraction of uniform samples having $w(\bullet) \geq \phi$. This can be done taking $O(1/(p\varepsilon^2))$ uniform samples. Once we have estimates of both p and pn/W we return their ratio as an estimate of W/n. We employed in total $O((1 + \frac{\tilde{\theta}}{\phi})/(p\,\varepsilon^2))$ proportional and uniform samples.

Perhaps surprisingly, this corresponds to taking the harmonic mean of the samples with weight at least ϕ and adjusting this estimate for the weight of the items with weight $< \phi$.

Hybrid setting, known n. We now sketch the algorithm for sum estimation in the hybrid setting with known n. We combine our formula to estimate W in the proportional setting and the above algorithm to estimate W/n in hybrid setting to obtain an algorithm that estimates W using $O(\sqrt[3]{n}/\varepsilon^{4/3})$ uniform and proportional samples. Define $U_{\geq\theta} = \{a \in U \mid w(a) \geq \theta\}$. We find a θ such that $P_{unif}(a \in U_{\geq\theta}) \approx n^{-1/3}\varepsilon^{-2/3}$, this can be done taking enough uniform samples and picking the empirical $(1 - n^{-1/3}\varepsilon^{-2/3})$-quantile[5] . We define $p = P_{prop}(a \in U_{\geq\theta}) \geq P_{unif}(a \in U_{\geq\theta}) \approx n^{-1/3}\varepsilon^{-2/3}$. We compute an estimate $\hat{p}$ of p counting the fraction of proportional samples falling in $U_{\geq\theta}$. Now we have two cases. If $\hat{p} \geq 1/2$, we can simulate sampling from the proportional distribution on $U_{\geq\theta}$ with $O(1)$ overhead by sampling proportionally until we get an element of $U_{\geq\theta}$. Then, we use the algorithm for sum estimation under proportional sampling restricted to elements in $U_{\geq\theta}$ to estimate $\sum_{a \in U_{\geq\theta}} w(a) = pW$. Dividing by the estimate $\hat{p} \approx p$, we get an estimate for W. Else, $\hat{p} < 1/2$, and thus $p \leq 2/3$ (assuming $\hat{p}$ is a good enough estimate

[5] We may assume that $\epsilon \geq 8/\sqrt{n}$ by running the coupon-collector-based algorithm when $\epsilon < 8/\sqrt{n}$. Under this assumption, it holds $n^{-1/3}\varepsilon^{-2/3} \in [0, 1]$ and taking the $(1 - n^{-1/3}\varepsilon^{-2/3})$-quantile then is meaningful.

of p). We then have $\theta n \geq \sum_{a \notin U_{\geq\theta}} w(a) = (1-p)W \geq W/3$. This allows us to use the harmonic-mean-based algorithm to estimate W/n with $\phi = \theta$ and $\tilde{\theta} = 3\theta$. In both cases we manage to provide an estimate of W using $O(\sqrt[3]{n}/\varepsilon^{4/3})$ samples.

Hybrid setting, unknown n. We now sketch our technique for sum estimation in the hybrid setting with unknown n. First, we sample items uniformly until we find the first repeated item and use the stopping time to infer the total number of items n. In expectation, we use $O(\sqrt{n})$ uniform samples and compute $\tilde{n}$, such that $n \leq \tilde{n}$ and $E[\tilde{n}] = n$. Now, we can use our algorithm for sum estimation in the proportional setting with advice $\tilde{n}$. This uses in expectation $O(\sqrt{\tilde{n}}/\varepsilon) = O(\sqrt{n}/\varepsilon)$ samples. As we prove, this complexity is optimal up to a constant factor for $\epsilon \geq 1/\sqrt{n}$. Again, if $\varepsilon \leq 1/\sqrt{n}$, then we use our coupon-collector-based algorithm that retrieves every element of U using $O(n \log n)$ samples.

Lower bounds.

The most technically challenging part of our paper is Line 3, where we prove lower bounds for all estimation problems we address in the first part of the paper. All our lower bound proofs follow a common thread. We now sketch the main ideas. First, we define two different instances of the estimation problem at hand (U_1, w_1) and (U_2, w_2) such that a $(1 \pm \varepsilon)$-approximation of W is sufficient to distinguish between them. Then, we define our *hard* instance as a mixture of the two: we take (U_1, w_1) with probability $1/2$ and (U_2, w_2) otherwise. We denote these events by $\mathcal{E}_1$ and $\mathcal{E}_2$ respectively. Second, we show that a Bayes classifier cannot distinguish between the two cases with probability $2/3$ using too few samples; since Bayes classifiers are risk-optimal[6] this implies that no classifier can have misclassification probability less than $1/3$ while using the same number of samples. To show that a Bayes classifier has a certain misclassification probability, we study the posterior distribution, conditioned on the samples it has seen. Let the multiset S represent the outcome of the samples. Since the prior is uniform, applying the Bayes theorem gives

$$\frac{P(\mathcal{E}_1 \mid S)}{P(\mathcal{E}_2 \mid S)} = \frac{P(S \mid \mathcal{E}_1)}{P(S \mid \mathcal{E}_2)}.$$

We denote this likelihood ratio by $\mathcal{R}(S)$. We show that whenever $|S|$ is (asymptotically) too small, we have $\mathcal{R}(S) \approx 1$ with probability close to 1. When $\mathcal{R}(S) \approx 1$, the posterior distribution is very close to uniform. This entails misclassification probability close to $1/2$. If X and Y are some random

[6] Risk of a classifier refers to the misclassification probability under some fixed distribution. For the Bayes classifier, we implicitly assume that this distribution is the same as the prior.

variables sufficient to reconstruct S (that is, there exists an algorithm that, given (X, Y), generates S' with the same distribution as S), we can define

$$\mathcal{R}(X) = \frac{P\,(X \,|\, \mathcal{E}_1)}{P\,(X \,|\, \mathcal{E}_2)} \quad \text{and} \quad \mathcal{R}(Y \,|\, X) = \frac{P\,(Y \,|\, X, \mathcal{E}_1)}{P\,(Y \,|\, X, \mathcal{E}_2)}$$

and we have, thanks to the Bayes theorem, $\mathcal{R}(S) = \mathcal{R}(X) \cdot \mathcal{R}(Y \,|\, X)$. In this way, we can break the problem of proving $\mathcal{R}(S) \approx 1$ into proving $\mathcal{R}(X) \approx 1$ and $\mathcal{R}(Y \,|\, X) \approx 1$. This allows us to reduce all our lower bounds to proving concentration of likelihood ratios for two basic problems: (1) distinguishing between two sets of size n and $(1 - \varepsilon)n$ by uniform sampling and (2) distinguishing between two sequences of i.i.d. random variables from $Bern(p)$ and $Bern(p - \varepsilon)$.

We now sketch how we bound the likelihood ratio $\mathcal{R}(S)$ for problem (1), the same technique applies to problem (2). In problem (1), we call $\mathcal{E}_1$ the event $|U| = n$ and $\mathcal{E}_2$ the event $|U| = (1 - \varepsilon)n$, and we set $P(\mathcal{E}_1) = P(\mathcal{E}_2) = 1/2$. First, we notice that $\mathcal{R}(S)$ depends only on the number $\ell(S)$ of distinct elements in S. Then, we prove three facts. First, $\ell(S) \,|\, \mathcal{E}_i$ is concentrated around $E[\ell(S) \,|\, \mathcal{E}_i]$. Second, $E[\ell(S) \,|\, \mathcal{E}_1] \approx E[\ell(S) \,|\, \mathcal{E}_2]$. Third, for small deviations of $\ell(S)$, we have small deviations of $\mathcal{R}(\ell(S))$. These three facts are sufficient to conclude that, with probability close to 1, $R(\ell(S))$ lies in a very narrow interval; further computations show that 1 lies in that interval, hence $\mathcal{R}(S) \approx 1$ with probability close to 1.

Preliminaries.

Problem definition. We now give a formal definition of the two settings that we consider. Let us have a set U of cardinality n and a weight function $w : U \to [0, \infty)$. We denote by W the sum $\sum_{a \in U} w(a)$. The following operations are allowed in the proportional sampling setting: (1) proportionally sample an item, this returns $(a, w(a))$ with probability $w(a)/W$; (2) given two items a, a', check whether $a = a'$. This is the only way we can interact with the items. In the hybrid setting, we may in addition (3) sample an item uniformly (that is, return $(a, w(a))$ for any $a \in U$ with probability $1/n$). In both settings, we want to compute an estimate $\hat{W}$ of W such that $(1 - \varepsilon)W \leq \hat{W} \leq (1 + \varepsilon)W$ with probability 2/3.

Notation. When $(1 - \varepsilon)W \leq \hat{W} \leq (1 + \varepsilon)W$ holds, we say that such $\hat{W}$ is a $(1 \pm \varepsilon)$-approximation of W. Some of our subroutines require "advice" in the form of a constant factor approximation of some value. For sake of consistency, we denote this constant factor approximation of $\bullet$ by $\tilde{\bullet}$. Similarly, if we want to estimate some value $\bullet$, we use $\hat{\bullet}$ to denote the estimate. Let us have some predicate ϕ that evaluates true on some subset of U and false on the rest. We denote by $P_{unif}(\phi(a))$ and $P_{prop}(\phi(a))$ the probability of ϕ evaluating to true for a being picked uniformly and proportionally, respectively.

We state all our results (both upper and lower bound) for some constant success probability $> 1/2$. These probabilities can be amplified to any other constants without increasing the asymptotic complexity. In pseudocode, we often say that we execute some algorithm with some failure probability. By this, we mean that one uses probability amplification to achieve that failure probability.

Relative bias estimation of a Bernoulli random variable. Let $X_1, X_2, \cdots$ be i.i.d. random variables distributed as $Bern(p)$. Lipton et al. [1993] gave a very simple algorithm that returns $\hat{p}$ such that, with probability at least 2/3, $\hat{p}$ is a $(1 \pm \varepsilon)$-approximation of p[7]. It can be summarized as follows.

Proposition 1 (follows from Lipton et al. [1993])**.** *Let $X_1, X_2, \cdots$ be i.i.d. random variables distributed as $Bern(p)$. There exists an algorithm that uses in expectation $O(\frac{1}{\varepsilon^2 p})$ samples and returns $\hat{p}$ such that $E[1/\hat{p}] = 1/p$ and*

$$P\left(|\hat{p} - p| > \varepsilon p\right) \leq \frac{1}{3}.$$

We call this algorithm BernoulliEstimator(ε). We assume that this algorithm has access to the sequence $X_1, X_2, \cdots$; we specify these random variables when invoking the algorithm.

Probability amplification and expected values. Consider an estimator that gives a guarantee on the estimate $\hat{x}$ that holds with some probability (say, guarantee that a proposition $\phi(\hat{x})$ holds with probability at least 2/3) and at the same time, we know that $E[\hat{x}] \leq y$ for some value y. We sometimes need to amplify the probability of the guarantee (that is, amplify the probability that $\phi(\hat{x})$ holds) but would like to retain a bound $E[\hat{x}] = O(y)$. We now argue that using the standard median trick is sufficient. Namely, we prove that

Lemma 2. *Let us have non-negative i.i.d. random variables $X_1 \cdots X_{2t-1}$ for some integer t, and let $X = median(X_1 \cdots X_{2t-1})$. It holds $E[X] \leq 2E[X_1]$.*

Proof. Let $X'_1, \cdots, X'_{2t-1}$ be the random variables $X_1, \cdots, X_{2t-1}$ sorted in increasing order. We then have

$$E[X] \leq E\left[\frac{1}{t} \cdot \sum_{i=t}^{2t-1} X'_i\right] \leq E\left[\frac{1}{t} \cdot \sum_{i=1}^{2t-1} X_i\right] \leq \frac{2t-1}{t} \cdot E[X_1].$$

□

[7] In that paper, the authors in fact solve a more general problem. For presentation of this special case, see Watanabe [2005].

```
Algorithm 1: PropEstimator($\tilde{n}, \varepsilon$)
Given a parameter $0 < \varepsilon < 1$ and advice $\tilde{n} \geq n$
$S \leftarrow$ perform $m = \sqrt{24\tilde{n}}/\varepsilon + 1$ samples
Return the estimate $\hat{W} = \binom{m}{2} \cdot \left( \sum_{s \in S} \frac{\binom{c_s}{2}}{w(s)} \right)^{-1}$
In case $c_s = 1$ for all $s \in S$ set $\hat{W} = \infty$.
```

Sum Estimation by Proportional Sampling.

In this section, we focus on sum estimation in the proportional setting. We design algorithms to estimate W and our objective is to minimize the total number of samples taken in the worst case. We present two different algorithms that provide an $(1 \pm \varepsilon)$-approximation of W with probability $2/3$. The first one, PropEstimator, assumes to have an upper bound on the number of elements $\tilde{n} \geq n$, and achieves sample complexity of $O(\sqrt{\tilde{n}}/\varepsilon)$. The second one, NoAdvicePropEstimator, does not assume any knowledge of n, and produce an ε-estimate using $O(\sqrt{n}/\varepsilon + \log n/\varepsilon^2)$ samples in expectation.

Algorithm with advice $\tilde{n} \geq |U|$.

Let $a_1 \dots a_m$ be m items picked independently at random from U with probabilities proportional to their weights. Let S be the set of sampled items, and for each $s \in S$ define c_s to be the number of times item s is sampled. For each $i, j \in [m]^2$ define Y_{ij} to be $1/w(a_i)$ if $a_i = a_j$ and 0 otherwise. We now estimate W as follows:
Before we prove correctness, we need the following lemma.

Lemma 3. *Given pairwise distinct $i, j, k \in [m]$, we have $E[Y_{i,j}] = 1/W$, $\mathrm{Var}(Y_{i,j}) \leq n/W^2$, and $Cov[Y_{i,j}, Y_{i,k}] = 0$*

Proof.

$$E[Y_{ij}] = \sum_{a \in U} \frac{1}{w(a)} P(x_i = x_j = a) = \sum_{a \in U} \frac{w(a)}{W^2} = \frac{1}{W}$$

$$\mathrm{Var}(Y_{ij}) \leq E[Y_{ij}^2] = \sum_{a \in U} \frac{1}{w(a)^2} P(x_i = x_j = a) = \sum_{a \in U} \frac{1}{W^2} = \frac{n}{W^2}$$

As for the covariance, it holds $Cov[Y_{i,j}, Y_{i,k}] = E[Y_{i,j} \cdot Y_{i,k}] - E[Y_{j,k}] \cdot E[Y_{i,k}]$. This is equal to 0 as

$$E[Y_{i,j} \cdot Y_{i,k}] = \sum_{a \in U} \frac{1}{w(a)^2} P(x_i = x_j = x_k = a) = \sum_{a \in U} \frac{w(a)}{W^3} = \frac{1}{W^2} = E[Y_{j,k}] \cdot E[Y_{i,k}]$$

□

Theorem 4. *Given parameters $\tilde{n}$ and $0 < \varepsilon < 1$,* PropEstimator$(\tilde{n}, \varepsilon)$ *has sample complexity $O(\frac{\sqrt{\tilde{n}}}{\varepsilon})$ and returns an estimate $\hat{W}$ such that $E[1/\hat{W}] = 1/W$. If, moreover, $\tilde{n} \geq n$, then $P(|\hat{W} - W| \leq \varepsilon W) \geq 2/3$.*

Proof. The sample complexity is clearly as claimed. We now prove that $1/\hat{W}$ is an unbiased estimator of $1/W$:

$$\frac{1}{\hat{W}} = \binom{m}{2}^{-1} \sum_{s \in S} \frac{\binom{c_s}{2}}{w(s)} = \binom{m}{2}^{-1} \sum_{1 \leq i < j \leq m} Y_{ij}$$

and thus

$$E\left[\frac{1}{\hat{W}}\right] = \binom{m}{2}^{-1} \sum_{1 \leq i < j \leq m} E\left[Y_{ij}\right] = \frac{1}{W}.$$

When i, j, k, ℓ are all distinct, Y_{ij} and $Y_{k\ell}$ are independent. Moreover, by Lemma 3, Y_{ij} and Y_{ik} are uncorrelated for $j \neq k$. Using the bound on $\mathrm{Var}(Y_{ij})$ from Lemma 3, we then have that

$$\begin{aligned} \mathrm{Var}\left[\frac{1}{\hat{W}}\right] &= \binom{m}{2}^{-2} \sum_{1 \leq i < j \leq m} \mathrm{Var}(Y_{ij}) \\ &\leq \binom{m}{2}^{-1} \frac{n}{W^2} \\ &\leq \frac{1}{12} \cdot \left(\frac{\varepsilon}{W}\right)^2 \end{aligned}$$

By Chebyshev inequality, it holds that

$$P\left(\left|\frac{1}{\hat{W}} - \frac{1}{W}\right| > \frac{\varepsilon}{2W}\right) \leq \frac{\mathrm{Var}\left[1/\hat{W}\right]}{(\varepsilon/2W)^2} \leq \frac{1}{3}$$

Finally, for $\varepsilon \leq 1$ we have

$$(1-\varepsilon)W \leq (1+\varepsilon/2)^{-1}W \leq \hat{W} \leq (1-\varepsilon/2)^{-1}W \leq (1+\varepsilon)W$$

This means that $|1/\hat{W} - 1/W| \leq \varepsilon/(2W)$ implies $|\hat{W} - W| \leq \varepsilon W$. Thus $P\left(|\hat{W} - W| \leq \varepsilon W\right) \geq 2/3$.

□

Algorithms for $|U|$ unknown.

In this section, we present the algorithm NoAdvicePropEstimator, which samples elements from U proportionally to their weights, and computes an $(1 \pm \varepsilon)$-approximation of W with probability 2/3, without any knowledge of $n = |U|$.

Algorithm 2: SetSizeEstimator()

```
$S_0 \leftarrow \emptyset$
for $i \in \mathbb{N}$ do
    Sample $a_i \in U$ uniformly
    if $a_i \in S_i$ then
        $\hat{s} \leftarrow |S_i|$
        $\hat{N} \leftarrow 4\hat{s}^2$
        return $\hat{N}$
    $S_{i+1} \leftarrow S_i \cup \{a_i\}$
```

NoAdvicePropEstimator takes $O(\sqrt{n}/\varepsilon + \log n/\epsilon^2)$ samples in expectation and works as follows. We partition U into buckets such that items in one bucket have roughly the same weight. We pick one bucket such that the items in this bucket are likely to have a sufficiently large total weight. We then estimate the sum restricted to this bucket. If we are able to do that, we can estimate the total weight by looking at what fraction of the proportional samples end up in this bucket. We estimate the sum restricted to the bucket as follows. Since the weights are roughly the same for all items in the bucket, we may use rejection sampling to efficiently simulate uniform samples from the bucket. That allows us to estimate the number of items in it, up to a constant factor. We use the algorithm PropEstimator with this estimated bucket size as the advice $\tilde{n}$.

Estimating $|U|$ through uniform sampling.

As a preliminary step, we assume that we are able to sample elements from U uniformly, rather than according to their weights. Under this assumption, we introduce the algorithm SetSizeEstimator that, using $O(\sqrt{n})$ expected samples, estimates $n = |U|$ up to a constant factor with probability $2/3$. The intuition behind SetSizeEstimator is fairly simple: if we sample uniformly with replacement from a universe of size n and we see the first repetition after t samples, then it is likely that $t \approx \sqrt{n}$.

Theorem 5. SetSizeEstimator *has expected sample complexity of* $O(\sqrt{n})$. *It returns an estimate* $\hat{N}$ *such that* $P(n \leq \hat{N}) \geq 2/3$ *and* $E[\hat{N}] = O(n)$.

Proof. We prove that when the algorithm aborts, it holds $P(\sqrt{n}/2 \leq \hat{s}) \geq 2/3$. The bound on $P(n \leq \hat{N})$ follows by the definition of $\hat{N}$. Define the event $\mathcal{E}_i = \{a_i \in S_i\}$, where we define that $a_i \notin S_i$ whenever the algorithm terminates before step i. It then holds $P(\mathcal{E}_i) \leq P(\mathcal{E}_i \mid \bigcap_{j<i} \bar{\mathcal{E}}_j) = i/n$. We

Algorithm 3: PropBucketSampler(b)

```
Sample $(a, w(a))$ proportionally
while $w(a) \notin [2^b, 2^{b+1})$ do
    Sample $(a, w(a))$ proportionally
return $(a, w(a))$
```

have

$$\begin{aligned} P\left(\hat{s} < \frac{\sqrt{n}}{2}\right) &= P\left(\bigcup_{i=0}^{\sqrt{n}/2-1} \mathcal{E}_i\right) \\ &\leq \sum_{i=0}^{\sqrt{n}/2-1} P(\mathcal{E}_i) \\ &\leq \sum_{i=0}^{\sqrt{n}/2-1} \frac{i}{n} \\ &= \frac{1}{n} \cdot \binom{\sqrt{n}/2}{2} < \frac{1}{6}. \end{aligned}$$

After $\sqrt{n}$ samples, each additional sample is a repetition with probability at least $1/\sqrt{n}$. The number of iterations before the algorithm returns is thus stochastically dominated by $\sqrt{n} + \mathit{Geom}(1/\sqrt{n})$. We may thus bound the expectation as

$$E\left[\hat{N}\right] = E\left[4\hat{s}^2\right] \leq O\left(n + E\left[\mathit{Geom}(1/\sqrt{n})^2\right]\right) \leq O(n)$$

where the last inequality is a standard result on the second moment of the geometric random variable. □

Simulating uniform sampling.

We define buckets $B_i = \{a \in U \mid w(a) \in [2^i, 2^{i+1})\}$ for each $i \in \mathbb{Z}$, and we show how to sample elements uniformly from B_i, while allowed to sample elements proportionally to their weight w. First, we show how to sample elements from a bucket B_b given a $b \in \mathbb{Z}$ proportionally in PropBucketSampler. We then use rejection sampling to obtain a uniform sample through UnifBucketSampler.

Lemma 6. *Let $p_b = P(w(a) \in [2^b, 2^{b+1}))$ for $a \in U$ sampled proportionally. Then, the expected sample complexity of both* PropBucketSampler *and* UnifBucketSampler *is $O(1/p_b)$.* PropBucketSampler *returns an item from the b-th bucket with distribution proportional to the weights.* UnifBucketSampler *returns an item from the b-th bucket distributed uniformly.*

Algorithm 4: UnifBucketSampler(b)

```
$(a, w(a)) \leftarrow$ PropBucketSampler($b$)
while Uniform($[0,1]$) $> \frac{2^b}{w(a)}$ do
    $(a, w(a)) \leftarrow$ PropBucketSampler($b$)
return $(a, w(a))$
```

Proof. PropBucketSampler performs samples until it samples an item a from bucket B_b; it returns a. This is equivalent to sampling proportionally conditioned on $a \in B_b$. This proves that the output has the claimed distribution. In each step, we finish with probability p_b, independent of other steps. The expected number of steps is, therefore, $1/p_b$. This proves the sample complexity.

Similarly, we terminate UnifBucketSampler after sampling $a \in B_b$ and Uniform($[0,1]$) $\leq \frac{2^b}{w(a)}$. Sampling until $a \in B_b$ is equivalent to sampling item a from B_b with probability $w(a)/A_b$ where A_b is the total weight of items in bucket B_b. Therefore, a is sampled in each step with probability

$$p_b \cdot \frac{w(a)}{A_b} \cdot \frac{2^b}{w(a)} = \frac{p_b 2^b}{A_b}$$

Since this probability is the same for all items $a \in B_b$, the resulting distribution is uniform. The rejection probability is upper-bounded by $1/2$. Therefore, UnifBucketSampler also has expected sample complexity of $O(1/p_b)$ □

Putting it together: Estimating W without advice.

Finally, we are ready to show the algorithm NoAdvicePropEstimator, that estimates W without relying on any advice $\tilde{n} \geq n$. To analyze it, we first need a lemma.

To analyze NoAdvicePropEstimator, we first need a lemma:

Lemma 7. *Consider b_1, b_2 and b as defined in* NoAdvicePropEstimator*, and let $a \in U$ be a random element sampled proportionally. Then,*

$$E\left[\frac{1}{P_{prop}\left(a \in B_b \mid b\right)}\right] = O(\log n).$$

Moreover, if we define n_b as the number of items in B_b we have

$$E\left[\frac{\sqrt{n_b}}{P_{prop}\left(a \in B_b \mid b\right)}\right] = O(\sqrt{n}).$$

.

Algorithm 5: NoAdvicePropEstimator(ε)

```
Sample $(a_1, w(a_1)), (a_2, w(a_2))$ proportionally
$b_i \leftarrow \lfloor \log w(a_i) \rfloor$ for $i = 1, 2$
$b \leftarrow \max(b_1, b_2)$
$\tilde{n}_b \leftarrow$ SetSizeEstimator() using UnifBucketSampler($b$) as
    sampling subroutine, with success probability 9/10
$\hat{W}_b \leftarrow$ PropEstimator($\frac{\varepsilon}{3}, \tilde{n}_b$) using PropBucketSampler($b$) as
    sampling subroutine, with success probability 9/10
$\hat{P}_b \leftarrow$ BernoulliEstimator $\left(\frac{\varepsilon}{3}\right)$ to estimate $P(a \in B_b)$ with success
    probability 9/10
$\hat{W} \leftarrow \hat{W}_b / \hat{P}_b$
return $\hat{W}$
```

Proof. Throughout this proof, we assume a to be sampled proportionally. We first prove the first statement. Define $k = \max\{j \,|\, B_j \neq \emptyset\}$, and set $\mathcal{B} = \{B_j | k - 2\log n < j \leq k \text{ and } B_j \neq \emptyset\}$. Notice that $|\mathcal{B}| \leq 2\log n$ and, for each $B_j \notin \mathcal{B}$, $x \in B_j$ we have

$$P\left(a = x\right) \leq \frac{2^{k-2\log n+1}}{W} = 2^{-2\log n+1} \cdot \frac{2^k}{W} \leq \frac{2}{n^2}$$

since there exists $y \in B_k$ and therefore $W \geq w(y) \geq 2^k$. Hence, sampling $a \in U$ proportionally we have

$$\begin{aligned} P\left(B_a \notin \mathcal{B}\right) &= P\left(\exists j, \text{ s.t. } a \in B_j \wedge B_j \notin \mathcal{B}\right) \\ &\leq \sum_{x \in \bigcup_{B_j \notin \mathcal{B}} B_j} P(a = x) \\ &\leq n \cdot \frac{2}{n^2} = \frac{2}{n}. \end{aligned}$$

Now, we notice that for each B_j, it holds

$$\begin{aligned} P\left(b \in B_j\right) &\leq 2 \cdot P\left(b_1 \in B_j\right) \cdot P\left(\exists i \leq j \,:\, b_2 \in B_i\right) \\ &= 2 \cdot P\left(a \in B_j\right) \cdot P\left(\exists i \leq j \,:\, a \in B_i\right) \\ &\leq 2 \cdot P\left(a \in B_j\right) \end{aligned} \tag{2.1}$$

where the factor two is given by the union bound, and we used $a \sim b_1 \sim b_2$.

Then, we can write

$$\begin{aligned}
E\left[\frac{1}{P\left(a\in B_b\,|\,b\right)}\right] &= \sum_{B_j\neq\emptyset}\frac{P\left(b\in B_j\right)}{P\left(a\in B_j\right)}\\
&= \sum_{B_j\in\mathcal{B}}\frac{P\left(b\in B_j\right)}{P\left(a\in B_j\right)}+\sum_{\substack{B_j\notin\mathcal{B}\\B_j\neq\emptyset}}\frac{P\left(b\in B_j\right)}{P\left(a\in B_j\right)}\\
&\leq 2\cdot|\mathcal{B}|+\sum_{\substack{B_j\notin\mathcal{B}\\B_j\neq\emptyset}}2P\left(\exists i\leq j\,:\,a\in B_i\right)\\
&\leq 2\cdot|\mathcal{B}|+\sum_{\substack{B_j\notin\mathcal{B}\\B_j\neq\emptyset}}2P\left(B_a\notin\mathcal{B}\right)\\
&\leq 4\log n+2n\cdot P\left(B_a\notin\mathcal{B}\right)\\
&\leq 4\log n+4=O(\log n).
\end{aligned}$$

The fist inequality is obtained using eq. (2.1). The second inequality descends from the fact that $B_j\notin\mathcal{B}$ and $i\leq j$ imply $B_i\notin\mathcal{B}$. The last two inequalities are obtained using n as an upper bound on the number of nonempty buckets B_j and recalling that $P(B_a\notin\mathcal{B})\leq 2/n$.

Now we can prove the second statement. Denote by n_b the number of elements in B_b and define $\ell=\arg\max_{i\in\mathbb{Z}} n_i 2^{i/2}$. If we define $S_i=\sum_{j\leq i}\sum_{a\in B_j} w(a)$, then we can rewrite the already proven inequality $P\left(b\in B_j\right)\leq 2\cdot P\left(a\in B_j\right)\cdot P\left(\exists i\leq j\,:\,a\in B_i\right)$ as

$$P\left(b\in B_j\right)\leq 2\cdot P\left(a\in B_j\right)\cdot\frac{S_j}{W}. \tag{2.2}$$

We now prove that there exists a constant $C>0$ such that, for all $i\in\mathbb{Z}$ it holds $S_{\ell-i}\leq C\cdot S_\ell\cdot 2^{-i/2}$. Notice that, by definition of ℓ, we have $n_j\cdot 2^{j/2}\leq n_\ell\cdot 2^{\ell/2}$ for all $j\in\mathbb{Z}$. We can now bound

$$\begin{aligned}
S_{\ell-i} &\leq \sum_{j\leq\ell-i} n_j\cdot 2^{j+1}\\
&\leq n_\ell\cdot\sum_{j\leq\ell-i}2^{\ell/2-j/2}\cdot 2^{j+1}\\
&= 2n_\ell 2^\ell\cdot\sum_{j\leq\ell-i}2^{j/2-\ell/2}\\
&\leq 2S_\ell\cdot\sum_{j\leq\ell-i}2^{j/2-\ell/2}\leq C\cdot S_\ell 2^{-i/2}.
\end{aligned}$$

Therefore, we have $\sum_{j<\ell}S_j=O(S_\ell)$. Notice that by the definition of ℓ we have $n_{\ell+i}\leq n_\ell\cdot 2^{-i/2}$ for each $i\geq 0$. Now we are ready to prove our final result.

$$
\begin{aligned}
E\left[\frac{\sqrt{n_b}}{P\left(a \in B_b \,|\, b\right)}\right] &= \sum_{j\in\mathbb{Z}} P\left(b \in B_j\right) \cdot \frac{\sqrt{n_j}}{P\left(a \in B_j\right)} \\
&\leq \sum_{j\in\mathbb{Z}} 2\frac{S_j}{W} \cdot \sqrt{n_j} \\
&\leq \sum_{j<\ell} 2\frac{S_j}{W} \cdot \sqrt{n} + \sum_{j\geq\ell} 2\sqrt{n_j} \\
&\leq \frac{O(S_\ell)}{W} \cdot \sqrt{n} + \sum_{i\geq 0} 2\sqrt{n_\ell} \cdot 2^{-i/4} = O(\sqrt{n}).
\end{aligned}
$$

The first inequality uses eq. (2.2), the second inequality is obtained splitting the series in two parts and using $S_j \leq W$. The last inequality is obtained plugging in $\sum_{j<\ell} S_j = O(S_\ell)$ and $n_{\ell+i} \leq n_\ell \cdot 2^{-i/2}$ for $i \geq 0$. □

Now we are ready to analyze NoAdvicePropEstimator.

Theorem 8. *Let $\hat{W}$ be the estimate returned by* NoAdvicePropEstimator*. Then $\hat{W}$ is an $(1 \pm \varepsilon)$-approximation of W with probability $2/3$. Moreover, its expected sample complexity is*

$$
O\left(\frac{\sqrt{n}}{\varepsilon} + \frac{\log(n)}{\varepsilon^2}\right).
$$

Proof. We start by proving correctness. Define $W_b = \sum_{x\in B_b} w(x)$ and $P_b = P(a \in B_b \,|\, b) = W_b/W$. Notice that W_b and P_b are random variables, since they depend on b. Now we prove that $\hat{W}$ is a $(1 \pm \varepsilon)$-approximation of W with probability $2/3$. Define the event $\mathcal{E}_1 = \{n \leq \tilde{n}_b\}$, we have $P(\mathcal{E}_1) \geq 9/10$ (where we use Theorem 5 together with probability amplification to amplify the success probability of $2/3$ to $9/10$). Define the event

$$
\mathcal{E}_2 = \left\{(1-\varepsilon/3)W_b \leq \hat{W}_b \leq (1+\varepsilon/3)W_b\right\}
$$

then, we have $P(\mathcal{E}_2 \,|\, \mathcal{E}_1) \geq 9/10$ (where we use Theorem 4 and probability amplification). Define the event

$$
\mathcal{E}_3 = \left\{(1-\varepsilon/3)P_b \leq \hat{P}_b \leq (1+\varepsilon/3)P_b\right\}
$$

then it holds $P(\mathcal{E}_3 \,|\, b) \geq 9/10$ (where we use Proposition 1 and probability amplification). On the event $\mathcal{E}_2 \cap \mathcal{E}_3$, it holds

$$
(1-\varepsilon) \cdot W \leq \frac{1-\varepsilon/3}{1+\varepsilon/3} \cdot \frac{W_b}{P_b} \leq \frac{\hat{W}_b}{\hat{P}_b} \leq \frac{1+\varepsilon/3}{1-\varepsilon/3} \cdot \frac{W_b}{P_b} \leq (1+\varepsilon) \cdot W.
$$

Then we can apply union bound and prove

$$\begin{aligned}P\left(\hat{W} < (1-\varepsilon)W \text{ or } \hat{W} > (1+\varepsilon)W\right) &\le P\left(\bar{\mathcal{E}}_2 \cup \bar{\mathcal{E}}_3\right)\\ &\le P\left(\bar{\mathcal{E}}_1\right) + P\left(\bar{\mathcal{E}}_2 \,|\, \mathcal{E}_1\right) + P(\bar{\mathcal{E}}_3)\\ &\le \frac{1}{10} + \frac{1}{10} + \frac{1}{10} \le \frac{1}{3}.\end{aligned}$$

It remains to prove that the expected number of samples that NoAdvicePropEstimator uses is as claimed. Denote by σ_1 the total number of samples taken on line 4, by σ_2 the total number of samples taken on line 5 and by σ_3 the total number of samples taken on line 6. We denote the number of samples employed during the i-th call to UnifBucketSampler(b) on line 4 with $\eta_1^{(i)}$; similarly, we denote the number of samples taken during the i-th call to PropBucketSampler(b) on line 5 with $\eta_2^{(i)}$. We can then write

$$\sigma_1 = \sum_{i=1}^{\tau_1} \eta_1^{(i)} \text{ and } \sigma_2 = \sum_{i=1}^{\tau_2} \eta_2^{(i)}$$

where τ_1 and τ_2 are the number of calls to UnifBucketSampler(b) performed on line 4 and line 5, respectively. First we notice that, thanks to Lemma 6, there exists a constant $K > 0$ such that $E[\eta_1^{(i)} \,|\, b], E[\eta_2^{(i)} \,|\, b] \le \frac{K}{P(a \in B_b \,|\, b)}$. Thanks to Theorem 5, we have $E[\tau_1 \,|\, b] = O(\sqrt{n_b})$ and $\tau_2 = O(\sqrt{\tilde{n}_b}/\varepsilon)$. Now we are ready to bound $E[\sigma_1]$ and $E[\sigma_2]$. We have

$$E\left[\sigma_1\right] = E\left[E\left[\sum_{i=1}^{\tau_1} \eta_1^{(i)} \,|\, b\right]\right] = E\left[E\left[\tau_1 \,|\, b\right] \cdot E\left[\eta_1^{(1)} \,|\, b\right]\right] \le E\left[\frac{K \cdot O(\sqrt{n_b})}{P(a \in B_b \,|\, b)}\right] = O\left(\sqrt{n}\right)$$

where the first equality is by the Wald's identity and the last equality is obtained applying Lemma 7. Similarly,

$$\begin{aligned}E\left[\sigma_2\right] &= E\left[E\left[\sum_{i=1}^{\tau_2} \eta_2^{(i)} \,|\, b\right]\right] = E\left[E\left[\tau_2 \,|\, b\right] \cdot E\left[\eta_2^{(1)} \,|\, b\right]\right]\\ &\le E\left[\frac{K \cdot \left(\sqrt{80\tilde{n}_b}/\varepsilon + 1\right)}{P(a \in B_b \,|\, b)}\right] = O\left(\frac{\sqrt{n}}{\varepsilon} + \log n\right) = O\left(\frac{\sqrt{n}}{\varepsilon}\right)\end{aligned}$$

where we used that $E[\sqrt{\tilde{n}_b}] \le \sqrt{E[\tilde{n}_b]} \le \sqrt{n}$, which holds thanks to Theorem 5 and Jensen inequality. In order to bound $E[\sigma_3 \,|\, b]$, recall that, thanks to Proposition 1, there exists a $C > 0$ such that, conditioning on the value of b, BernoulliEstimator$(\frac{\varepsilon}{3})$ takes in expectation at most $\frac{C}{P(a \in B_b \,|\, b)\varepsilon^2}$ samples in order to estimate $P(a \in B_b)$. Therefore we have

$$E[\sigma_3] = E\left[E[\sigma_3 \,|\, b]\right] \le E\left[\frac{C}{P\left(a \in B_b \,|\, b\right)\varepsilon^2}\right] = O\left(\frac{\log n}{\varepsilon^2}\right).$$

where the last equality holds by Lemma 7. This concludes the proof, since the total number of samples taken by NoAdvicePropEstimator is $\sigma_1+\sigma_2+\sigma_3$. By the bounds we have proven above, the expectation of $\sigma_1+\sigma_2+\sigma_3$ is as claimed. □

Sum Estimation by Hybrid Sampling.

In this section, we assume that we can sample elements both proportionally and uniformly. Again, we solve the task of providing an estimate $\hat{W}$ of W such that $\hat{W}$ is a $(1 \pm \varepsilon)$-approximation of W with probability $2/3$.

We notice that if we take $\Theta(n \log n)$ uniform samples then, with probability $2/3$, we see every element of U. This simple analysis is well-known under the name of Coupon Collector problem. If we know n (or any constant-factor approximation of it) we may simply take $\Theta(n \log n)$ samples, assume we have seen every element at least once, and compute W exactly. If we do not know n, we first use SetSizeEstimator to compute an estimate $\hat{N}$ of n accurate up to a constant factor. If SetSizeEstimator succeeds, then taking $O(\hat{N} \log \hat{N})$ samples suffices to collect all elements.

Therefore, it is sufficient to show an algorithm with complexity $T(n, \varepsilon)$ to obtain a complexity of the form $O(\min(T(n,\varepsilon), n \log n))$, as we can just run the coupon-collector algorithm in parallel and take the result provided by the first of the two algorithms to finish its execution. In what follows we only show how to achieve a complexity of $O(\sqrt[3]{n}/\varepsilon^{4/3})$ when $|U|$ is known, and of $O(\sqrt{n}/\varepsilon)$ when $|U|$ is unknown. As a consequence, the complexities that we achieve in this settings are $O(\min(\sqrt[3]{n}/\varepsilon^{4/3}, n \log n))$ and of $O(\min(\sqrt{n}/\varepsilon, n \log n))$ respectively.

Algorithms for $|U|$ known.

In this section, we show an algorithm that, given $n = |U|$, returns a $(1 \pm \varepsilon)$-approximation of W with probability $2/3$ using $O(\sqrt[3]{n}/\varepsilon^{4/3})$ samples. First, we introduce a subroutine that uses harmonic mean to estimate the average weight W/n; then we combine it with PropEstimator to obtain the main algorithm of this section.

Harmonic-mean-based estimator.

Here we show the algorithm HarmonicEstimator$(\varepsilon, \tilde{\theta}, \phi)$ that returns an $(1 \pm \varepsilon)$-approximation $\hat{\theta}$ of W/n with probability $2/3$. HarmonicEstimator$(\varepsilon, \tilde{\theta}, \phi)$ takes as advice an upper bound on the average weight $\tilde{\theta} \geq W/n$, and a parameters ϕ such that we expect $P_{unif}(w(a) \geq \phi)$ not to be too small and $\tilde{\theta}/\phi$ not to be too large. A more formal statement follows.

To see the intuition behind this algorithm, consider the case when $\phi \leq w(a)$ for all $a \in U$. It then holds $\hat{p} \approx 1$. We take k samples, and let $1/H$ be the

Algorithm 6: HarmonicEstimator($\varepsilon, \tilde{\theta}, \phi$)

```
$\hat{p} \leftarrow$ BernoulliEstimator $\left(P_{unif}(w(a) \geq \phi), \frac{\varepsilon}{3}\right)$ with success
  probability 9/10
$k \leftarrow 45 \cdot \frac{\tilde{\theta}}{\phi(1-\varepsilon/3)\hat{p}\varepsilon^2}$
Sample $a_1 \dots a_k$ proportionally
for $i = 1 \dots k$ do
    if $w(a_i) \geq \phi$ then
        $b_i = 1/w(a_i)$
    else
        $b_i = 0$
$H = \sum_{i=1}^{k} b_i/k$
$\hat{\theta} \leftarrow \hat{p}/H$
return $\hat{\theta}$
```

harmonic mean of the weights of the sampled items. We have $E[H] = n/W$, and $\hat{\theta} \approx 1/H$ as $\hat{p} \approx 1$. Unfortunately H might have a high variance due to elements having very small weights. To fix this, we consider a parameter ϕ such that $w(a) < \phi$ for some $a \in U$. Instead of $E[H] = n/W$, we then have $E[H] = n'/W$ for $n' = |\{a \in U \mid w(a) \geq \phi\}|$. We then multiply $1/H$ by $\hat{p}$ in order to adjust for the fraction of items that were ignored. Note that, while increasing ϕ, the variance of $1/H$ decreases; however, also n' decreases and this means that computing an estimate $\hat{p} \approx n'/n$ requires more samples. This introduces a trade-off between the algorithm's complexity and the variance of H.

Lemma 9. *Given parameters $\tilde{\theta}$ and $0 < \varepsilon < 1$,* HarmonicEstimator($\varepsilon, \tilde{\theta}, \phi$) *has expected sample complexity $O((1 + \frac{\tilde{\theta}}{\phi})/(p\,\varepsilon^2))$ where $p = P_{unif}(w(a) \geq \phi)$. It returns an estimate $\hat{W}$ such that $P(\hat{\theta} < W/(20n)) \leq 1/20$. If, moreover, $\tilde{\theta} \geq W/n$, then $P(|\hat{W} - W| \leq \varepsilon W) \geq 2/3$.*

Proof. We start by proving that $\hat{\theta}$ is a $(1+\varepsilon)$-approximation of W/n with probability 2/3 when $\tilde{\theta} \geq W/n$. Define the event $\mathcal{E} = \{\hat{p}$ is a $(1\pm\varepsilon/3)$-approximation of $p\}$. By Proposition 1 and using probability amplification, we have $P(\mathcal{E}) \geq 9/10$. For each $i = 1 \dots k$ we have

$$E[b_i] = \sum_{\substack{a \in U, \\ w(a) \geq \phi}} \frac{1}{w(a)} \cdot \frac{w(a)}{W} = \frac{n_{\geq \phi}}{W} = \frac{p \cdot n}{W}$$

where $n_{\geq \phi}$ is the number of elements in $a \in U$ with $w(a) \geq \phi$. Notice that this implies $E[H] = p \cdot n/W$. Moreover, for each $i = 1 \dots k$

$$\mathrm{Var}(b_i) \leq E\left[b_i^2\right] = \sum_{\substack{a \in U, \\ w(a) \geq \phi}} \frac{1}{w^2(a)} \cdot \frac{w(a)}{W} \leq \frac{n_{\geq \phi}}{\phi \cdot W} = \frac{p \cdot n}{\phi \cdot W}.$$

Conditioning on $\mathcal{E}$, we have that $(1-\varepsilon/3)\hat{p} \leq p$. The way we have set k allows us to bound

$$\mathrm{Var}(H \,|\, \mathcal{E}) = \frac{\mathrm{Var}(b_i)}{k} = \frac{p \cdot n}{\phi \cdot W} \cdot \frac{\varepsilon^2(1-\varepsilon/3)\hat{p}}{45} \cdot \frac{\phi}{\tilde{\theta}} \leq \left(\frac{p \cdot n}{W}\right)^2 \cdot \frac{\varepsilon^2}{45} = E[H]^2 \cdot \frac{\varepsilon^2}{45}$$

where we used that $\tilde{\theta} \geq W/n$. It holds $E[H \,|\, \mathcal{E}] = E[H]$. We may thus apply Chebyshev's inequality to get

$$P\left(|H - E[H]| > \frac{\varepsilon}{3} E[H] \,\middle|\, \mathcal{E}\right) \leq \frac{\mathrm{Var}(H \,|\, \mathcal{E})}{\left(\frac{\varepsilon}{3} E[H]\right)^2} \leq \frac{9}{45} = \frac{1}{5}$$

Since $\varepsilon < 1$, we have $(1-\varepsilon/3)^{-1} \leq 1+\varepsilon/2$ and $(1+\varepsilon/3)^{-1} \geq 1-\varepsilon/2$. Therefore

$$P\left(\left|\frac{1}{H} - \frac{1}{E[H]}\right| > \frac{\varepsilon}{2} \cdot \frac{1}{E[H]} \,\middle|\, \mathcal{E}\right) \leq \frac{1}{5}.$$

Again, since $\varepsilon < 1$, we have $(1+\varepsilon/3)\cdot(1+\varepsilon/2) \leq 1+\varepsilon$ and $(1-\varepsilon/3)\cdot(1-\varepsilon/2) \geq 1-\varepsilon$. Hence, using the union bound

$$P\left(\left|\frac{\hat{p}}{H} - \frac{p}{E[H]}\right| > \varepsilon \cdot \frac{p}{E[H]}\right) \leq P(\bar{\mathcal{E}}) + P\left(\left|\frac{1}{H} - \frac{1}{E[H]}\right| > \frac{\varepsilon}{2} \cdot \frac{1}{E[H]} \,\middle|\, \mathcal{E}\right) \leq \frac{1}{3}.$$

Since $p/E[H] = W/n$, we have that the estimate $\hat{\theta} = \hat{p}/H$ is a $(1+\varepsilon)$-approximation of W/n with probability $\geq 2/3$.

We now argue the sample complexity. The expected number of samples used on line 1 is by Proposition 1 equal to $O(1/(p\,\varepsilon^2))$. In the rest of the algorithm, we use k samples. It holds

$$E[k] = E\left[\frac{1}{\hat{p}}\right] \cdot O\left(\frac{\tilde{\theta}}{\phi\,\varepsilon^2}\right) = O\left(\frac{\tilde{\theta}}{p\,\phi\,\varepsilon^2}\right)$$

where the second equality holds by Proposition 1. The sample complexity is thus as claimed.

```
Algorithm 7: HybridEstimator($n, \varepsilon$)
   Abort this algorithm if it uses more than $C\,\frac{n^{1/3}}{\varepsilon^{4/3}}$ samples, where $C$ is
     a large enough constant.
1  Find $\theta$ such that $P\left(\frac{n^{2/3}}{\varepsilon^{2/3}} \leq \left|\{a \in U \mid w(a) \geq \theta\}\right| \leq 2 \cdot \frac{n^{2/3}}{\varepsilon^{2/3}}\right) \geq \frac{19}{20}$
2  $\hat{p} \leftarrow$ BernoulliEstimator $\left(P_{prop}(w(a) \geq \theta), \frac{\varepsilon}{3}\right)$ with success
     probability 19/20
3  if $\hat{p} \geq 1/2$ then
4      $\tilde{n}_{\geq\theta} \leftarrow 2 \cdot \frac{n^{2/3}}{\varepsilon^{2/3}}$
5      $\hat{W}_{\geq\theta} \leftarrow$ PropEstimator $\left(\tilde{n}_{\geq\theta}, \frac{\varepsilon}{3}\right)$ with success probability 19/20,
         run on proportional samples conditioned on $w(a) \geq \theta$, obtained
         by rejecting elements with $w(a) < \theta$
6      $\hat{W} \leftarrow \hat{W}_{\geq\theta}/\hat{p}$
7  else
8      $\hat{\rho} \leftarrow$ HarmonicEstimator($\varepsilon, 3\theta, \theta$) with success probability
         19/20
9      $\hat{W} \leftarrow n \cdot \hat{\rho}$
10 return $\hat{W}$
```

Finally, we prove that, regardless of $\tilde{\theta}$, it holds $P(\hat{\theta} < W/(20n)) \leq 1/20$. It holds $E[H/\hat{p}] = E[H]E[1/\hat{p}] = n/W$ where $E[1/\hat{p}] = 1/p$ by Proposition 1. Therefore, by the Markov's inequality

$$P\left(\frac{\hat{p}}{H} \leq \frac{W}{20n}\right) = P\left(\frac{H}{\hat{p}} \geq \frac{20n}{w}\right) \leq \frac{1}{20}.$$

□

Combining the two algorithms.

Here, we combine HarmonicEstimator with PropEstimator to obtain HybridEstimator. It works in the hybrid setting and provides a $(1 \pm \varepsilon)$-approximation of W with probability 2/3, using in expectation $O(\sqrt[3]{n}/\varepsilon^{4/3})$ samples. While analysing HybridEstimator, we can restrict ourselves to $\varepsilon \geq 8/\sqrt{n}$. Indeed, for very small values of ε (namely, $\varepsilon \leq 1/(\sqrt{n}\log n)$) we use the coupon-collector algorithm, and for intermediate values of ε (namely, $1/(\sqrt{n}\log n) < \varepsilon < 8/\sqrt{n}$) we use PropEstimator($n, \varepsilon$). The coupon collector algorithm gives a sample complexity of $O(n \log n)$, that is better than $O(\sqrt[3]{n}/\varepsilon^{4/3})$ for $\varepsilon < 1/(\sqrt{n}\log n)$. PropEstimator gives a sample complexity of $O(\sqrt{n}/\varepsilon)$, that is better than $O(\sqrt[3]{n}/\varepsilon^{4/3})$ for $\varepsilon < 8/\sqrt{n}$.

To implement line 1 it is sufficient to sample uniformly $120\,n^{1/3}\varepsilon^{2/3}$ elements of U and define θ as the element with the 180-th largest weight. (Note that for $\varepsilon \geq 8/\sqrt{n}$ we have $120\,n^{1/3}\varepsilon^{2/3} \geq 480$, so this is well-defined.) Let k

be such that there are k elements $a \in U$ with $w(a) \geq \theta$. A standard analysis using Chebyshev inequality shows that k is concentrated around $\frac{3}{2}\frac{n^{2/3}}{\varepsilon^{2/3}}$.

Theorem 10. *Given ε such that $8/\sqrt{n} \leq \varepsilon < 1$,* HybridEstimator$(n, \varepsilon)$ *uses $O(\sqrt[3]{n}/\varepsilon^{4/3})$ samples. With probability at least $2/3$, the returned estimate $\hat{W}$is a $(1 \pm \varepsilon)$-approximation of W.*

Proof. We first analyze a variant of the algorithm that does not abort. (That is, an algorithm with the same pseudocode except for the abortion condition removed.) We now show that this algorithm returns a $(1 \pm \varepsilon)$-approximation with probability at least $5/6$.

The scheme summarized above to find θ at line 1 succeeds with probability at least $19/20$. We call this event $\mathcal{E}_1$. Define the event $\mathcal{E}_2 = \{\hat{p}$ is a $(1 \pm \varepsilon/3)$-approximation of $p\}$. It holds $P(\mathcal{E}_2) \geq 19/20$.

We now consider the case $\hat{p} \geq 1/2$. On line 5 we employ the algorithm PropEstimator. Whenever it performs a sample, we simulate a proportional sample from the set $U_\theta = \{a \in U \,|\, w(a) \geq \theta\}$ by sampling until we sample item a such that $w(a) \geq \theta$. It is easy to see that the distribution obtained with this sampling scheme is exactly the proportional distribution on the set U_θ. Conditioning on $\mathcal{E}_1$, $\tilde{n}_{\geq\theta}$ is a valid advice and (by Theorem 4) PropEstimator returns a $(1 \pm \varepsilon/3)$-approximation of $W_\theta = \sum_{w(a)\geq\theta} w(a)$ with probability at least $2/3$. We amplify this probability to $19/20$. Hence, we have

$$P\left(\left\{\hat{W}_\theta \text{ is a } (1 \pm \varepsilon)\text{-approximation of } W_{\geq\theta}\right\} \cap \mathcal{E}_2\right) \geq \\ 1 - \left(\frac{1}{20} + P(\bar{\mathcal{E}_1}) + P(\bar{\mathcal{E}_2})\right) \geq \frac{5}{6}.$$

On this event, since $\varepsilon < 1$, we have

$$(1-\varepsilon)W \leq \frac{1-\varepsilon/3}{1+\varepsilon/3}W \leq \hat{W} \leq \frac{1+\varepsilon/3}{1-\varepsilon/3}W \leq (1+\varepsilon)W.$$

Now we analyse the case $\hat{p} < 1/2$. Whenever $3\theta \geq W/n$, HarmonicEstimator$(\varepsilon, 3\theta, \theta)$ returns a $(1 \pm \varepsilon)$-approximation of W/n with probability $2/3$ (by Lemma 9). We amplify that probability to $19/20$. Now we argue that, conditioning on $\mathcal{E}_2$, we have $3\theta \geq W/n$. Define $p = P_{prop}(w(a) \geq \theta)$, then (on $\mathcal{E}_2$) we have $p \leq (1 + \epsilon/3)\hat{p} \leq (1 + 1/3)1/2 \leq 2/3$. Hence, $n\theta \geq (1-p)W \geq W/3$ and thus $3\theta \geq W/n$, where the first inequality is obtained using $\sum_{w(a)<\theta} w(a) = (1-p)W$. Applying union bound gives that HarmonicEstimator$(\varepsilon, 3\theta, \theta)$ succeeds with probability at least $1 - (1/20 + P(\bar{\mathcal{E}_2}))) \geq 5/6$.

Therefore, regardless of the value of $\hat{p}$, we have shown that $\hat{W}$ is a $(1 \pm \varepsilon)$-approximation of W with probability at least $5/6$. Thus, the modified algorithm without abortion is correct with probability at least $5/6$. We now argue that the probability that Algorithm 7 is aborted is at most $1/6$ (for C large enough). By the union bound, its success probability is at least $2/3$.

```
Algorithm 8: NoAdviceHybridEstimator($\varepsilon$)
1 $\tilde{n} \leftarrow$ SetSizeEstimator(), with success probability 5/6 (using
    uniform sampling)
2 $\hat{W} \leftarrow$ PropEstimator($\tilde{n}, \varepsilon$), with success probability 5/6 (using
    proportional sampling)
3 return $\hat{W}$
```

In what follows, we compute how many samples are taken on each line. On line 1, we use only $120n^{1/3}\varepsilon^{2/3}$ samples. Thanks to Proposition 1, BernoulliEstimator on line 2 uses $O(1/(p\varepsilon^2))$ samples in expectation. In what follows, we condition on $\mathcal{E}_1 \cap \mathcal{E}_2$. It holds $p \geq P_{unif}(w(a) \geq \theta)$ and thus we have $p \geq \frac{1}{\varepsilon^{2/3}n^{1/3}}$. The number of samples used on line 2 is then in expectation $O(n^{1/3}/\varepsilon^{4/3})$. By the (conditional) Markov's inequality the probability that we use more than $C_1 n^{1/3}/\varepsilon^{4/3}$ is at most $1/30$, for some C_1 large enough. PropEstimator uses $O(\sqrt{n^{2/3}/\epsilon^{2/3}}/\epsilon) = O(n^{1/3}/\varepsilon^{4/3})$ samples in the worst case. The rejections cause only constant factor expected slowdown. Again, by the (conditional) Markov's inequality, for C_2 large enough, we use more than $C_2 n^{1/3}/\varepsilon^{4/3}$ with probability at most $1/30$. Since we have $p \geq \frac{1}{\varepsilon^{2/3}n^{1/3}}$, on line 8 HarmonicEstimator takes $O(n^{1/3}/\varepsilon^{4/3})$ samples in expectation (by Lemma 9). Thus, there exists a constant C_3 such that on line 8 we use more than $C_3 n^{1/3}/\varepsilon^{4/3}$ samples with probability at most $1/30$.

Set $C = 120 + C_1 + C_2 + C_3$. It then holds by the union bound that we use more than $C\sqrt[3]{n}/\varepsilon^{4/3}$ samples (and thus abort) with probability at most $P(\bar{\mathcal{E}}_1) + P(\bar{\mathcal{E}}_2) + 1/30 + 1/30 = 1/6$. □

Algorithms for $|U|$ unknown.

In this section we show an algorithm that, without any knowledge of $n = |U|$, provides a $(1 \pm \varepsilon)$-approximation of W with probability $2/3$ using $O(\sqrt{n}/\varepsilon)$ samples. This complexity is strictly higher than the one of Line 8 for $\varepsilon = \omega(1/\sqrt{n})$. However, it is near-optimal when we do not know n, as we will see in Line 3.

Theorem 11. NoAdviceHybridEstimator(ε) *uses in expectation* $O(\sqrt{n}/\varepsilon)$ *samples and, with probability at least* $2/3$, *we have* $(1-\varepsilon)W \leq \hat{W} \leq (1+\varepsilon)W$.

Proof. By Theorem 5, SetSizeEstimator takes $O(\sqrt{n})$ samples and returns $\tilde{n}$ such that $n \leq \tilde{n}$ with probability $2/3$. We amplify this probability to $5/6$. Conditioning on $n \leq \tilde{n}$, PropEstimator returns $(1 \pm \varepsilon)$-estimate of W with probability at least $2/3$ by Theorem 4. We amplify this probability to $5/6$. By the union bound, the algorithm returns a $(1 \pm \varepsilon)$-estimate with probability at least $2/3$.

Moreover, by Theorem 5 and Lemma 2, $E[\tilde{n}] = O(n)$. By the Jensen inequality, PropEstimator then uses in expectation $O(\sqrt{n}/\varepsilon)$ samples. □

Coupon collector algorithm for unknown $|U|$.

We can design a simple $O(n \log n)$ time algorithm that computes W exactly with probability $2/3$ in the hybrid setting, without knowing $n = |U|$. We denote this algorithm with NoAdviceCouponCollector. First, we use SetSizeEstimator to give an estimate $\hat{N}$ of n such that $n \leq \hat{N} \leq Cn$ for some constant C, with probability $\geq 5/6$. This guarantee holds thanks to Theorem 5 combined with probability amplification and Markov's inequality. We say that SetSizeEstimator succeeds if that is the case. Then, by standard coupon collector we know that, conditioned on SetSizeEstimator succeeding, $O(\hat{N} \log \hat{N}$ samples suffice to collect all elements in U, with probability $5/6$. Then, we simply compute the sum of their weights and by union bound this sum equals W with probability $\geq 2/3$.

We can use this to ensure we never use more than $O(n \log n)$ samples in the hybrid setting. Specifically, we may execute in parallel this algorithm together with one of the above algorithms and abort when one of them returns an estimate. If one wishes to implement this in practice, it is possible to instead do the following. Given the parameter ε, compute a threshold n_0 such that we would like to (if we knew n) execute NoAdviceCouponCollector if $n \leq n_0$ and NoAdviceHybridEstimator if $n > n_0$. We then run NoAdviceCouponCollector and we abort if in case we find n_0 distinct elements. If this happens, we then run NoAdviceHybridEstimator.

Lower Bounds.

In this section, we give lower bounds for the problems we study in this paper. The proofs of the lower bounds all follow a common thread. In what follows, we use the term *risk* to refer to the misclassification probability of a classifier.

The roadmap of all our proofs follows. First, we define two different instances of the problem (U_1, w_1) and (U_2, w_2) such that a $(1 \pm \varepsilon)$-approximation of W is sufficient to distinguish between them. Then, we define our *hard* instance as an equally likely mixture of the two, namely we take (U_1, w_1) with probability $1/2$ and (U_2, w_2) otherwise. We denote these events by $\mathcal{E}_1$ and $\mathcal{E}_2$ respectively. Second, we show that a Bayes classifier[8] cannot distinguish between the two cases with probability $2/3$ using too few samples; since Bayes classifiers are risk-optimal this implies that no classifier can have risk less than $2/3$ while using the same number of samples. To show that a Bayes classifier

[8] Suppose we have a partition of the probability space Ω into events $\mathcal{E}_1, \cdots, \mathcal{E}_k$. We want to guess which event happened, based on observation X. Bayes classifier outputs as its guess $\mathcal{E}_\ell$ that maximizes $P(\mathcal{E}_\ell | X)$.

has a certain risk, we study the posterior distribution. Let S represent the outcome of the samples. Since the prior is uniform, applying Byes theorem gives

$$\frac{P\left(\mathcal{E}_1 \,|\, S\right)}{P\left(\mathcal{E}_2 \,|\, S\right)} = \frac{P\left(S \,|\, \mathcal{E}_1\right)}{P\left(S \,|\, \mathcal{E}_2\right)}.$$

We call this ratio $\mathcal{R}(S)$ and show that $\mathcal{R}(S) \approx 1$ with probability close to 1. When $\mathcal{R}(X) \approx 1$, the posterior distribution is very close to uniform, and this entails a risk close to $1/2$. First, we show this formally with some technical lemmas, and then we instantiate our argument for each of the studied problems.

Lemma 12. *Given two disjoint events $\mathcal{E}_1, \mathcal{E}_2$ such that $P(\mathcal{E}_1) = P(\mathcal{E}_2) = 1/2$ and a random variable X, we define the ratio*

$$\mathcal{R}(x) = \frac{P\left(X = x \,|\, \mathcal{E}_1\right)}{P\left(X = x \,|\, \mathcal{E}_2\right)}.$$

Notice that $\mathcal{R}(X)$ is a random variable since it depends on the outcome of X. If

$$P\left(\frac{7}{8} \le \mathcal{R}(X) \le \frac{8}{7}\right) \ge \frac{14}{15}$$

then any classifier taking X and classifying $\mathcal{E}_1, \mathcal{E}_2$ has risk $\ge 2/5$.

Proof. First, we notice that since Bayes classifiers (BC) are risk-optimal, then it is sufficient to prove our statement for a Bayes classifier. Define $p_i = P(\mathcal{E}_i \,|\, X)$ for $i = 1, 2$, then Bayes classifier simply returns $\arg\max_{i=1,2} p_i$. By Bayes theorem and because $P(\mathcal{E}_1) = P(\mathcal{E}_2)$ we have $\mathcal{R}(X) = p_1/p_2$. If $7/8 \le \mathcal{R}(X) \le 8/7$, then

$$\frac{1}{p_2} = \frac{p_1 + p_2}{p_2} = 1 + \mathcal{R}(X) \in \left[\frac{15}{8}, \frac{15}{7}\right]$$

and the same holds for p_1, therefore $7/15 \le p_1, p_2 \le 8/15$. Hence, conditioning on $7/8 \le \mathcal{R}(X) \le 8/7$, the probability of correct classification is at most $8/15$. Finally, we have

$$P\left(\text{BC returns correct answer}\right) \le P\left(\mathcal{R}(X) < \frac{7}{8} \vee \mathcal{R}(X) > \frac{8}{7}\right) + P\left(\text{BC returns correct answer} \,\middle|\, \frac{7}{8} \le \mathcal{R}(X\right.$$

$$\le \frac{1}{15} + \frac{8}{15} = \frac{3}{5}.$$

□

Before proving the main theorems, we need several lemmas.

Lemma 13. *Consider an instance of our problem (U, w) so that $n = |U|$ and $w(a) = 1$ for each $a \in U$. We take m independent samples and denote by ℓ the number of distinct elements obtained. Then,* $\mathrm{Var}(\ell) \leq m^2/n$. *Moreover, for each $\delta > 0$*

$$P\left(|\ell - E[\ell]| \geq \frac{1}{\sqrt{\delta}} \cdot \frac{m}{\sqrt{n}}\right) \leq \delta.$$

Note that, since all weights are the same, the proportional sampling is equivalent to sampling uniformly from U.

Proof. We use the Efron-Stein inequality to prove a bound on the variance of ℓ. Let X_i be the i-th sample. Now ℓ is a function of $X_1, \cdots, X_m$ and we write it as $\ell = f(X_1, \cdots, X_m)$. Let $X'_1, \cdots, X'_m$ be an independent copy of $X_1, \cdots, X_m$. Let $\ell'_i = f(X_1, \cdots, X_{i-1}, X'_i, X_{i+1}, \cdots, X_m)$. It clearly holds that $|\ell - \ell'_i| \in \{0, 1\}$, moreover $\ell - \ell'_i = 1$ if and only if X_i does not collide with with any X_j for $j \neq i$ and X'_i does collide with some X_k for $k \neq i$. It holds that $|\{X_1, \cdots, X_{i-1}, X_{i+1}, \cdots X_m\}| \leq m$. Therefore, the probability that a X'_i lies in this set is $\leq m/n$. Since ℓ and ℓ'_i are symmetric, we have that also $\ell - \ell'_i = -1$ with probability $\leq m/n$. Hence, $E[(\ell - \ell'_i)^2] = P(|\ell - \ell'_i| = 1) \leq 2m/n$. Applying Efron-Stein we get

$$\mathrm{Var}(\ell) \leq \frac{1}{2} \cdot \sum_{i=1}^{m} E\left[(\ell - \ell'_i)^2\right] \leq \frac{m^2}{n}$$

Now we just plug this bound on the variance into Chebyshev inequality and we get the desired inequality. □

Fingerprints. Given a sample S, we define its *fingerprint* F as the set of tuples $(c_a, w(a))$ where for each distinct item a in S, we add to F such a tuple with c_a being equal to the number of copies of a in S. Having a fingerprint of S is sufficient for any algorithm, oblivious of (U, w), to produce a sample S' that is equal to S, up to relabeling of the elements. Since the only allowed queries are testing equality of two items and the weight query, one may easily prove that the execution of the algorithm on these two samples is the same (indeed, these two samples are indistinguishable by the equality and weight queries). Therefore, we can safely assume that an algorithm in the proportional setting takes as an input the fingerprint F of S, rather than S. For algorithms in the hybrid setting, we can assume that it takes as input separately the fingerprint of the proportional and the fingerprint of the uniform samples.

Lemma 14. *Let us have parameters n and $\epsilon < 1/3$. Let $N = n$ with probability $1/2$, and $N = (1 - \varepsilon)n$ otherwise. Consider the random instance of the sum estimation problem (U, w) with $|U| = N$ and $w(a) = 1$ for each $a \in U$. Consider a sample of size m and its fingerprint $F_m = \{(c_i, w(a_i))\}_{i=1\ldots\ell}$ and* *define the ratio*

$$\mathcal{R}(F_m) = \frac{P\left(F_m \mid N = n\right)}{P\left(F_m \mid N = (1-\varepsilon)n\right)}.$$

If $m = o(\sqrt{n}/\varepsilon)$ and $m = o(n)$, then

$$P\left(\frac{98}{100} \leq \mathcal{R}(F_m) \leq \frac{100}{98}\right) \geq \frac{99}{100}$$

for n large enough.

Proof. We can explicitly compute the likelihood of a given fingerprint $F_m = \{(c_i, w(a_i)\}_{i=1\ldots\ell}$ where ℓ is the number of distinct elements as

$$\begin{aligned} P\left(F_m \middle| N = r\right) =& \frac{\ell!}{r^m}\binom{r}{\ell}\binom{m}{c_1 \ldots c_\ell} \\ =& \frac{1}{r^{m-\ell}}\prod_{i=1}^{\ell-1}\left(1-\frac{i}{r}\right)\binom{m}{c_1 \ldots c_\ell} \end{aligned}$$

and therefore

$$\begin{aligned} \mathcal{R}(F_m) &= (1-\varepsilon)^{m-\ell} \cdot \prod_{i=1}^{\ell-1} \frac{1-i/n}{1-i/(1-\varepsilon)n} \\ &= (1-\varepsilon)^{m-\ell} \cdot \prod_{i=1}^{\ell-1}\left(1+\frac{\varepsilon i}{(1-\varepsilon)n - i}\right) \end{aligned}$$

Note that $\mathcal{R}(F_m)$ depends only on ℓ. From now on we denote it with $\mathcal{R}(\ell)$.

Now we define an interval $[a, b]$ such that $P(\ell \in [a, b]) \geq 99/100$. To do so, we first compute the expectation of ℓ and then use the concentration bound of Lemma 13. We prove that

$$E\left[\ell | N = (1-\varepsilon)n\right] \leq E\left[\ell | N = n\right] \leq E\left[\ell | N = (1-\varepsilon)n\right] + O\left(\varepsilon\frac{m^2}{n}\right).$$

The expression of $E\left[\ell | N = n\right]$ is given by

$$E\left[\ell | N = n\right] = n \cdot \left(1 - \left(1 - \frac{1}{n}\right)^m\right)$$

since each item is not sampled with probability $(1-\frac{1}{n})^m$. From this expression, it is apparent that $E\left[\ell | N = n\right]$ is increasing in n. Expanding this formula, we get

$$E\left[\ell | N = n\right] - E\left[\ell | N = (1-\varepsilon)n\right] = \varepsilon\frac{m^2}{2n} + O\left(\varepsilon\frac{m^3}{n^2}\right) = O\left(\varepsilon\frac{m^2}{n}\right)$$

where the last estimate uses the $m = o(n)$ assumption. Now we define $a = E\left[\ell | N = (1-\varepsilon)n\right] - 10m/\sqrt{n}$ and $b = E\left[\ell | N = n\right] + 10m/\sqrt{n}$. Using the last result we proved, we have

$$|b-a| = 20\frac{m}{\sqrt{n}} + O\left(\varepsilon\frac{m^2}{n}\right) = o\left(\frac{1}{\varepsilon}\right).$$

Note that, like all asymptotics in this proof, the $o(1/\epsilon)$ is for the limit $n \to +\infty$ and makes sense even for ϵ being a constant. Thanks to Lemma 13 we have $P(\ell \notin [a,b] \,|\, N = n) \le 1/100$ and $P(\ell \notin [a,b] \,|\, N = (1-\varepsilon)n) \le 1/100$, and therefore $P(\ell \notin [a,b]) \le 1/100$.

Now we give bounds on $\mathcal{R}(a)$ and $\mathcal{R}(b)$. It is apparent from the explicit formula above that $\ell \mapsto \mathcal{R}(\ell)$ is an increasing function. We have

$$\begin{aligned}\mathcal{R}(a)\cdot\frac{99}{100} &\le \mathcal{R}(a)\sum_{k\in[a,b]\cap\mathbb{Z}} P\left(\ell = k \,|\, N = (1-\varepsilon)n\right)\\ &\le \sum_{k\in[a,b]\cap\mathbb{Z}} \mathcal{R}(k)P\left(\ell = k \,|\, N = (1-\varepsilon)n\right)\\ &\le \sum_{k\in[a,b]\cap\mathbb{Z}} P\left(\ell = k \,|\, N = n\right) \le 1\end{aligned}$$

and thus, $\mathcal{R}(a) \le 100/99$. Analogously,

$$\begin{aligned}\frac{1}{\mathcal{R}(b)}\cdot\frac{99}{100} &\le \frac{1}{\mathcal{R}(b)}\sum_{k\in[a,b]\cap\mathbb{Z}} P\left(\ell = k \,|\, N = n\right)\\ &\le \sum_{k\in[a,b]\cap\mathbb{Z}} \frac{1}{\mathcal{R}(k)}P\left(\ell = k \,|\, N = n\right)\\ &\le \sum_{k\in[a,b]\cap\mathbb{Z}} P\left(\ell = k \,|\, N = (1-\varepsilon)n\right) \le 1\end{aligned}$$

and thus, $\mathcal{R}(b) \ge 99/100$. We now have an upper bound on $\mathcal{R}(a)$ and a lower bound on $\mathcal{R}(b)$. However, we need a lower bound on $\mathcal{R}(a)$ and an upper bound on $\mathcal{R}(b)$ (that is, the other way around). For each $k < m$ we have

$$\frac{\mathcal{R}(k+1)}{\mathcal{R}(k)} = \frac{1}{1-\varepsilon}\cdot\left(1 + \frac{\varepsilon k}{(1-\varepsilon)n - k}\right) \le (1+2\varepsilon)\cdot\left(1+\frac{2\varepsilon m}{n}\right) \le 1+3\varepsilon$$

for n large enough, where we used $k < m = o(n)$ and $\varepsilon \le 1/3$. Hence,

$$\begin{aligned}\mathcal{R}(b) &\le \mathcal{R}(a)\cdot(1+3\varepsilon)^{\lceil|b-a|\rceil}\\ &\le \frac{100}{99}\cdot e^{3\varepsilon\lceil|b-a|\rceil}\\ &\le \frac{100}{99}\cdot e^{o(1)} \le \frac{100}{98}\end{aligned}$$

where the last inequality holds for n large enough because $m = o(\sqrt{n}/\varepsilon)$.

$$\begin{aligned}\mathcal{R}(a) &\geq \mathcal{R}(b) \cdot (1+3\varepsilon)^{-\lceil |b-a| \rceil} \\ &\geq \frac{99}{100} \cdot e^{-3\varepsilon \lceil |b-a| \rceil} \\ &\geq \frac{99}{100} \cdot e^{-o(1)} \geq \frac{98}{100}\end{aligned}$$

Finally, we have for n large enough that

$$P\left(\mathcal{R}(\ell) \notin \left[\frac{98}{100}, \frac{100}{98}\right]\right) \leq P\left(\ell \notin [a,b]\right) \leq \frac{1}{100}.$$

□

Using the same approach as Lemma 14, we prove a similar result for the task of estimating the bias p of a Bernoulli random variable up to an additive ε. In this setting, we provide an asymptotic lower bound on the number of samples, where the asymptotics are meant for the limit $(p, \varepsilon) \to 0$. The following lemma is folklore and it is implied by Fact 2.2 in Canonne [2022]. Here we prove it for the sake of completeness, without claiming any original contribution.

Lemma 15. *Let $0 < \varepsilon < p$ and set $q = p$ with probability $1/2$, and $q = p - \varepsilon$ otherwise. Let $X_1 \dots X_m$ be a sequence of i.i.d. Bernoulli random variables with bias q, and let $\ell = \sum_{i=1}^{m} X_i \sim Bin(m,q)$. Define the ratio*

$$\mathcal{R}(\ell) = \frac{P\left(\ell \,|\, q = p\right)}{P\left(\ell \,|\, q = p - \varepsilon\right)}.$$

If $m = o(p/\varepsilon^2)$ then

$$P\left(\frac{98}{100} \leq \mathcal{R}(\ell) \leq \frac{100}{98}\right) \geq \frac{99}{100}.$$

for p (and thus also ϵ) small enough.

Proof. We follow the same scheme we adopted in the proof of Lemma 14. First, we compute $\mathcal{R}(\ell)$ explicitly

$$\mathcal{R}(\ell) = \frac{p^{\ell}(1-p)^{m-\ell}}{(p-\varepsilon)^{\ell}(1-p+\varepsilon)^{m-\ell}} = \frac{\left(1 + \frac{\varepsilon}{1-p}\right)^{\ell - m}}{\left(1 - \frac{\varepsilon}{p}\right)^{\ell}}.$$

We have $E[\ell \,|\, q = p] = mp$, $E[\ell \,|\, q = p - \varepsilon] = m(p-\varepsilon)$, $\mathrm{Var}(\ell \,|\, q = p) \leq mp$, and $\mathrm{Var}(\ell \,|\, q = p - \varepsilon) \leq mp$. We define $a = m(p-\varepsilon) - 10\sqrt{mp}$ and $b = mp + 10\sqrt{mp}$, and using Chebyshev inequality we have $P(\ell \in [a,b] \,|\, q = p) \geq 99/100$ and

$P(\ell \in [a,b] \,|\, q = p - \varepsilon) \geq 99/100$. Hence, $P(\ell \in [a,b]) \geq 99/100$. Notice that $|b-a| = m\varepsilon + 20\sqrt{mp} = o(p/\varepsilon)$ where the second equality holds by the assumption that $m = o(p/\epsilon^2)$. Now, we bound $\mathcal{R}(a)$ and $\mathcal{R}(b)$. Again, we notice that $\ell \mapsto \mathcal{R}(\ell)$ is an increasing function.

$$\begin{aligned}\mathcal{R}(a) \cdot \frac{99}{100} &\leq \mathcal{R}(a) \sum_{k \in [a,b] \cap \mathbb{Z}} P\left(\ell = k \,|\, q = p - \varepsilon\right) \\ &\leq \sum_{k \in [a,b] \cap \mathbb{Z}} \mathcal{R}(k) P\left(\ell = k \,|\, q = p - \varepsilon\right) \\ &\leq \sum_{k \in [a,b] \cap \mathbb{Z}} P\left(\ell = k \,|\, q = p\right) \leq 1\end{aligned}$$

and thus, $\mathcal{R}(a) \leq 100/99$. Analogously, we prove $\mathcal{R}(b) \geq 99/100$. For each $k < m$ we have

$$\frac{\mathcal{R}(k+1)}{\mathcal{R}(k)} = \frac{1 + \frac{\varepsilon}{1-p}}{1 - \frac{\varepsilon}{p}} \leq 1 + 3\frac{\varepsilon}{p}$$

for p and ε sufficiently small. Hence,

$$\begin{aligned}\mathcal{R}(b) &\leq \mathcal{R}(a) \cdot \left(1 + 3\frac{p}{\varepsilon}\right)^{\lceil |b-a| \rceil} \\ &\leq \frac{100}{99} \cdot e^{3\frac{p}{\varepsilon}\lceil |b-a| \rceil} \\ &\leq \frac{100}{99} \cdot e^{o(1)} \leq \frac{100}{98}\end{aligned}$$

where the last inequality holds for p and ϵ sufficiently small. Analogously, we prove $\mathcal{R}(a) \geq 98/100$. Finally, we have

$$P\left(\mathcal{R}(\ell) \notin \left[\frac{98}{100}, \frac{100}{98}\right]\right) \leq P\left(\ell \notin [a,b]\right) \leq \frac{1}{100}.$$

□

Proportional sampling.

In this section, we assume, as we did in Section 2, that we can sample only proportionally. We prove that $\Omega(\sqrt{n}/\varepsilon)$ samples are necessary to estimate W with probability 2/3, thus PropEstimator is optimal up to a constant factor.

Deciding the number of samples at run-time. In all our lower bounds, we show that it is not possible that an algorithm takes $m = o(T(n,\varepsilon))$ samples in the *worst case* and correctly approximates W with probability 2/3. All these lower bounds safely extend to lower bounds on the expected number of samples $E[m]$. All our proofs work by showing that the Bayes classifier has risk $1/2 - o(1)$. Suppose now that we have an algorithm A that uses in expectation $\mu(n, \epsilon) = o(T(n, \epsilon))$ samples. We now define a classifier as follows. We run A and abort if it uses more than $20\mu(n, \epsilon)$ samples[9]. We return the answer given by A or an arbitrary value if we have aborted the algorithm. By the Markov's inequality, the probability that we abort is at most $1/20$. Our classifier has risk $1/3 + 1/20 < 2/5$. Since any constant success probability greater than $1/2$ is equivalent up to probability amplification, we also have a classifier with risk $1/3$ that uses $O(\mu(n, \epsilon)) = o(T(n, \epsilon)$ samples. Since a Bayes classifier with such parameters does not exist (as we show) and Bayes classifiers are risk-optimal, this is a contradiction.

Theorem 16. *In the proportional setting, there does not exist an algorithm that, for every instance (U, w) with $|U| = n$, takes m samples for $m = o(\sqrt{n}/\varepsilon)$ and returns a $(1 \pm \varepsilon)$-approximation of W with probability $2/3$. This holds also when n is known to the algorithm.*

Proof. As already proven, we may assume that the algorithm only gets the fingerprint F_m of the sample S of size m, instead of S itself. In the rest of the proof, we separately consider two cases: $\varepsilon \geq 1/\sqrt{n}$ and $\varepsilon < 1/\sqrt{n}$.

Case $\varepsilon \geq \frac{1}{\sqrt{n}}$: We first define the hard instance (U, w). Define the random variable k as $k = (1 - \varepsilon)n$ with probability $1/2$ and $k = n$ otherwise, then let $U = \{a_1 \dots a_n\}$ and

$$w(a_i) = \begin{cases} 1 & \text{if } i \leq k \\ 0 & \text{otherwise.} \end{cases}$$

Items with weight zero are never sampled while sampling proportionally while we are sampling uniformly from those with weight 1. Moreover $m = o(\sqrt{n}/\varepsilon) = o(n)$ for $\varepsilon \geq 1/\sqrt{n}$. Hence, this is exactly the settings of Lemma 14. If we let F_m to be the fingerprint of the samples and define $\mathcal{R}(F_m)$ as in Lemma 14, we have

$$P\left(\frac{98}{100} \leq \mathcal{R}(F_m) \leq \frac{100}{98}\right) \geq \frac{99}{100}.$$

Applying Lemma 12 gives us the desired result.

Case $\varepsilon < \frac{1}{\sqrt{n}}$: For convenience, we show an instance of size $n + 1$ instead of n. First, we define $s^2 = \min\left\{\frac{\sqrt{n}}{\varepsilon m}, \frac{n}{4}\right\}$ and notice that $s = \omega(1)$ and $s \leq \sqrt{n}/2$. Define the random variable k as $k = n - s\sqrt{n}$ with probability $1/2$ and $k = n$

[9] Note that, while the algorithm does not know $\mu(n, \epsilon)$, this is not an issue in this argument. The reason is that a classifier is defined as an arbitrary function from the set of possible samples and private randomness to the set of classes. This allows us to "embed" μ into the classifier

otherwise. Define the events $\mathcal{E}_1 = \{k = n - s\sqrt{n}\}$ and $\mathcal{E}_2 = \{k = n\}$. We construct (U, w) so that $U = \{a_0 \dots a_n\}$ and

$$w(a_i) = \begin{cases} \frac{s\sqrt{n}}{\varepsilon} - n & \text{if } i = 0 \\ 1 & \text{if } 1 \leq i \leq k \\ 0 & \text{otherwise.} \end{cases}$$

Notice that our choice of s together with $\varepsilon < 1/\sqrt{n}$ guarantees $n - s\sqrt{n} \geq n/2$ and $s\sqrt{n}/\varepsilon - n = \omega(n)$. We have that $W = \sum_{i=0}^{n} w(a_i)$ differs by more than a factor of $(1+\varepsilon)$ between events $\mathcal{E}_1$ and $\mathcal{E}_2$.

Consider an element $a \in U$ sampled proportionally and define $p_i = P(a \neq a_0 \,|\, \mathcal{E}_i)$ for $i = 1, 2$. Then,

$$p_2 = \frac{n}{n - s\sqrt{n} + w(a_0)} = \frac{\varepsilon\sqrt{n}}{s}$$
$$p_1 = \frac{n - s\sqrt{n}}{n - s\sqrt{n} + w(a_0)} = \frac{p_2 - \varepsilon}{1 - \varepsilon}$$

and $|p_1 - p_2| \leq \varepsilon$. We perform m samples in total and define the random variable X counting the number of times items other than a_0 are sampled. We define F_X as the fingerprint of the sampled items different from a_0. Given F_X, we may easily reconstruct the fingerprint of the whole sample by adding the tuple $(m - X, w(a_0))$. It thus holds $P(F_m|\mathcal{E}_i) = P(F_X|\mathcal{E}_i)$ for $i \in \{1, 2\}$.

Now we define the event $\mathcal{L} = \{X \leq 30E[X]\}$ and by Markov's inequality $P(\mathcal{L}) \geq 29/30$. Moreover,

$$E[X] = m(p_1 + p_2)/2 \leq m(p_2 + \varepsilon) \leq m(\varepsilon\sqrt{n}/s + \varepsilon) \leq 2\varepsilon m\sqrt{n}/s = o\left(\frac{n}{s}\right).$$

Define $\varepsilon' = s/\sqrt{n}$. Conditioning on $\mathcal{L}$, we have $X \leq 30E[X] = o(\sqrt{n}/\varepsilon') = o(n)$. If we further condition on $X = x$ for some $x \leq 30E[X]$ and we look at F_X, we are exactly in the setting of Lemma 14. Therefore,

$$P\left(\frac{98}{100} \leq \frac{P\left(F_X \,|\, X = x, \mathcal{E}_1\right)}{P\left(F_X \,|\, X = x, \mathcal{E}_2\right)} \leq \frac{100}{98}\right) \geq \frac{99}{100}$$

and integrating over $\mathcal{L}$ we obtain

$$P\left(\frac{98}{100} \leq \frac{P\left(F_X \,|\, X, \mathcal{E}_1\right)}{P\left(F_X \,|\, X, \mathcal{E}_2\right)} \leq \frac{100}{98} \,\middle|\, \mathcal{L}\right) \geq \frac{99}{100}.$$

Now, we will bound the ratio

$$\mathcal{R}(X) = \frac{P\left(X \,|\, \mathcal{E}_1\right)}{P\left(X \,|\, \mathcal{E}_2\right)}.$$

We have $X = \sum_{i=1}^{m} X_i$, where X_i is an indicator for the i-th sample not being equal to a_0. Therefore, $X \,|\, \mathcal{E}_j \sim Bin(m, p_j)$ for $j = 1, 2$. It holds,

$|p_1 - p_2| \le \varepsilon$. Because $m = o(\sqrt{n}/\epsilon)$ and by the definition of s, we have $m = o(\sqrt{n}/(s\varepsilon)) = o(p_2/\varepsilon^2)$. We can apply Lemma 15 and obtain

$$P\left(\frac{98}{100} \le \mathcal{R}(X) \le \frac{100}{98}\right) \ge \frac{99}{100}.$$

Finally, we put the bounds together. We consider the ratio

$$\mathcal{R}(F_m) = \frac{P\left(F_m \,|\, \mathcal{E}_1\right)}{P\left(F_m \,|\, \mathcal{E}_2\right)} = \frac{P\left(F_X \,|\, \mathcal{E}_1\right)}{P\left(F_X \,|\, \mathcal{E}_2\right)} = \frac{P\left(X \,|\, \mathcal{E}_1\right) \cdot P\left(F_X \,|\, X, \mathcal{E}_1\right)}{P\left(X \,|\, \mathcal{E}_2\right) \cdot P\left(F_X \,|\, X, \mathcal{E}_2\right)}.$$

By the union bound, along with $7/8 < (98/100)^2$, we get

$$P\left(\mathcal{R}(F_m) \notin \left[\frac{7}{8}, \frac{8}{7}\right]\right) \le$$
$$P\left(\bar{\mathcal{L}}\right) + P\left(\frac{P\left(F_X \,|\, X, \mathcal{E}_1\right)}{P\left(F_X \,|\, X, \mathcal{E}_2\right)} \notin \left[\frac{98}{100}, \frac{100}{98}\right] \,\middle|\, \mathcal{L}\right) + P\left(\frac{P\left(X \,|\, \mathcal{E}_1\right)}{P\left(X \,|\, \mathcal{E}_2\right)} \notin \left[\frac{98}{100}, \frac{100}{98}\right]\right) \le$$
$$\frac{1}{30} + \frac{1}{100} + \frac{1}{100} \le \frac{1}{15}.$$

We can apply Lemma 12 and conclude the proof. □

Sum estimation in hybrid setting, known n.

In this section, we assume, as we did in Section 8, that we can sample both proportionally and uniformly. We will prove that $\Omega(\min(\sqrt[3]{n}/\varepsilon^{4/3}, n))$ samples are necessary to estimate W with probability $2/3$. This complexity is the minimum of the sample complexity of the HybridEstimator and (up to a logarithmic factor) the complexity of the standard coupon collector algorithm. Recall that for algorithms in the hybrid setting, we can assume that they take as input two separate fingerprints: the fingerprint of the proportional and the fingerprint of the uniform samples.

Theorem 17. *In the hybrid setting, there does not exist an algorithm that, for every instance (U, w) with $|U| = n$, takes $m = o(\min(\sqrt[3]{n}/\varepsilon^{4/3}, n))$ proportional and uniform samples and returns a $(1 \pm \varepsilon)$-approximation of W with probability $2/3$. This holds also when n is known to the algorithm.*

The proof of Theorem 17 closely mimics the proof of Lemma 14. Indeed, in order to "fool" the uniform samples, we pad our distribution with many 0-weight elements. Indeed, only $n^{2/3}/\varepsilon^{2/3}$ element will have non-zero weight. On those elements, we basically apply Lemma 14. Then, to bound the information obtained by sampling (uniformly) zero-weight elements we use Lemma 15.

Proof. It is enough to prove that for $\varepsilon \ge 8/\sqrt{n}$, any algorithm returning a $(1 \pm \varepsilon)$-approximation of W with probability $2/3$ must take $\Omega(\sqrt[3]{n}/\varepsilon^{4/3})$

samples. Indeed, if $\varepsilon < 8/\sqrt{n}$, a $(1 \pm \varepsilon)$-approximation is also a $(1 \pm 8/\sqrt{n})$-approximation, and then we have reduced to the case $\varepsilon \geq 8/\sqrt{n}$ where we know that $\Omega(\sqrt[3]{n}/n^{-\frac{1}{2}\cdot\frac{4}{3}}) = \Omega(n)$ samples are necessary. In either case we then need $\Omega(\min(\sqrt[3]{n}/\varepsilon^{4/3}, n))$ samples.

Define the random variable k as $k = (1-\varepsilon)n^{2/3}/\varepsilon^{2/3}$ with probability 1/2 and $k = n^{2/3}/\varepsilon^{2/3}$ otherwise. Define the events $\mathcal{E}_1 = \{k = (1-\varepsilon)n^{2/3}/\varepsilon^{2/3}\}$ and $\mathcal{E}_2 = \{k = n^{2/3}/\varepsilon^{2/3}\}$. We construct (U, w) so that $U = \{a_1 \dots a_n\}$ and

$$w(a_i) = \begin{cases} 1 & \text{if } 1 \leq i \leq k \\ 0 & \text{otherwise.} \end{cases}$$

We have that $W = \sum_{i=1}^{n} w(a_i)$ differs by more than a factor of $(1+\varepsilon)$ between events $\mathcal{E}_1$ and $\mathcal{E}_2$. Notice that $k \leq n/4$ as $\varepsilon \geq 8/\sqrt{n}$. Let S_p be the multiset of proportional samples and S_u the multiset of uniform samples. Let $T_h = (S_p \cup S_u) \cap \{k+1, \cdots, n\}$ and $T_l = (S_p \cup S_u) \cap \{1, \cdots, k\}$. Let $F_l = \{(c_i, w(a_i))\}_i$ be the fingerprint of T_l and $F_h = \{(c_i, w(a_i))\}_i$ of be the fingerprint of T_h. We now argue hat (F_l, F_h) is sufficient for any algorithm, oblivious of the choice of k, to reconstruct sample multisets S'_u and S'_p distributed as S_u and S_p. We pick $|F_l| - m$ items at random from F_l and let S'_u be the multiset of these items, together with all items in F_h. We let S'_p be the multiset of the items left in F_l. It is easy to verify that $(S'_u, S'_p) \sim (S_u, S_p)$. Thus, we can assume that the algorithm is given (F_l, F_h) as input, instead of the sample multisets S_u and S_p.

Consider $a \in U$ sampled uniformly, and define $p_i = P(w(a) = 1 \,|\, \mathcal{E}_i)$ for $i = 1, 2$. Then,

$$p_1 = \frac{n^{2/3}/\varepsilon^{2/3}}{n} = \frac{1}{n^{1/3}\varepsilon^{2/3}}$$
$$p_2 = \frac{(1-\varepsilon)n^{2/3}/\varepsilon^{2/3}}{n} = \frac{1}{n^{1/3}\varepsilon^{2/3}} - \frac{\varepsilon^{1/3}}{n^{1/3}}$$

and defining $\varepsilon' = \varepsilon^{1/3}/n^{1/3}$ we obtain $p_2 = p_1 - \varepsilon'$. Let $X = |S_u \cap \{1, \cdots, k\}|$. Since $X \,|\, \mathcal{E}_j \sim Bin(m, p_j)$ for $j = 1, 2$ and $m = o(n^{1/3}/\varepsilon^{4/3}) = o(p_1/\varepsilon'^2)$, we can apply Lemma 15 and obtain

$$P\left(\frac{98}{100} \leq \frac{P(X \,|\, \mathcal{E}_1)}{P(X \,|\, \mathcal{E}_2)} \leq \frac{100}{98}\right) \geq \frac{99}{100}.$$

Conditioning on $X = x$ for some $x = 1 \dots m$, F_l represents a sample of $x + |S_p|$ items drawn uniformly from a set of cardinality k, so we are in the setting of lemma 14. Moreover, we have

$$|F_l| \leq |S_p| + |S_u| = o\left(\frac{\sqrt[3]{n}}{\varepsilon^{4/3}}\right) = o\left(\frac{\sqrt{n^{2/3}/\varepsilon^{2/3}}}{\varepsilon}\right).$$

Hence, Lemma 14 holds and we have

$$P\left(\frac{98}{100} \le \frac{P\left(F_l \,|\, X = x, \mathcal{E}_1\right)}{P\left(F_l \,|\, X = x, \mathcal{E}_2\right)} \le \frac{100}{98}\right) \ge \frac{99}{100}$$

and integrating over $x = 1 \ldots m$ we have

$$P\left(\frac{98}{100} \le \frac{P\left(F_l \,|\, X, \mathcal{E}_1\right)}{P\left(F_l \,|\, X, \mathcal{E}_2\right)} \le \frac{100}{98}\right) \ge \frac{99}{100}.$$

Similarly, we have that $|F_h| \le |S_u| = o(\sqrt[3]{n}/\varepsilon^{4/3}) = o(\sqrt{n}/\varepsilon)$ where the second inequality holds because we are assuming $\varepsilon > 8/\sqrt{n}$. Moreover, conditioning on $X = x$ for some $x = 1 \ldots m$, F_h represent a sample of $|S_u| - x$ items drawn uniformly from a set of size $n - k$. It holds $n - k \ge 3n/4$, and $n - k$ thus differs by at most a factor $1 - \varepsilon$ between the two events $\mathcal{E}_1, \mathcal{E}_2$. Again, we are in the right setting to apply Lemma 14, and integrating over $x = 1 \ldots m$ gives

$$P\left(\frac{98}{100} \le \frac{P\left(F_h \,|\, X, \mathcal{E}_1\right)}{P\left(F_h \,|\, X, \mathcal{E}_2\right)} \le \frac{100}{98}\right) \ge \frac{99}{100}.$$

We are now ready to put everything together. Note that F_l and F_h are independent once conditioned on $(\mathcal{E}_1, X)$ or $(\mathcal{E}_2, X)$. Define the ratio

$$\mathcal{R}(F_l, F_h) = \frac{P\left((F_l, F_h) \,|\, \mathcal{E}_1\right)}{P\left((F_l, F_h) \,|\, \mathcal{E}_2\right)} = \frac{P\left(X \,|\, \mathcal{E}_1\right) \cdot P\left(F_l \,|\, X, \mathcal{E}_1\right) \cdot P\left(F_h \,|\, X, \mathcal{E}_1\right)}{P\left(X \,|\, \mathcal{E}_2\right) \cdot P\left(F_l \,|\, X, \mathcal{E}_2\right) \cdot P\left(F_h \,|\, X, \mathcal{E}_2\right)}$$

Using the union bound, along with $7/8 < (98/100)^3$, we get

$$\begin{aligned} P\left(\mathcal{R}(F_l, F_h) \notin \left[\frac{7}{8}, \frac{8}{7}\right]\right) &\le \\ P\left(\frac{P\left(X \,|\, \mathcal{E}_1\right)}{P\left(X \,|\, \mathcal{E}_2\right)} \notin \left[\frac{98}{100}, \frac{100}{98}\right]\right) + P\left(\frac{P\left(F_l \,|\, X, \mathcal{E}_1\right)}{P\left(F_l \,|\, X, \mathcal{E}_2\right)} \notin \left[\frac{98}{100}, \frac{100}{98}\right]\right) &+ \\ P\left(\frac{P\left(F_h \,|\, X, \mathcal{E}_1\right)}{P\left(F_h \,|\, X, \mathcal{E}_2\right)} \notin \left[\frac{98}{100}, \frac{100}{98}\right]\right) &\le \\ \frac{1}{100} + \frac{1}{100} + \frac{1}{100} &< \frac{1}{15}. \end{aligned}$$

We can apply Lemma 12 and conclude the proof.

□

Sum estimation in hybrid setting, unknown n.

We now prove a lower bound for the hybrid setting, in case the algorithm does not know n.

Theorem 18. *In the hybrid setting, there does not exist an algorithm that, for every instance (U, w), takes $m = o(\min(\sqrt{n}/\varepsilon, n))$ samples and returns a $(1 \pm \varepsilon)$-approximation of W with probability $2/3$. This holds also when the algorithm is provided with an advice $\tilde{n}$ such that $(1 - \varepsilon)n \le \tilde{n} \le n$.*

Proof. Employing the same argument as in Theorem 17, it is sufficient to prove that for $\varepsilon \geq 1/\sqrt{n}$ a lower bound of $\Omega(\sqrt{n}/\varepsilon)$ holds.

Consider the instance (U, w) where $w(a) = 1$ for each $a \in U$ and we set $|U| = n$ with probability 1/2 and $|U| = (1-\varepsilon)n$ otherwise. Providing a $(1\pm\varepsilon)$-approximation of W is equivalent to distinguishing between the two cases. On this instance, sampling uniformly and proportionally is the same. Therefore, we are in the setting of Lemma 14. Combining Lemma 14 and Lemma 12 like in the proofs above, we get that no classifier can distinguish between $|U| = n$ and $|U| = (1-\varepsilon)n$ with probability 2/3 using $o(\sqrt{n}/\varepsilon)$ samples. □

Counting Edges in a Graph.

In this section, we show an algorithm that estimates the average degree of a graph $G = (V, E)$ in the model in which we are allowed to perform random vertex queries, random edge queries, and degree queries. Recall that a *random vertex query* returns a uniform sample form V, a *random edge queries* returns a uniform sample from E, and a *degree queries* returns $\deg(v)$ when we provide $v \in V$ as argument. In this section, we denote the number of vertices and edges with n and m respectively.

Here, we show that HarmonicEstimator from Line 8, can be adapted to estimate the average degree d. It achieves a complexity of $O(\frac{m \log\log n}{n'\varepsilon^2} + \frac{n}{n'\varepsilon^2})$ in expectation, where n' is the number of non-isolated[10] vertices. This is very efficient when there are few isolated vertices and the graph is sparse. Moreover, the only way we use sampling of vertices is to estimate the number of non-isolated vertices. Therefore, if we assume that there are no isolated vertices in the graph, it is sufficient to only be able to uniformly sample edges.

Our approach is similar to that of Katzir et al. [2011] but the authors in the paper do not prove bounds on the time complexity. Moreover, their algorithm only works when there are no isolated vertices.

Theorem 19. *Given a graph $G = (V, E)$, consider a model that allows (1) random vertex queries, (2) random edge queries, and (3) degree queries. In this model, there exists an algorithm that, with probability at least 2/3, returns a $(1 \pm \varepsilon)$-approximation $\hat{d}$ of the average degree $d = 2m/n$. This algorithm performs $O(\frac{m \log\log d}{n'\varepsilon^2} + \frac{n}{n'\varepsilon^2})$ queries in expectation, where n' is the number of non-isolated vertices.*

Proof. We first show an algorithm that is given $\tilde{\theta}$ such that $d \leq \tilde{\theta}$, has time complexity $O(\frac{\tilde{\theta} n}{\varepsilon^2 n'} + \frac{n}{\varepsilon^2 n'})$, and is correct with probability 2/3. We define a sum estimation problem (U, w). We set the universe to be $U = V$ and for each vertex $v \in U$, we set its weight $w(v) = \deg(v)$. Then $W = \sum_{a \in U} w(a) = 2m$ and $W/n = d$. Sampling an edge uniformly at random and picking one

[10] Recall that a vertex is isolated if it has degree 0.

of its endpoints at random corresponds to sampling a vertex proportionally to its weight. Moreover, we can sample vertices uniformly. Therefore, we are able to implement both queries of the hybrid setting. We run HarmonicEstimator$(\varepsilon, \tilde{\theta}, 1)$. By Lemma 9, it returns with probability at least $2/3$ a $1 \pm \varepsilon$-approximation of d, and its sample complexity is $O(\frac{\tilde{\theta} n}{\varepsilon^2 n'} + \frac{n}{\varepsilon^2 n'})$ since what is called p in Lemma 9 is now n'/n.

It remains to get rid of the need for advice $\tilde{\theta}$. We use the standard technique of performing a geometric search. See, for example, Goldreich and Ron [2006] for more details. We initialize $\tilde{\theta} = 1$ and in each subsequent iteration, we double $\tilde{\theta}$. Let K be a large enough constant. In each iteration, we run $K \log\log \tilde{\theta}$ independent copies of HarmonicEstimator$(\varepsilon, \tilde{\theta}, 1)$ and denote with $d_1 \dots d_{K \log\log \tilde{\theta}}$ the returned estimates. We define $\hat{d}$ as the median of $d_1 \dots d_{K \log\log \tilde{\theta}}$. We say that the d_i succeeds if both the following hold: (i) $d_i \geq d/20$, (ii) $\tilde{\theta} < d$ or d_i is a $(1 \pm \varepsilon)$-approximation of d. Otherwise we say that d_i fails. We extend this definition to $\hat{d}$. Thanks to Lemma 9, d_i fails with probability $\leq 1/3 + 1/20$. By a standard argument based on the Chernoff bound, for K large enough, we have that $\hat{d}$ fails with probability at most $2/(\pi \log^2(2\tilde{\theta}))$. Denote with $\mathcal{E}_j$ the event that $\hat{d}$ succeeds at iteration j (i.e., when $\tilde{\theta} = 2^{j-1}$). Define $\mathcal{E} = \bigcap_{j \geq 0} \mathcal{E}_j$. Union bound gives $P(\mathcal{E}) \geq 1 - \frac{2}{\pi} \sum_{j>0} \frac{1}{j^2} = 2/3$. We stop our algorithm when $\hat{d} \leq \tilde{\theta}/20$ and return $\hat{d}$. Conditioning on $\{\hat{d} \leq \tilde{\theta}/20\} \cap \mathcal{E}$, we have $\tilde{\theta}/20 \geq \hat{d} \geq d/20$. This implies that $\tilde{\theta} \geq d$ and hence $\hat{d}$ is a $(1 \pm \varepsilon)$-approximation of d. Since $P(\mathcal{E}) \geq 2/3$, we have that with probability $2/3$ the returned value is a $(1 \pm \varepsilon)$-approximation of d.

One iteration of our algorithm has time complexity $O(\frac{\tilde{\theta} n \log\log \tilde{\theta}}{\varepsilon^2 n'})$. We argue that the expected complexity is dominated by the first iteration in which $\tilde{\theta} \geq 40\, d$. The time complexity of each additional iteration (conditioned on being executed) increases by a factor $2 + o(1)$. Each additional iteration is executed only if the previous one resulted in an estimate $\hat{d} > \tilde{\theta}/20 \geq 2\, d$. This happens only when $\hat{d}$ is not a $(1 \pm \varepsilon)$-approximation of d, and assuming a correct advice $\tilde{\theta} \geq d$ this happens outside of $\mathcal{E}$. Therefore, we execute each additional iteration with probability $\leq 1/3$. Since the time spend in each iteration increases by a factor $2 + o(1)$ while the probability of executing the iteration decreases by a factor of 3, the expected complexity contributed by each additional iteration for $\tilde{\theta} \geq 40\, d$ decreases by a factor of $2/3 + o(1)$. Therefore, the expected complexity is dominated (up to a constant factor) by the first execution with $\tilde{\theta} \geq 40\, d$. If $d \geq 1$, then in this iteration, we have $\tilde{\theta} = \Theta(d)$. The time complexity is then $O(\frac{m \log\log d}{\epsilon^2 n'})$. If $d < 1$, then it holds $\theta = O(1)$ in this execution. The complexity is then $O(\frac{n}{n' \epsilon^2})$. This gives total time complexity of $O(\frac{m \log\log d}{n' \epsilon^2} + \frac{n}{n' \epsilon^2})$. □

Open Problems.

We believe there are many interesting open problems related to our work. We now give a non-comprehensive list of questions that we think would give more understanding of weighted sampling and its applications.

More efficient algorithm for spacial classes of inputs. Are there some large classes of inputs for which it is possible to get a more efficient algorithm? Can the problem be parameterized by some additional parameters apart from n, ε (e.g. empirical variance) that tend to be small in practice?

Different sampling probabilities. Are there settings where one may efficiently sample with probability depending on the value of an item but not exactly proportional? Could this be used to give a general algorithm for estimating the sum W? An example of such a result is Dasgupta et al. [2014a] where the authors show an efficient algorithm for estimating the average degree of a graph when sampling vertices with probabilities proportional to $\frac{m}{n} + d(v)$.

Get a complete understanding of edge counting. The complexity of the problem of approximately counting edges in a graph is understood in terms of n, m in the setting where we can only sample vertices uniformly at random. What is the complexity of counting edges when we allow only random edge queries? What if both random edge and random vertex queries are allowed? As we show, it may be useful to parameterize the problem by the fraction of vertices that are not isolated. What is the complexity of the problem under such parameterization?

Acknowledgements

Lorenzo Beretta and Jakub Tětek belong to Basic Algorithms Research Copenhagen (BARC), University of Copenhagen. BARC is supported by the VILLUM Foundation grant 16582. Jakub Tětek received funding from the Bakala Foundation. Lorenzo Beretta receives funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 801199.

We are grateful to Rasmus Pagh for his advice. We would like to thank our supervisor, Mikkel Thorup, for helpful discussions and his support. We would like to thank Talya Eden for her advice regarding related work.

Chapter 3

Edge Sampling and Graph Parameter Estimation via Vertex Neighborhood Accesses

Jakub Tětek Mikkel Thorup

{j.tetek,mikkel2thorup}@gmail.com

Basic Algorithms Research Copenhagen
University of Copenhagen

Abstract

In this paper, we consider the problems from the area of sublinear-time algorithms of *edge sampling*, *edge counting*, and *triangle counting*. Part of our contribution is that we consider three different settings, differing in the way in which one may access the neighborhood of a given vertex. In previous work, people have considered *indexed neighbor access*, with a query returning the i-th neighbor of a given vertex. *Full neighborhood access model*, which has a query that returns the entire neighborhood at a unit cost, has recently been considered in the applied community. Between these, we propose *hash-ordered* neighbor access, inspired by coordinated sampling, where we have a global fully random hash function, and can access neighbors in order of their hash values, paying a constant for each accessed neighbor.

For edge sampling and counting, our new lower bounds are in the most powerful full neighborhood access model. We provide matching upper bounds in the weaker hash-ordered neighbor access model. Our new faster algorithms can be provably implemented efficiently on massive graphs in external memory and with the current APIs for, e.g., Twitter or Wikipedia. For triangle counting, we provide a separation: a better upper bound with full neighborhood access than the known lower bounds

with indexed neighbor access. The technical core of our paper is our edge-sampling algorithm on which the other results depend. We now describe our results on the classic problems of edge and triangle counting.

We give an algorithm that uses hash-ordered neighbor access to approximately count edges in time $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}} + \frac{1}{\varepsilon^2})$ (compare to the state of the art without hash-ordered neighbor access of $\tilde{O}(\frac{n}{\varepsilon^2\sqrt{m}})$ by Eden, Ron, and Seshadhri [ICALP 2017]). We present an $\Omega(\frac{n}{\varepsilon\sqrt{m}})$ lower bound for $\varepsilon \geq \sqrt{m}/n$ in the full neighborhood access model. This improves the lower bound of $\Omega(\frac{n}{\sqrt{\varepsilon m}})$ by Goldreich and Ron [Rand. Struct. Alg. 2008]) and it matches our new upper bound for $\varepsilon \geq \sqrt{m}/n$. We also show an algorithm that uses the more standard assumption of pair queries ("are the vertices u and v adjacent?"), with time complexity of $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}} + \frac{1}{\varepsilon^4})$. This matches our lower bound for $\varepsilon \geq m^{1/6}/n^{1/3}$.

Finally, we focus on triangle counting. For this, we use the full power of the full neighbor access. In the indexed neighbor model, an algorithm that makes $\tilde{O}(\frac{n}{\varepsilon^{10/3}T^{1/3}} + \min(m, \frac{m^{3/2}}{\varepsilon^3 T}))$ queries for T being the number of triangles, is known and this is known to be the best possible up to the dependency on ε (Eden, Levi, Ron, and Seshadhri [FOCS 2015]). We improve this significantly to $\tilde{O}(\min(n, \frac{n}{\varepsilon T^{1/3}} + \frac{\sqrt{nm}}{\varepsilon^2\sqrt{T}}))$ full neighbor accesses, thus showing that the full neighbor access is fundamentally stronger for triangle counting than the weaker indexed neighbor model. We also give a lower bound, showing that this is the best possible with full neighborhood access, in terms of n, m, T.

Introduction

In this paper, we consider three well-studied problems from the area of sublinear-time algorithms: edge sampling, edge counting, and triangle counting in a graph. All of the three problems we consider have been studied before (in their approximate versions); see Eden and Rosenbaum [2018a], Eden et al. [2020] for edge sampling, Feige [2006a], Goldreich and Ron [2008], Seshadhri [2015], Eden et al. [2017a], Dasgupta et al. [2014b] for edge counting, and Itai and Rodeh [1977], Kolountzakis et al. [2012], Eden et al. [2015], Bera and Seshadhri [2020] for triangle counting. We first give an algorithm for exact edge sampling. We then apply this algorithm to both edge and triangle counting. We consider these three problems both in the well-studied indexed neighbor model[1], but also in two new models that we introduce.

The first of them is the *full neighborhood access model.* This model has recently been considered in the applied community (Ben-Eliezer et al. [2021]), and similar settings are commonly used in practice, as we discuss in chapter 3. In this model, upon querying a vertex, the algorithm receives the whole neighborhood. To the best of our knowledge, we are the first to formally define this model and to give an algorithm with provable guarantees on both correctness and its query complexity. In this model, we get an algorithm for triangle counting significantly more efficient than what is possible in the indexed neighbor model.

We also introduce a model we call *hash-ordered neighbor access.* This is an intermediate model, stronger than the indexed neighbor model, but weaker than the full neighborhood access model. We show that for edge sampling and counting, this model is sufficient to get an algorithm that nearly matches lower bounds (which we also prove) that work in models even stronger than the full neighborhood access model. The queries provided by the hash-ordered neighbor access can be implemented efficiently (see Chapter 3). Interestingly, the same data structure can be used to implement pair queries as well as hash-ordered neighbor access. This model formalizes the fact that the algorithms for edge counting and sampling only use the full neighborhood access in a very limited way.Moreover, our algorithms for edge sampling and counting are such that they may be efficiently implemented using queries implemented by, e.g., Twitter's, and Wikipedia's APIs, well as in some external memory setting; we discuss this in Line 7.

To appreciate our bounds, note that in a graph consisting of a clique of size[2] $\approx \sqrt{m}$ and the rest being an independent set, we need $n/\sqrt{m}$ queries just to find one edge. This provides a lower bound for both edge sampling and counting.

[1] In this setting, we have the following queries: given a vertex, return its degree; given a vertex v and a number $i \leq d(v)$, return the i-th neighbor v in an arbitrary ordering; return a random vertex.

[2] We use n and m to denote the number of vertices and edges in the graph, respectively.

Dependency on ε. The focus of sublinear-time algorithms is usually on approximate solutions, as many problems cannot be solved exactly in sublinear time. We thus have some error parameter ε, that controls how close to the exact solution the output of our algorithm should be. Throughout this paper, we put emphasis not only on the dependence of the running time on n and m but also on ε. After all, ε can be polynomial in n or m (that is, it may hold $\varepsilon = \Omega(n^\delta)$ for some $\delta > 0$). While the dependency on ε in sublinear-time algorithms has often been ignored, we believe it would be a mistake to disregard it. We are not the only ones with this opinion. For example, Goldreich says in his book [Goldreich, 2018, page 200] that he "begs to disagree" with the sentiment that the dependency on ε is not important and stipulates that "the dependence of the complexity on the approximation parameter is a key issue".

Sampling edges. The problem of sampling edges has been first systematically studied by Eden and Rosenbaum [2018a] (although it was previously considered in Kaufman et al. [2004]). They show how to sample an edge pointwise ε-close to uniform (see Definition 20) in time $O(\frac{n}{\sqrt{\varepsilon m}})$ in the indexed neighbor setting, and they prove this is optimal in terms of n, m. We show that the power of this setting is sufficient to improve the time complexity exponentially in ε, to $O(\frac{n \log \varepsilon^{-1}}{\sqrt{m}}) = \tilde{O}(\frac{n}{\sqrt{m}})$.

Sampling multiple edges has recently been considered by Eden et al. [2021a]. They present an algorithm that runs in time[3] $\tilde{O}(\sqrt{s}\frac{n}{\varepsilon\sqrt{m}} + s)$ and samples s edges pointwise ε-close to uniform with high probability, but they do not prove any lower bounds. We prove that their algorithm is optimal in terms of n, m, s.

We give more efficient algorithms with the hash-ordered neighbor access. Specifically, we give the first sublinear-time algorithm that w.h.p. returns a sample of edges from *exactly* uniform distribution. It runs in expected time $\tilde{O}(\sqrt{s}\frac{n}{\sqrt{m}} + s)$, which is the same complexity as that from Eden et al. [2021a] for approximate edge sampling for constant $\varepsilon = \Omega(1)$ (we solve exact edge sampling, which is equivalent to the case $\varepsilon = 0$). We give a near-matching lower bound for all choices of n, m, s. Apart from sampling with replacement, our methods also lead to algorithms for sampling without replacement and Bernoulli sampling.[4]

Apart from being of interest in its own right, the problem of sampling multiple edges is also interesting in that it sheds light on the relationship between two standard models used in the area of sublinear algorithms. While many algorithms only use vertex accesses, many also use random edge sampling. An algorithm for uniform edge sampling then can be used to simulate

[3] The authors claim complexity $\tilde{O}(\sqrt{s}\frac{n}{\varepsilon\sqrt{m}})$, which is sublinear in s. However, one clearly has to spend at least $\Omega(s)$ time and perform $\Omega(\min(m, s))$ queries.

[4] Bernoulli sampling is defined as sampling each edge independently with some probability p.

random edge queries. Our algorithm is not only more efficient, but also has an advantage over the algorithm by Eden et al. [2021a] that it can be used as a black box. This is not possible with their algorithm as it only samples edges *approximately* uniformly.

We use this reduction between the settings with and without random edge queries in our new algorithm for triangle counting. We prove that, perhaps surprisingly, this reduction results in near-optimal complexity in terms of n, m, T (where T is the number of triangles). We also use our edge sampling algorithm for counting edges[5], also resulting in a near-optimal complexity, this time even in terms of ε, for $\varepsilon \geq \sqrt{m}/n$.

Since we consider edge sampling to be the technical core of our paper, we focus on that in this extended abstract. We defer the rest of our results to the full version of the paper.

Counting edges. The problem of counting edges in sublinear time was first considered by Feige [2006a]. In his paper, he proves a new concentration inequality and uses it to give a $2 + \varepsilon$ approximation algorithm for counting edges running in time $O(\frac{n}{\varepsilon\sqrt{m}})$. This algorithm only uses random vertex and degree queries but no neighbor access. It is also proven by Feige [2006a] that in this setting, $\Omega(n)$ time is required for $2 - \varepsilon$ approximation for any $\varepsilon > 0$.

Since we are dealing with a graph, it is natural to also consider a query that allows us to access the neighbors of a vertex. Goldreich and Ron [2008] use indexed neighbor queries to break the barrier of 2-approximation and show a $(1 \pm \varepsilon)$-approximation that runs in time $\tilde{O}(\frac{n}{\varepsilon^{4.5}\sqrt{m}})$. They also present a lower bound of $\Omega(\frac{n}{\sqrt{\varepsilon m}})$. To prove this lower bound, they take a graph with m edges and add a clique containing εm edges. To hit the clique with constant probability, $\Omega(\frac{n}{\sqrt{\varepsilon m}})$ vertex samples are required.

Using a clever trick based on orienting edges towards higher degrees, Seshadhri [2015] shows a much simpler algorithm. This approach has been later incorporated into the journal version of the paper by Eden et al. [2015] and published in that paper. The trick of orienting edges also led to an algorithm for estimating moments of the degree distribution (Eden et al. [2017a]). The moment estimation algorithm can estimate the number of edges in time $\tilde{O}(\frac{n}{\varepsilon^2\sqrt{m}})$ by estimating the first moment – the average degree. This is currently the fastest algorithm known for counting edges.

We show two more efficient algorithms that use either the pair queries or the hash-ordered neighbor access. Specifically, in this setting, we give an algorithm that approximately counts edges in time $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}} + \frac{1}{\varepsilon^2})$. This bound is strictly better than the state of the art (assuming $\varepsilon \ll 1$ and $m \ll n^2$).

[5] We do not directly use the result on sampling s edges for fixed s. Instead, we use a variant which instead samples each edge independently with some given probability p.

We also show that the (in some sense) slightly weaker setting[6] of indexed neighbor with pair queries ("are vertices u and v adjacent") is sufficient to get an algorithm with time complexity $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}} + \frac{1}{\varepsilon^4})$. This improves upon the state of the art for ε being not too small. Our methods also lead to an algorithm in the indexed neighbor access setting that improves upon the state of the art for ε small enough.

We show lower bounds that are near-matching for a wide range of ε. Specifically, we prove that $\Omega(\frac{n}{\varepsilon\sqrt{m}})$ samples are needed for $\varepsilon \geq \sqrt{m}/n$, improving in this range upon $\Omega(\frac{n}{\sqrt{\varepsilon m}})$ from Goldreich and Ron [2008]. This lower bound holds not only with full neighbor access, but also in some more general settings. For example, Twitter API implements a query that also returns the degrees of the neighbors. Our lower bound also applies to that setting.

Triangle counting. The number of triangles T in a graph can be trivially counted in time $O(n^3)$. This has been improved by Itai and Rodeh [1977] to $O(m^{3/2})$. This is a significant improvement for sparse graphs. The first improvement for *approximate* triangle counting has been given by Kolountzakis et al. [2012], who improved the time complexity to $\tilde{O}(m + \frac{m^{3/2}}{\varepsilon^2 T})$ (recall that T is the number of triangles). This has been later improved by Eden et al. [2017a] to $\tilde{O}(\frac{n}{\varepsilon^{10/3}T^{1/3}} + \min(m, \frac{m^{3/2}}{\varepsilon^3 T}))$. In that paper, the authors also prove that their algorithm is near-optimal in terms of n, m, T.

Variants of the full neighborhood access model are commonly used in practice, and the model has been recently used in the applied community (Ben-Eliezer et al. [2021]). Perhaps surprisingly, no algorithm performing asymptotically fewer queries than the algorithm by Eden et al. [2017a] is known in this setting. Since the number of neighborhood queries is often the bottleneck of computation (the rate at which one is allowed to make requests is often severely limited), more efficient algorithms in this model could significantly decrease the cost of counting triangles in many real-world networks. We fill this gap by showing an algorithm that performs $\tilde{O}(\min(n, \frac{n}{\varepsilon T^{1/3}} + \frac{\sqrt{nm}}{\varepsilon^2\sqrt{T}}))$ queries. This is never worse than $\tilde{O}(\frac{n}{\varepsilon^{10/3}T^{1/3}} + \frac{m^{3/2}}{\varepsilon^3 T})$ and it is strictly better when $T \ll m^{9/4}/n^{3/2}$ or $\varepsilon \ll 1$. This also improves the complexity in terms of n, m, T. Our result also proves a separation between the two models, as the algorithm by Eden et al. [2017a] is known to be near-optimal in terms of n, m, T in the indexed neighbor model. Our triangle counting algorithm relies on our result for sampling edges, showcasing the utility of that result. Using the algorithm of Eden and Rosenbaum [2018a] to simulate the random edge queries would result in both worse dependency on ε and a more complicated analysis. We also prove near-matching lower bounds in terms of n, m, T.

[6] Any algorithm with pair queries with time/query complexity Q can be simulated in $O(Q \log\log n)$ time/queries using hash-ordered neighbor access. We discuss this in Chapter 3

Setting without random vertex/edge queries. If we are not storing the whole graph in memory, the problem of sampling vertices in itself is not easy. There has been work in the graph mining community that assumes a model where random vertex or edge queries are not available and we are only given a seed vertex. The complexity of the algorithms is then parameterized by an upper bound on the mixing time of the graph. The problem of sampling vertices in this setting has been considered by Chierichetti and Haddadan [2018], Ben-Eliezer et al. [2021]. The problems of approximating the average degree has been considered by Dasgupta et al. [2014b]. Counting triangles in this setting has been considered by Bera and Seshadhri [2020].

What *is* a vertex access?

Motivation behind full neighborhood access. All sublinear-time algorithms with asymptotic bounds on complexity published so far assume only a model which allows for indexed neighbor access. However, this model is usually too weak to model the most efficient ways of processing large graphs, as non-sequential access to the neighborhood is often not efficient or not possible at all. The full neighborhood access model attempts to capture this. For example, to access the i-th neighbor of a vertex in the Internet graph[7], one has to, generally speaking, download the whole webpage corresponding to the vertex. Similarly, when accessing a real-world network through an API (this would usually be the case when accessing the Twitter graph, Wikipedia graph, etc.), while it is possible to get just the i-th neighbor of a vertex, one may often get the whole neighborhood at little additional cost. The reason is that the bottleneck is usually the limit on the allowed number of queries in some time period and in one query, one may usually get many neighbors. While there is usually a limit on this number of neighbors that can be fetched in one query, this limit is often large enough that for the vast majority of vertices, the whole neighborhood can be returned as a response to one query.

As a sidenote, this limit is typically larger than the average degree. For example, in the case of Twitter, the average degree is 1414 MacCarthy [2016][8] while the limit is 5000 Twi. This fact can be used to formally prove that our edge sampling and counting algorithms can be efficiently implemented using standard API calls, as provided for example by Twitter or Wikipedia, as well as when the graph is stored on a hard drive (discussed below). We discuss the details of this in Line 7.

Although the above-mentioned APIs do not support random vertex queries, there are methods that implement the random vertex query and are efficient

[7] Internet graph is a directed graph with vertices being webpages and a directed edge for each link.

[8] The study considers the average number of followers of *an active account* but also count follows by inactive accounts. The actual average degree is thus likely somewhat lower. This number however suffices for our argument.

in practice (Ben-Eliezer et al. [2021], Chiericetti et al. [2016]). In the full neighborhood access model, we do not assume any specific vertex sampling method; any algorithm implementing the random vertex query may be used. This further strengthens the case for our model. If it takes several API calls to get one random vertex, one may perform multiple API calls on the neighborhood of each randomly sampled vertex without significantly increasing the complexity.

Another motivation for the full neighborhood access comes from graphs stored in external memory. When storing the graph on a hard drive, one may read $\approx$ 1-3 MB in the same amount of time as the overhead caused by a non-linear access[9]. If each neighbor is stored in roughly 10-30 bytes, then when we access one neighbor, we may read on the order of 10^5 neighbors while increasing the running time only by a small constant factor.

These considerations suggest that an algorithm in the full neighborhood access model with lower complexity may often be preferable to an asymptotically less efficient algorithm in the weaker indexed neighbor access setting. It is also for these reasons that this model has been recently used in the applied community (Ben-Eliezer et al. [2021]).

Lower bounds and the full neighbor access. While some lower-bounds in the past have been (implicitly) shown in the full neighborhood access model (such as the one for edge counting by Goldreich and Ron [2008]), others do not apply to that setting (such as the one for triangle counting by Eden et al. [2017a]; in fact, we prove that it does not hold in the full neighborhood access model). As we have argued, there are many settings where one can easily get many neighbors of a vertex at a cost similar to getting one vertex. Lower bounds proven in the indexed neighbor access model then do not, in general, carry over to these settings. This highlights the importance of proving lower bounds in the full neighborhood model, which are applicable to such situations.

Motivation behind hash-ordered neighbor access. We introduce a model suitable for locally stored graphs, inspired by coordinated sampling. This model is also suitable for graphs stored in external memory. We call this model the hash-ordered neighbor access. It is an intermediate model, stronger than the indexed neighbor access but weaker than the full neighborhood access model.

While in the indexed neighbor model, the neighbors can be ordered arbitrarily, this is not the case with hash-ordered neighbor access. In this setting,

[9] Consider for example Seagate ST4000DM000 (Seagate [2014]) and Toshiba MG07SCA14TA (Toshiba [2019]). These are two common server hard drives. The average seek (non-linear access) times of these hard drives are 12 and 8.5 ms, respectively. Their average read speeds are 146 and 260 MB/s, respectively. This means that in the time it takes to do one seek, one can read ≈ 1.75 and ≈ 2.2 MB, respectively.

we have a global hash function $h : V \to (0,1]$ which we may evaluate. Moreover, we assume that the neighbors of a vertex are ordered with respect to their hash values.

The simplest way to implement hash-ordered neighbor access is to store for each vertex its neighborhood in an array, sorted by the hash values. This has no memory overhead as compared to storing the values in an array. One may also efficiently support hash-ordered access on dynamic graphs using standard binary search trees.

We believe that the hash-value-ordered array is also a good way to implement pair queries when storage space is scarce. We may tell whether two vertices u and v are adjacent as follows. We evaluate $h(u)$ and search the neighborhood of v for a vertex with this hash value. We use that the hash values are random, thus allowing us to use interpolation search. This way, we implement the pair query in time $O(\log \log d(v)) \leq O(\log \log n)$ (Peterson [1957], Armenakis et al. [1985]).

We believe that the hash-ordered neighbor access can be useful for solving a variety of problems in sublinear time. Specifically, we show that it allows us to sample higher-degree vertices with higher probability — something that could be useful in other sublinear-time problems.

Our techniques

In the part of this paper where we consider edge sampling, we replace each (undirected) edge by two directed edges in opposite directions. We then assume the algorithm is executed on this directed graph.

Sampling One Edge by a Random Length Random Walk

The algorithm for sampling one edge is essentially the same as that for sampling an edge in bounded arboricity graphs from Eden et al. [2019a]. We use different parameters and a completely different analysis to get the logarithmic dependence on ε^{-1}.

We call a vertex v heavy if $d(v) \geq \theta$ and light otherwise, for some parameter θ that is to be chosen later. A (directed) edge uv is heavy/light if u is heavy/light. Instead of showing how to sample edges from a distribution close to uniform, we show an algorithm that samples each (directed) edge with probability in $[(1-\varepsilon)c/(2m), c/(2m)]$ for some $c > 0$ and fails otherwise (with probability $\approx 1 - c$). One may then sample an edge $1 + \varepsilon$ pointwise close to random by doing in expectation $\approx 1/c$ repetitions.

It is easy to sample light edges with probability exactly $c/(2m) = (n\theta)^{-1}$ – one may pick vertex v at random, choose j uniformly at random from $[\theta]$ and return the j-th outgoing edge incident to the picked vertex. Return "failure" if $d(v) < j$. We now give an intuition on how we sample heavy edges.

	Upper bound previous work	Upper bound this paper	Lower bound previous work	Lower bound this paper
Sampling one edge				
Indexed neighbor	$\tilde{O}(\frac{n}{\sqrt{\varepsilon m}})$	$O(\frac{n \log \varepsilon^{-1}}{\sqrt{m}})$	$\Omega(\frac{n}{\sqrt{m}})$	
Sampling s edges				
Indexed neighbor	$\tilde{O}(\sqrt{s}\frac{n}{\varepsilon\sqrt{m}} + s)$	$\tilde{O}(\sqrt{sn} + s)$ for $m \geq \Omega(n)$		
Hash-ordered neighbor access		$\tilde{O}(\sqrt{s}\frac{n}{\sqrt{m}} + s)$	$\Omega(\frac{n}{\sqrt{m}} + s)$	$\Omega(\sqrt{s}\frac{n}{\sqrt{m}} + s)$
Edge counting				
Indexed neighbor	$\tilde{O}(\frac{n}{\varepsilon^2\sqrt{m}})$	$\tilde{O}(\frac{\sqrt{n}}{\varepsilon} + \frac{1}{\varepsilon^2})$ for $m \geq \Omega(n)$		
Hash-ordered neighbor access		$\tilde{O}(\frac{n}{\varepsilon\sqrt{m}} + \frac{1}{\varepsilon^2})$		
Indexed neighbor + pair queries		$\tilde{O}(\frac{n}{\varepsilon\sqrt{m}} + \frac{1}{\varepsilon^4})$		
Full neighborhood access			$\Omega(\frac{n}{\sqrt{\varepsilon m}})$	$\Omega(\frac{n}{\varepsilon\sqrt{m}})$ for $\varepsilon \geq \Omega(\frac{\sqrt{m}}{n})$
Triangle counting				
Indexed neighbor w/ random vertex query	$\tilde{O}(\frac{n}{\varepsilon^{10/3}T^{1/3}} + \frac{m^{3/2}}{\varepsilon^3 T})$		$\Omega(\frac{n}{T^{1/3}} + \frac{m^{3/2}}{T})$	
Indexed neighbor w/ random edge query	$\tilde{O}(m^{3/2}/(\varepsilon^2 T))$		$\Omega(m^{3/2}/T)$	
Full neighborhood access w/ random edge query		$\tilde{O}(m/(\varepsilon^2 T^{2/3}))$		$\Omega(m/T^{2/3})$
Full neighborhood access w/ random vertex query		$\tilde{O}(\frac{n}{\varepsilon T^{1/3}} + \frac{\sqrt{nm}}{\varepsilon^2\sqrt{T}})$		$\Omega(n/T^{1/3} + \sqrt{nm/T})$
Full neighborhood access w/ random vertex, edge queries		$\tilde{O}(\varepsilon^{-2}\min(\frac{m}{T^{2/3}}, \sqrt{\frac{nm}{T}}))$		$\Omega(\min(\frac{m}{T^{2/3}}, \sqrt{\frac{nm}{T}}))$

Table 3.1: Comparison of our results with previous work. Note that the empty cells in the table do not imply that nothing is known about the problems — an algorithm that works in some model can also be used in any stronger model and similarly a lower bound that holds in some model also holds in any weaker model. Any problem can be trivially solved in $O(n + m)$. We do not make this explicit in the bounds. Similarly, any stated lower bound is assumed to hold in the sublinear regime, unless specified otherwise.

We set θ such that at least one half of the neighbors of any heavy vertex are light (we need a constant factor approximation of m for this; we use one of the standard algorithms to get it). Consider a heavy vertex v. We use the procedure described above to sample a (directed) light edge uw and we consider the vertex w. Since at least one half of the incoming edges of any heavy vertex are light, the probability of picking v is in $[cd(v)/(4m), cd(v)/(2m)]$. Sampling an incident edge, we thus get that each heavy edge is sampled with probability in $[c/(4m), c/(2m)]$.

Combining these procedures for sampling light and heavy edges (we do not elaborate here on how to do this), we may sample edges such that for some $c' > 0$ we sample each light edge with probability $c'/(2m)$ while sampling each heavy edge with probability in $[c'/(4m), c'/(2m)]$ (the value c' is different from c due to combining the procedures for sampling light and heavy edges).

We now show how to reduce the factor of 2 to $1 \pm \varepsilon$. Consider a heavy vertex v. Pick a directed edge uw from the distribution of the algorithm we just described and consider w. The probability that $w = v$ is in $[\frac{3c'd(v)}{8m}, \frac{c'd(v)}{2m}]$, as we now explain. Let h_v be the fraction of neighbors of v that are heavy. Light edges are picked with probability $c'/(2m)$ and at least half of the incoming edges are light. The remaining edges are picked with probability in $[c'/(4m), c'/(2m)]$. The probability of sampling v is then $\geq (1 - h_v)c'/(2m) + h_v c'/(4m) \geq \frac{3c'}{8m}$ because $h_v \leq 1/2$. The probability is also clearly $\leq c'/(2m)$, thus proving the claim. Combining light edge sampling and heavy edge sampling (again, we do not elaborate here on how to do this), we are now able to sample an edge such that each light edge is sampled with probability $c''/(2m)$ and each heavy edge with probability in $[\frac{3c''}{8m}, \frac{c''}{2m}]$ (again, the value c'' is different from c, c' due to combining light and heavy edge sampling). Iterating this, the distribution converges pointwise to uniform at an exponential rate.

One can show that this leads to the following algorithm based on constrained random walks of random length. The length is chosen uniformly at random from $[k]$ for integer $k \approx \lg \varepsilon^{-1}$. The algorithm returns the last edge of the walk. The random walk has constraints that, when not satisfied, cause the algorithm to fail and restart. These constraints are (1) the first vertex v of the walk is light and all subsequent vertices are heavy except the last one (which may be either light or heavy) and (2) picking $X \sim Bern(d(v)/\sqrt{2m})$ [10], the first step of the walk fails if $X = 0$ (note that this is equivalent to using rejection sampling to sample the first edge of the walk).

Sampling Multiple Edges and Edge counting using Hash-Ordered Neighbor Access

The problems of edge counting and sampling share the property that in solving both these problems, it would be of benefit to be able to sample vertices in a way that is biased towards vertices with higher degree. This is clearly the case for edge sampling as it can be easily seen to be equivalent to sampling vertices with probabilities proportional to their degrees. Biased sampling is also useful for edge counting. If we let v be chosen at random from some distribution and for a fixed vertex u define $p_u = P(v = u)$, then $X = \frac{d(v)}{2p_v}$ is an unbiased estimate of the number of edges, called the Horvitz-Thompson estimator Horvitz and Thompson [1952a]. The variance is $Var(X) \leq E(X^2) = \sum_{v \in V} p_v \left(\frac{d(v)}{p(v)}\right)^2$ which decreases when high-degree vertices have larger probability p_v. The crux of this part of our paper, therefore, lies in how to perform this biased sampling.

[10] Bernoulli trial $Bern(p)$ is a random variable having value 1 with probability p and 0 otherwise.

Biased sampling procedure. We first describe how to perform biased sampling in the case of the indexed neighbor access model and then describe a more efficient implementation in the hash-ordered neighbor access model. We say a vertex v is heavy if $d(v) \geq \theta$ for some value θ and we say it is light otherwise. The goal is to find all heavy vertices in the graph as that will allow us to sample heavy vertices with higher probability. In each step, we sample a vertex and look at its neighborhood. This takes in expectation $O(m/n)$ time per step. After $\Theta(n \log(n)/\theta)$ steps, we find w.h.p. all vertices with degree at least θ. We call this technique high-degree exploration.

How to exploit hash-ordered neighbor access? We do not actually need to find all heavy vertices in order to be able to sample from them when we have the vertex hash queries. We make a sample S of vertices large enough such that, w.h.p., each heavy vertex has one of its neighbors in S ($|S| = \Theta(n \log(n)/\theta$ suffices). Pick all heavy vertices incident to the sampled vertices that have $h(v) \leq p$. We can find these vertices in constant time per vertex using hash-ordered access. Since each heavy vertex has at least one of its neighbors in S, these are in fact all heavy vertices v with $h(v) \leq p$. Since the hash values are independent and uniform on $[0, 1]$, each heavy vertex is (w.h.p.) picked independently with probability p. This allows us to sample heavy vertices with larger probability than if sampling uniformly.

We apply this trick repeatedly for $k = 1, \cdots, \log n$ with thresholds $\theta_k = 2^k\theta$ and $p_k = 2^k p$. This way, instead of having one threshold θ and one probability of being sampled for each v with $d(v) \geq \theta$, we have logarithmically many thresholds and the same number of different probabilities. Since the ratio between θ_k and p_k is constant, we see that the probability of each vertex v with $d(v) \geq \theta$ being sampled is up to a factor of 2 proportional to $d(v)$. Moreover, for each vertex, we know exactly its probability of being sampled.

Edge sampling. We describe how to sample each edge independently with some probability p. This setting is called Bernoulli sampling. Light edges can be sampled in a way similar to that used for sampling light edges in the algorithm for sampling one edge. Using the biased sampling algorithm allows us to sample heavy vertices with higher probability. We then prove that for each vertex v, the probability that v is sampled is at least the probability that one of the edges incident to v is sampled when sampling each edge independently with probability p. This allows us to use rejection sampling to sample each vertex v with a probability equal to the probability of at least one of its incident edges being sampled. By sampling k incident edges of such vertex where k is chosen from the right distribution, we get that each heavy edge is sampled independently with some fixed probability p. Sampling light and heavy edges separately and taking union of those samples gives us an algorithm that performs Bernoulli sampling from the set of all edges.

One can use Bernoulli sampling to sample edges without replacement.

Specifically, when it is desired to sample s edges without replacement, one may use Bernoulli sampling to sample in expectation, say, $2s$ samples and if at least s are sampled, then return a random subset of the random edges, otherwise repeat.

To sample s edges with replacement, we perform Bernoulli sampling $\Theta(s)$ times, setting the probability such that each time, we sample in expectation $O(1)$ edges. From each (non-empty) Bernoulli sample, we take one edge at random and add it into the sample. While implementing this naively would result in a linear dependence on s, this can be prevented. The reason is that it is sufficient to perform the pre-processing for Bernoulli sampling only once. This allows us to spend more time in the "high-degree exploration phase", making the Bernoulli sampling itself more efficient.

Edge counting by sampling. When using Bernoulli sampling with some inclusion probability p, the number of sampled edges $|S|$ is concentrated around pm. We estimate m as $|S|/p$ and prove that for an appropriate choice of p, the approximation has error at most ε with high probability. The "appropriate value" of p depends on m. We find it using a geometric search.

Counting edges directly. We also give an independent algorithm for edge counting based on a different idea. We use the biased sampling procedure to sample higher-degree vertices with higher probability. We then use the above-mentioned Horvitz-Thompson estimator; sampling higher-degree vertices with higher probability reduces the variance. We then take the average of an appropriate number of such estimators to sufficiently further reduce the variance.

Edge Counting Using Pair Queries

Seshadhri [2015] shows a bound on the variance of the following estimator: sample a vertex v, get a random neighbour u of v, if $(d(v), id(v)) < (d(u), id(u))$ [11] then answer $nd(v)$, otherwise answer 0. This is an unbiased estimate of m. We combine this idea with the technique of high-degree exploration. Direct all edges from the endpoint with lower degree to the one with higher degree. The biggest contributor to the variance of the estimator from Seshadhri [2015] are the vertices that have out-degree roughly $\sqrt{m}$ (one can show that there are no higher-out-degree vertices). Our goal is to be able to sample these high out-degree ($d^+(v) \geq \theta$ for some parameter θ) vertices with higher probability. Using the Horvitz-Thompson estimator for vertices of out-degree at least θ will decrease the variance. We use an estimator inspired by the one from Seshadhri [2015] for vertices with out-degree $< \theta$.

[11] We are assuming id is a bijection between V and $[n]$ and $<$ on the tuples is meant with respect to the lexicographic ordering.

We make a sample S of vertices and let S' be the subset of S of vertices having degree $\geq \theta$. The intuition for why we consider S' is the following. If we pick S to be large enough ($|S| = \Theta(n \log(n)/\theta)$ is sufficient), any vertex v with $d^+(v) \geq \theta$ will have at least one of its out-neighbors sampled in S. Moreover, it holds $d(v) \geq d^+(v) \geq \theta$; since we direct edges towards higher-degree endpoints, these sampled out-neighbors also have degree $\geq \theta$. This means that they also lie in S'. Any high-out-degree vertex thus has, w.h.p., a neighbor in S'. At the same time, S' has the advantage of being significantly smaller than S as there can only be few vertices with degree $\geq \theta$ in the graph (at most $2m/\theta$, to be specific), so each vertex of S lies in S' with probability $\leq 2m/(n\theta)$.

Now we pick each incident edge to S' with a fixed probability p and for each picked edge uv for $u \in S'$, mark the vertex v. A vertex v is then marked with probability $p_v = 1 - (1-p)^{r(v)}$ where $r(v) = |N(v) \cap S'|$. It then holds $p_v \approx pr(v)$ (we ensure $r(v)$ is not too large, which is needed for this to hold). This is (w.h.p.) roughly proportional to the out-degree of v. Using the Horvitz-Thompson estimator, this reduces the variance. This suffices to get the improved complexity.

It remains to show how to efficiently compute $r(v)$ (we need to know the value in the Horvitz-Thompson estimator). We set the threshold θ so as to make sure that only a small fraction of all vertices can have degree greater than the threshold. Then, S' will be much smaller than S (as S is a uniform sample). In fact, it will be so small that we can afford to use pair queries to check which of the vertices in S' are adjacent to v. This is the main trick of our algorithm.

Another obstacle that we have to overcome is that when we are given a vertex, we cannot easily determine its out-degree. We need to know this to decide which of the above estimators to use for the vertex. Fortunately, the cost of estimating it is roughly inverse to the probability we will need the estimate, thus making the expected cost small.

Triangle Counting Using Full Neighborhood Access

Warmup: algorithm with random edge queries. We now show a warmup which assumes that we may sample edges uniformly at random. We then use this as a starting point for our algorithm. This warmup is inspired by and uses some of the techniques used in Kallaugher et al. [2019].

Consider an edge $e = uv$. By querying both endpoints, we can determine the number of triangles containing e (because this number is equal to $|N(u) \cap N(v)|$). Let $t(e)$ be the number of triangles containing e. One of the basic ideas that we use is that we assign each triangle to its edge with the smallest value $t(e)$. This trick has been used before for edge counting in Seshadhri [2015] and for triangle counting in Kallaugher et al. [2019]. We denote by $t^+(e)$ the number of edges assigned to e. Consider a uniformly random edge $e = uv$ and a uniform vertex $w \in N(u) \cap N(v)$. Let $X = t(e)$ if uvw is assigned to uv and $X = 0$ otherwise. The expectation is $E(X) = \sum_{e \in E} \frac{1}{m} \cdot \frac{t^+(e)}{t(e)} t(e) = T/m$ and we may thus give an unbiased estimator of T as mX. The variance of X is $Var(X) \leq \sum_{e \in E} \frac{1}{m} \cdot \frac{t^+(e)}{t(e)} t(e)^2 = \frac{1}{m} \sum_{e \in E} t^+(e)t(e)$. A bound used in Kallaugher et al. [2019] can be used to prove that this is $O(T^{4/3}/m)$. Taking $s = \Theta(m/(\varepsilon^2 T^{2/3}))$ samples and taking the average then gives a good estimate with probability at least 2/3 by the Chebyshev inequality.

We prove that this is optimal up to a constant factor in terms of m and T when only random edge queries (and not random vertex queries) are allowed. We will now consider this problem in the full neighborhood access model, which only allows for random vertex queries (and not random edge queries). Combining this algorithm with our edge sampling algorithm results in complexity $O(\sqrt{s}\frac{n}{\sqrt{m}} + s) = O(\frac{n}{\varepsilon T^{1/3}} + \frac{m}{\varepsilon^2 T^{2/3}})$ in the full neighborhood access model. This, however, is not optimal.

Sketch of our algorithm: algorithm with random vertex queries. We will now describe a more efficient algorithm that uses both random edge and random vertex queries. We then remove the need for random edge queries by simulating them with our algorithm for edge sampling. Perhaps surprisingly, black-box application of our edge sampling algorithm results in near-optimal complexity.

In order to find out the value $t(e)$ for as many edges as possible, one may sample each vertex independently with some probability p. For any edge e whose both endpoints have been sampled, we can compute $t(e)$ with no additional queries. The sum of $t(e)$'s that we learn in this way is in expectation $3p^2T$. We may thus get an unbiased estimator of T. We may express the variance by the law of total variance and a standard identity for the variance of a sum. By doing this, we find out that there are two reasons the variance is high. First, the variance of the number of triangles contributed to the estimate by one edge can be large[12]. This is true for edges that are contained in relatively many triangles. The second reason is the correlations between edges: if we have edges e, e' sharing one vertex and both endpoints of e have been sampled, it is more likely that both endpoints of e' are sampled, too. This introduces correlation between the contributions coming from different edges, thus increasing the variance. We now sketch a solution to both these issues.

The first issue could be solved by applying the above-described trick with assigning each triangle to its edge $e = uv$ with the smallest value $t(e)$. Pick w uniformly from $N(u) \cap N(v)$ and let $X_e = t(e)$ if uvw is assigned to uv and let $X_e = 0$ otherwise. An analysis like the one described above would give good bounds on the variance of X_e. The issue with this is that for each

[12] Formally, we are talking about the variance of a random variable X_e equal to $t(e)$ if both endpoints of e are sampled and 0 otherwise.

edge with non-zero $t(e)$, we query the vertex w. If the number of edges in the subgraph induced by the sampled edges is much greater than the number of sampled vertices, this will significantly increase the query complexity. To solve this issue, we separately consider two situations. If an edge has many triangles assigned to it, we do the following. We sample a set of vertices N (this set is shared for all edges) large enough such that, by the Chernoff bound, $\frac{|\{w \in N \cap N(u) \cap N(v), \text{ s.t. } uvw \text{ is assigned to } uv\}|}{|N \cap N(u) \cap N(v)|} \approx t^+(e)/t(e)$. Instead of sampling w uniformly from $N(u) \cap N(v)$, we then sample from $N \cap N(u) \cap N(v)$. Since we do not need to query any vertex twice, we may bound the cost of this by $|N|$. On the other hand, if the number of triangles assigned to e is small, the variance of X_e (which is proportional to $t^+(e)t(e)$) is relatively small (as $t^+(e)$ is small). We then may afford to only use the estimator X_e with some probability p' and if we do, we use X_e/p' as the (unbiased) estimate of $t^+(e)$. This increases the variance contributed by the edge e, but we may afford this as it was small before applying this trick. This way, we have to only make an additional query with probability p', thus decreasing the query complexity of this part of the algorithm.

The second problem is created by vertices whose incident edges have many tiangles (at least θ) assigned to them (as becomes apparent in the analysis). A potential solution would be to not use this algorithm for the vertices with more than θ triangles assigned to incident edges and instead estimate the number of these triangles by the edge-sampling-based algorithm from the warm-up. Why is this better than just using that algorithm on its own? There cannot be many such problematic vertices. Namely, there can only be $\ell = T/\theta$ such vertices. This allows us to get a better bound on the variance. Specifically, instead of bounding the variance by $O(T^{4/3}/m)$, we prove a bound of $\tilde{O}(\ell T/m)$. An issue we then have to overcome is that we cannot easily tell apart the "heavy" and "light" vertices. Let us have a vertex v and we want to know whether it is light (it has at most θ triangles assigned to its incident edges) or whether it is heavy. The basic idea is to sample some vertices (this set is common for all vertices) and then try to infer whether a vertex is with high probability light or whether it may potentially be heavy based on the number of triangles containing edges between v and this set of sampled vertices.

These techniques lead to a bound of $\tilde{O}(\frac{n}{\varepsilon T^{1/3}} + \frac{\sqrt{nm}}{\varepsilon^2 \sqrt{T}})$. One can always read the whole graph in n full neighborhood queries, leading to a bound of $\tilde{O}(\min(n, \frac{n}{\varepsilon T^{1/3}} + \frac{\sqrt{nm}}{\varepsilon^2 \sqrt{T}}))$.

Lower bounds. Our lower bounds match our algorithms in terms of dependency on n, m, T. The lower bound of $\Omega(n/T^{1/3})$ is standard and follows from the difficulty of hitting a clique of size $T^{1/3}$. We thus need to prove a lower bound of $\Omega(\min(n, \sqrt{nm/T}))$. This amounts to proving $\Omega(\sqrt{nm/T})$ under the assumption $T \geq m/n$. We thus assume for now this inequality.

Our lower bound is by reduction from the OR problem (given booleans

$x_1, \cdots, x_n$, compute $\bigvee_{i=1}^n x_i$) in a style similar to the reductions in Eden and Rosenbaum [2018b]. The complexity of the OR problem is $\Omega(n)$. For an instance of the OR problem of size $\sqrt{nm/T}$, we define a graph G with $\Theta(n)$ vertices and $\Theta(m)$ edges. The number of triangles is either $\geq T$ if $\bigvee_{i=1}^n x_i = 1$ or 0 if $\bigvee_{i=1}^n x_i = 0$. Moreover, any query on G can be answered by querying one x_i for some $i \in [n]$. It follows that any algorithm that solves triangle counting in G in Q queries can be used to solve the OR problem of size $\Theta(\sqrt{nm/T})$ in Q queries. This proves the desired lower bound.

We now describe the graph G. We define a few terms. A section consists of 4 groups of $\sqrt{nT/m}$ vertices. The whole graph consists of sections and m/n non-section vertices. There are $\sqrt{nm/T}$ sections, one for each x_i. In the i-th section, there is a complete bipartite graph between the first two groups of vertices if $x_i = 0$ and between the third and fourth if $x_i = 1$. There is a complete bipartite graph between each third or fourth group of a section and the non-section vertices. See Figure 3.1 on page 123 for an illustration of this construction.If $x_i = 0$ for all i, then G is triangle-free. If $x_i = 1$, then the i-th section together with the non-section vertices forms $\sqrt{nT/m} \cdot \sqrt{nT/m} \cdot m/n = T$ triangles. At the same time, a query "within a section" only depends on the value x_i corresponding to that section, so we can implement it by one query to the instance of the OR problem. A query that does not have both endpoints within the same section is independent of the OR problem instance. The number of vertices and edges is $\Theta(n)$ and $\Theta(m)$ as desired, and G thus satisfies all conditions.

Preliminaries

Graph access models

Since a sublinear-time algorithm does not have the time to pre-process the graph, it is important to specify what queries the algorithm may use to access the graph. We define the *indexed neighbor access model* by the following queries:

- For $i \in [n]$, return the i-th vertex in the graph
- For $v \in V$, return $d(v)$
- For $v \in V$ and $i \in [d(v)]$, return the i-th neighbor of v
- For a given vertex v, return $id(v)$ such that if v is the i-th vertex, then $id(v) = i$

where the vertices in the graph as well as the neighbors of a vertex are assumed to be ordered adversarially. Moreover, the algorithm is assumed to know n. This definition is standard; see Goldreich and Ron [2008] for more details. *Pair* *queries* are often assumed to be available in addition to the queries described above:

- Given vertices u, v, return whether the two vertices are adjacent

This has been used, for example, in Eden et al. [2017a], Eden and Rosenbaum [2018a], Assadi [2020].

In this paper, we introduce a natural extension of the above-described setting without pair queries, which we call the *hash-ordered neighbor access model.* In this model, there is the following additional query

- For $v \in V$, return $h(v)$

where the hash of v, denoted $h(v)$, is a number picked independently uniformly at random from $[0, 1]$. Moreover, neighborhoods of vertices are assumed to be ordered with respect to the hashes of the vertices. Our algorithms do not require the vertices to be ordered with respect to the hash values (in contrast to the neighborhoods), although that would also be a natural version of this model.

We define the *full neighborhood access model* as follows. Each vertex v has a unique $id(v) \in [n]$. We then have one query: return the *id*'s of all neighbors of the i-th vertex. We then measure the complexity of an algorithm by the number of queries performed, instead of the time complexity of the algorithm.

Pointwise ε-Approximate sampling

Definition 20. *A discrete probability distribution $\mathcal{P}$ is said to be pointwise ε-close to $\mathcal{Q}$ where $\mathcal{P}, \mathcal{Q}$ are assumed to have the same support, denoted $|\mathcal{P} - \mathcal{Q}|_P \leq \varepsilon$, if*

$$|\mathcal{P}(x) - \mathcal{Q}(x)| \leq \varepsilon \mathcal{Q}(x), \qquad \textit{or equivalently} \quad 1 - \varepsilon \leq \frac{\mathcal{P}(x)}{\mathcal{Q}(x)} \leq 1 + \varepsilon$$

for all x from the support.

In this paper, we consider distributions pointwise ε-close to uniform. This measure of similarity of distributions is related to the total variational distance. Specifically, for any $\mathcal{P}, \mathcal{Q}$, it holds that $|\mathcal{P} - \mathcal{Q}|_{TV} \leq |\mathcal{P} - \mathcal{Q}|_P$ Eden and Rosenbaum [2018a].

Conditioning principle

Let X, Y be two independent random variables taking values in a set A and let f be a function on A. If $Y \sim f(X)$, then $X \sim X|(f(X) = Y)$. In other words, if we want to generate a random variable X from some distribution, it is sufficient to be able to generate (1) a random variable from the distribution conditional on some function of X and (2) a random variable distributed as the function of X. We call this the conditioning principle. We often use this to generate a sample – we first choose the sample size from the appropriate distribution and then sample the number of elements accordingly.

Notation

We use relations $f(x) \lesssim g(x)$ with the meaning that $f(x)$ is smaller than $g(x)$ up to a constant factor. The relations $\simeq, \gtrsim$ are defined analogously. The notation $f(x) \sim g(x)$ has the usual meaning of $f(x)/g(x) \to 1$ for $x \to \infty$. We use $\lg x$ to denote the binary logarithm of x. We use $N(v)$ to denote the set of neighbors of v. Given a vertex v and integer i, we let $v[i]$ to be the i-th neighbor of v. Given a (multi-)set of vertices S, we let $d(S) = \sum_{v \in S} d(v)$ and $d_S(v) = |N(v) \cap S|$. For an edge $e = uv$, we denote by $N(e)$ the set of edges incident to either u or v. In addition to sampling with and without replacement, we use the less standard name of Bernoulli sampling. In this case, each element is included in the sample independently with some given probability p which is the same for all elements. Given a priority queue $\mathcal{Q}$, the operation $\mathcal{Q}.top()$ returns the elements with the lowest priority. $\mathcal{Q}.pop()$ returns the element with the lowest priority and removes it from the queue.

We use $Bern(p)$ to denote a Bernoulli trial with bias p, $Unif(a, b)$ to be the uniform distribution on the interval $[a, b]$, $Bin(n, p)$ to be the binomial distribution with universe size n and sample probability p. For distribution $\mathcal{D}$ and event $\mathcal{E}$, we use $X \sim (\mathcal{D}|\mathcal{E})$ to denote that X is distribution according to the conditional distribution $\mathcal{D}$ given $\mathcal{E}$.

Algorithms with advice

Many of our algorithms depend on the value of m or T. We are however not assuming that we know this quantity (in fact, these are often the quantities we want to estimate). Fortunately, for our algorithms, it is sufficient to know this value only up to a constant factor. There are standard techniques that can be used to remove the need for this advice.

Specifically, we use the advice removal procedure from Tětek [2021]. Similar advice removal procedures have been used before, for example in Eden et al. [2017a], Goldreich and Ron [2006], Aliakbarpour et al. [2018a], Assadi et al. [2019b], Eden and Rosenbaum [2018a]. This advice removal can be summarized as follows.

Fact 21. *Let us have a graph parameter ϕ that is polynomial in n. Suppose there is an algorithm that takes as a parameter $\tilde{\phi}$ and has time complexity $T(n, \tilde{\phi}, \varepsilon)$ decreasing polynomially in $\tilde{\phi}$. Moreover, assume that for some $c > 1$, it outputs $\hat{\phi}$ such that $P(\hat{\phi} \geq c\phi) \leq 1/3$ and if moreover $\phi \leq \tilde{\phi} \leq c\phi$, then $P(|\hat{\phi} - \phi| > \varepsilon\phi) \leq 1/3$. Then there exists an algorithm that has time complexity $O(T(n, \phi, \varepsilon) \log\log n)$ and returns $\hat{\phi}$ such that $P(|\hat{\phi} - \phi| > \varepsilon\phi)$ (and does not require advice $\tilde{\phi}$).*

We now give a sketch of the reduction. We start with $\tilde{\phi}$ with a polynomial upper-bound on ϕ. For m, this may be n^2, for T, we can use n^3. We then geometrically decrease $\tilde{\phi}$. For each value of $\tilde{\phi}$, we perform probability amplification. Specifically, we have the algorithm run $\Theta(\log \log n)$ times and take the median estimate. We stop when this median estimate is $\geq c\tilde{\phi}$ and return the estimate. See [Tětek, 2021, Section 2.5] for details.

Sampling without replacement

Suppose we want to sample k elements with replacement from the set $[n]$. This can be easily done in expected time $O(k)$. If $k > n/2$, we instead sample $n-k$ items without replacement and take the complement. We may, therefore, assume that $k \leq n/2$. We repeat the following k times: we sample items with replacement until we get an element we have not yet seen before, which we then add into the sample. Since $k \leq n/2$, each repetition takes in expectation $O(1)$ time and the total time is in expectation $O(k)$.

Edge sampling

We start with some definitions which we will be using throughout this section. Given a threshold θ (the exact value is different in each algorithm), we say a vertex v is heavy if $d(v) \geq \theta$ and light if $d(v) < \theta$. We denote the set of heavy (light) vertices by V_H (V_L). In this section, we replace each unoriented edge in the graph by two oriented edges in opposite directions. We then assume the algorithm is executed on this oriented graph. We then call a (directed) edge uv heavy (light) if u is heavy (light). If we can sample edges from this oriented graph, we can also sample edges from the original graph by sampling an edge and forgetting its orientation.

Sampling one edge in the indexed neighbor access model

In this section, we show Algorithm 10 which samples an edge pointwise ε-approximately in expected time $O(\frac{n}{\sqrt{m}} \log \frac{1}{\varepsilon})$. This algorithm is, up to a change of parameters, the one used in Eden et al. [2019a] but we provide a different analysis that is tighter in the case of general graphs (in Eden et al. [2019a], the authors focus on graphs with bounded arboricity). This algorithm works by repeated sampling attempts, each succeeding with probability $\approx \frac{\sqrt{m}}{n \log \varepsilon^{-1}}$. We then show that upon successfully sampling an edge, the distribution is pointwise ε-close to uniform.

We show separately for light and heavy edges that they are sampled almost uniformly. The case of light edges (Observation 22) is analogous to the proof in Eden and Rosenbaum [2018a]. We include it here for completeness.

Algorithm 9: `Sampling_attempt(`k`)` subroutine

```
$\theta \leftarrow \lceil \sqrt{2m} \rceil$
Sample a vertex $u_0 \in V$ uniformly at random
If $u_0$ is heavy, return "failure"
Choose a number $j \in [\theta]$ uniformly at random
Let $u_1$ be the $j$'th neighbor of $u_0$; return "failure" if $d(u_0) < j$
for $i$ from 2 to $k$ do
    If $u_{i-1}$ is light, return "failure"
    $u_i \leftarrow$ random neighbor of $u_{i-1}$
Return $(u_{k-1}, u_k)$.
```

Algorithm 10: Sample an edge from distribution pointwise ε-close to uniform

```
1 Pick $k$ from $\{1, \cdots, \ell = \lceil \lg \frac{1}{\varepsilon} \rceil + 2\}$ uniformly at random
2 Call Sampling_attempt(k), if it fails, go back to line 1, otherwise
    return the result
```

Observation 22. *For k chosen uniformly from $[\ell]$, any fixed light edge $e = uv$ is chosen by Algorithm 9 with probability $\frac{1}{\ell n \theta}$*

Proof. The edge uv is chosen exactly when $k = 1$ (this happens with probability $\frac{1}{\ell}$), $u_0 = u$ (happens with probability $\frac{1}{n}$), and j is such that v is the j-th neighbor of u (happens with probability $\frac{1}{\theta}$). This gives total probability of $\frac{1}{\ell n \theta}$. □

We now analyze the case of heavy edges. Before that, we define for v being a heavy vertex $h_{v,1} = \frac{d_H(v)}{d(v)}$ and for $i \geq 2$

$$h_{v,i} = h_{v,1} \sum_{w \in n_H(v)} h_{w,i-1}/d_H(v)$$

For light vertices, the h-values are not defined.

Lemma 23. *For k chosen uniformly from $[\ell]$, for any heavy edge vw*

$$P(u_{k-1} = v, u_k = w | k \geq 2) = (1 - h_{v,\ell-1}) \frac{1}{(\ell - 1) n \theta}$$

Proof. Let k be chosen uniformly at random from $\{2, \cdots, r\}$. We show by induction on r that $P(u_{k-1} = v | r) = (1 - h_{v,r-1}) \frac{d(v)}{(r-1) n \theta}$ for any heavy vertex v. If we show this, the lemma follows by substituting $r = \ell$ and by uniformity of u_k on the neighborhood of u_{k-1}.

For $r = 2$, the claim holds because when $k = 2$, there is probability $\frac{1}{n\theta}$ that we come to v from any of the $(1 - h_{v,1}) d(v)$ adjacent light vertices.

We now show the induction step. In the following calculation, we denote by $P_r(\mathcal{E})$ the probability of event $\mathcal{E}$ when k is chosen uniformly from $[r]$. Consider some vertex v and take a vertex $w \in N(v)$. It now holds $P_r(u_{k-2} = w) = P_{r-1}(u_{k-1} = w)$. We have

$$\begin{aligned}
P_r(u_{k-1} = v) &= \sum_{w \in N(v)} P_r(u_{k-2} = w) P(u_{k-1} = v | u_{k-2} = w) && (3.1)\\
&= \sum_{w \in N_L(v)} P(u_0 = w) P(u_1 = v | u_0 = w) P(k = 2) && (3.2)\\
&\quad + \sum_{w \in N_H(v)} P_{r-1}(u_{k-1} = w | k > 1) P(u_k = v | u_{k-1} = w) P(k > 2) && (3.3)\\
&= \sum_{w \in N_L(v)} \frac{1}{n}\frac{1}{\theta}\frac{1}{r-1} + \sum_{w \in N_H(v)} (1 - h_{w,r-2}) \frac{d(w)}{(r-2)n\theta}\frac{1}{d(w)}\frac{r-2}{r-1} && (3.4)\\
&= (1 - h_{v,2}) \frac{d(v)}{(r-1)n\theta} + \sum_{w \in N_H(v)} (1 - h_{w,r-2}) \frac{1}{(r-1)n\theta} && (3.5)\\
&= (1 - h_{v,1}) \frac{d(v)}{(r-1)n\theta} + h_{v,1} d(v) \Big(1 - \sum_{w \in N_H(v)} h_{w,r-2}/d_H(v)\Big) \frac{1}{(r-1)n\theta} && (3.6)\\
&= (1 - h_{v,1}) \frac{d(v)}{(r-1)n\theta} + (h_{v,1} - h_{v,r-1}) \frac{d(v)}{(r-1)n\theta} && (3.7)\\
&= (1 - h_{v,r}) \frac{d(v)}{(r-1)n\theta} && (3.8)
\end{aligned}$$

□

Before putting it all together, we will need the following bound on $h_{v,i}$.

Lemma 24. *For any $v \in V_H(G)$ and $k \geq 1$ it holds that*

$$h_{v,k} \leq 2^{-k}$$

Proof. We first prove that for any $v \in V(G)$, it holds that $h_{v,1} \leq 1/2$. This has been shown in Eden and Rosenbaum [2018a] and we include this for completeness. We then argue by induction that this implies the lemma.

Since v is heavy, it has more than θ neighbors. Moreover, there can be at most $\frac{m}{\theta}$ heavy vertices, meaning that the fraction of heavy neighbors of v can be bounded as follows

$$h_{v,1} \leq \frac{m}{\theta}\frac{1}{\theta} = \frac{m}{\lceil\sqrt{2m}\rceil}\frac{1}{\lceil\sqrt{2m}\rceil} \leq \frac{1}{2}$$

We now show the claim by induction. We have shown the base case and it therefore remains to prove the induction step:

$$h_{v,i} = h_{v,1} \sum_{w \in n_H(v)} h_{w,i-1}/d_H(v) \leq \frac{1}{2} \sum_{w \in n_H(v)} 2^{-(i-1)}/d_H(v) = 2^{-i}$$

□

We can now prove the following theorem

Theorem 25. *For $\varepsilon \leq \frac{1}{2}$, the Algorithm 10 runs in expected time $O(\frac{n}{\sqrt{m}} \log \frac{1}{\varepsilon})$ and samples an edge from a distribution that is pointwise ε-close to uniform.*

Proof. We first prove the correctness and then focus on the time complexity.

Correctness. We first show that in an iteration of Algorithm 10, each edge is sampled with probability in $[(1-\varepsilon)\frac{1}{\ell n \theta}, \frac{1}{\ell n \theta}]$. For light edges, this is true by Observation 22. We now prove the same for heavy edges. Similarly, a heavy edge (v, w) is chosen when $k \geq 2$, $u_{k-1} = v$ and $w = u_k$. Using Lemma 23,

$$\begin{aligned} P(k \geq 2, u_{k-1} = v, u_k = w) &= P(k \geq 2) P(u_{k-1} = v, u_k = w | k \geq 2) && (3.9) \\ &= \frac{\ell - 1}{\ell}(1 - h_{v,\ell-1}) \frac{1}{(\ell-1) n \theta} && (3.10) \\ &= (1 - h_{v,\ell-1}) \frac{1}{\ell n \theta} && (3.11) \end{aligned}$$

We can now use Lemma 24 to get a lower bound of

$$\geq (1 - 2^{-\ell+1}) \frac{1}{\ell n \theta} \geq (1 - \tfrac{1}{2}\varepsilon) \frac{1}{\ell n \theta} \qquad (3.12)$$

Similarly, since the value $h_{v,\ell}$ is always non-negative, it holds

$$P(k \geq 2, u_{k-1} = v, u_k = w) \leq \frac{1}{\ell n \theta} \qquad (3.13)$$

Consider one execution of Algorithm 9 with k chosen uniformly from $[\ell]$ and let $\mathcal{S}$ denote the event that the execution does not end with failure. Let e be the sampled edge and e', e'' some fixed edges. Then since $P(e = e' | \mathcal{S}) = \frac{P(e=e')}{P(\mathcal{S})}$ and for any fixed e' it holds that

$$(1 - \tfrac{1}{2}\varepsilon) \frac{1}{\ell n \theta} \leq P(e' = e) \leq \frac{1}{\ell n \theta}$$

it follows that

$$1 - \tfrac{1}{2}\varepsilon \leq \frac{P(e = e')}{P(e = e'')} \leq (1 - \tfrac{1}{2}\varepsilon)^{-1} \leq 1 + \varepsilon$$

where the last inequality holds because $\varepsilon \leq \frac{1}{2}$. Algorithm 9 perform sampling attempts until one succeeds. This means that the returned edge comes from the distribution conditional on $\mathcal{S}$. As we have noted, this scales the sampling probabilities of all edges by the same factor of $P(\mathcal{S})$ and the output distribution is, therefore, pointwise ε-close to uniform.

Time complexity. Consider again one execution of Algorithm 9. Since for every fixed edge e', the probability that $e = e'$ is at least $(1-\varepsilon)\frac{1}{\ell n\theta}$, the total success probability is

$$P(\mathcal{S}) = P(\bigvee e = e') = \sum_{e' \in E} P(e = e') \geq m(1-\varepsilon)\frac{1}{\ell n\theta}$$

where the second equality holds by disjointness of the events. The expected number of calls of Algorithm 9 is then

$$\frac{\ell n\theta}{(1-\varepsilon)m} = \frac{\sqrt{2}(\lceil \lg \frac{1}{\varepsilon} \rceil + 1)n}{(1-\varepsilon)\sqrt{m}} = O\left(\frac{n}{\sqrt{m}} \log \frac{1}{\varepsilon}\right)$$

Each call of Algorithm 9 takes in expectation $O(1)$ time because in each step of the random walk, we abort with probability at least $1/2$ (we are on a heavy vertex as otherwise we would have aborted, in order not to abort, the next vertex also must be heavy; the fraction of neighbors that are heavy is $h_{v,1} \leq 1/2$). Therefore, the complexity is as claimed. □

Biased vertex sampling using hash-ordered access

We now describe a sampling procedure (Algorithm 12) which allows us to sample vertices such that vertices with high degree are sampled with higher probability. We will then use this for sampling multiple edges and later in Line 6for approximate edge counting. The procedure is broken up into two parts – one for pre-processing and one for the sampling itself. We do not make it explicit in the pseudocode how the data structures built during pre-processing are passed around for the sake of brevity.

In the rest of this section, let $p_V = \frac{2\log n + \log \delta^{-1}}{\theta}$. p_N is a parameter that determines the sampling probabilities; see Lemma 27 for the exact role this parameter plays. The algorithm works as follows. We make $\lg n$ vertex samples (line 2) which we call S_k's. We will then have a priority queue for each S_k called $\mathcal{Q}_k$ which allows us to iterate over $N(S_k)$ in the order of increasing value of $h(v)$. This could be done by inserting all vertices in $N(S_k)$ into the priority queue. We, however, use a more efficient way based on the hash-ordered neighbor access. We start with a priority queue that has for each vertex v from S_k its first neighbor (as that is the one with the lowest hash value) represented as $(v, 1)$ with the priority equal to its hash. Whenever we use the pop operation, popping the i-th neighbor of $v \in S_k$ represented (v, i), we insert into $\mathcal{Q}_k$ the next neighbor of v, represented as $(v, i+1)$ with priority equal to its hash. This allows us to access $N(S_k)$ in the order of $h(v)$.

To be able to perform multiple independent runs of the biased vertex sampling algorithm, we will have to resample the hash value from an appropriate distribution for any vertex that the algorithm processes. We will call these resampled hash values virtual and will denote them $h'(v)$. At the beginning,

$h'(v) = h(v)$ for all vertices v. When a vertex v from $N(S_k)$ is processed, we put it, represented as (v, "*virtual*"), back into the priority queue $\mathcal{Q}_k$ with priority equal to the virtual hash. Each priority queue $\mathcal{Q}_k$ therefore contains vertices of the form (v, i), representing $v[i]$, which have not been used by the algorithm yet and vertices of the form (v, "*virtual*") that have been processed already but they have been re-inserted into the priority queue with a new priority $h'(v)$.

Note that that a vertex may be present in the priority queues multiple times (for example, if v is the i-th neighbor of u and j-th neighbour of w, then v may be once inserted as (u, i) and once as (w, j) and once as (v, "*virtual*")). We make sure that all copies of one vertex always have the same priority, namely $h'(v)$. We assume in the algorithm that the hash values of all vertices are different (this happens with probability 1). One may also use $(h'(v), id(v))$ as the priority of the vertex v (with comparisons performed lexicographically) to make the algorithm also work on the event when two vertices have the same hash. This may be useful if implementing the algorithm in practice.

In the pre-processing phase, we initialize the priority queues $\mathcal{Q}_1, \cdots, \mathcal{Q}_{\lg n}$. These will be then used in subsequent calls of the biased vertex sampling algorithm. As we mentioned, we do not make it explicit how they are passed around (they can be thus though of as global variables).

In what follows, we let $v(a) = w[i]$ for for $a = (w, i)$ and $v(a) = w$ for $a = (w$, "*virtual*"). We assume that the variables T_k are sets (as opposed to multisets).

Algorithm 11: Biased vertex sampling – preprocessing algorithm, given a tradeoff parameter θ

```
1 for $k \in \{0, \cdots, \lg n\}$ do
2     $S_k \leftarrow$ sample $np_V/2^k = \frac{n\log(2n/\delta)}{2^k\theta}$ vertices with replacement
3     If $d(S_k) \geq \frac{4m\log(2n/\delta)}{2^k\theta}$, go back to line 2
4     for $v \in S_k$ do
5         Insert into $\mathcal{Q}_k$ the tuple $(v, 1)$ with priority $h'(v[1])$
```

Remark. In practice, implementing Algorithm 12 poses the problem that the virtual hash values $h'(v)$ may be quickly converging to 1 as ℓ increases, making it necessary to use many bits to store them. This can be circumvented as follows. Whenever $1-(1-p_N 2^k)^\ell \geq 1/2$ during the execution of the algorithm, we access the whole neighborhood of the vertex, allowing us to resample the hashes (by setting virtual hashes) from scratch. In other words, when we have seen in expectation half of the vertices adjacent to a vertex, we get all of them, allowing us to resample them from the same distribution they had at the beginning. (Using this approach would require a minor modification to Algorithm 12.)

Algorithm 12: Biased vertex sampling – sampling algorithm, given parameters θ, p_N

```
Let $\ell$ be such that this is the $\ell$-th execution of the algorithm.
for $k \in \{0, \cdots, \log n\}$ do
    $T_k \leftarrow \emptyset$
    while either $h'(v(a)) \leq 1-(1-p_N 2^k)^\ell$ or $p_N 2^k \geq 1$ for $a \leftarrow \mathcal{Q}_k.top()$ do
        while $v(b) = v(a)$ for $b \leftarrow \mathcal{Q}_k.pop()$ do
            if $b$ is of the form $(v, i)$ then
                Replace $(v, i)$ by $(v, i+1)$ in $\mathcal{Q}_k$, set priority to $h'(v[i+1])$
        if $d(v(a)) \notin [2^k\theta, 2^{k+1}\theta)$ then
            Skip $a$, continue with next iteration of the loop
        $T_k \leftarrow T_k \cup \{v(a)\}$
        $h'(v(a)) \leftarrow Unif([1-(1-p_N 2^k)^\ell, 1])$ or 1 if $1-(1-p_N 2^k)^\ell > 1$
        Add to $\mathcal{Q}_k$ an item $(v(a),$ "virtual"$)$ with priority $h'(v(a))$
return $T_0 \cup \cdots \cup T_{\lg n}$
```

In what follows, let $h'_\ell(v)$ be the virtual hash at the beginning of the ℓ-th execution of Algorithm 12 and let $k_v = \lfloor \lg(d(v)/\theta) \rfloor$.

Lemma 26. *Condition on each heavy vertex v having a neighbor in S_{k_v}. At the beginning of the ℓ-th execution of Algorithm 12, for all heavy v with $p_N 2^{k_v} < 1$, $h'(v)$ is distributed uniformly on $[1-(1-p_N 2^{k_v})^{\ell-1}, 1]$ and the events $h'_\ell(v) \leq 1-(1-p_N 2^{k_v})^\ell$ for $v \in V$ and $\ell \in \mathbb{Z}^+$ are jointly independent. Moreover, any a such that $v(a) = u$ in $\mathcal{Q}_k$ has priority $h'(u)$.*

Proof. We first focus on the distribution of the virtual hashes; we prove that the priorities are equal to the hashes afterwards. The proof of both parts is by induction on ℓ.

We define $X_{v,\ell}$ to be the indicator for $h'_\ell(v) \leq 1-(1-p_N 2^{k_v})^\ell$. We now focus on one vertex v and drop it in the subscript. We prove by induction that, conditioned on $X_1, \cdots X_{\ell-1}$, the distribution of $h_\ell(v)$ is uniform on $1-(1-p_N 2^{k_v})^{\ell-1}$. This implies that the (unconditional) distribution of $h'_\ell(v)$ is as claimed. We will also use this below to prove independence. The distribution of $h_1(v)$ is as claimed as the virtual hash values are initially equal to the original (non-virtual) hash values which are assumed to be uniformly distributed on $[0, 1]$ and we are conditioning on an empty set of random variables. Consider $h_{\ell-1}(v)$ conditioned on $X_1, \cdots, X_{\ell-2}$. The distribution is uniform on $[1-(1-p_N 2^{k_v})^{\ell-2}, 1]$ by the inductive hypothesis. Conditioning on $X_{\ell-1} = 1$, we resample $h'(v)$ uniformly from $[1-(1-p_N 2^{k_v})^{\ell-1}, 1]$. Conditioning on $X_{\ell-1}=0$, is equivalent to conditioning on $h'(v) > 1-(1-p_N 2^{k_v})^{\ell-1}$. This conditional distribution is uniform on $[1-(1-p_N 2^{k_v})^{\ell-1}, 1]$. Either way, the distribution is as claimed. This proves the inductive step.

We now argue independence across vertices. The algorithm has the property that the virtual hash value of one vertex does not affect the values of other vertices (note that when processing vertex v, all conditions in the algorithm only depend on $h'(v)$ and independent randomness). This implies that $X_{u,1}, X_{u,2}, \cdots$ and $X_{v,1}, X_{v,2}, \cdots$ are independent for $u \neq v$. This together with what we have shown above implies joint independence. To prove this formally, consider some finite subset $S \subseteq V \times \mathbb{Z}^+$ and let us have $x_a \in \{0,1\}$ for each $a \in S$. Let $V(S) = \{v | \exists \ell \in \mathbb{Z}^+, (v,\ell) \in S\}$ and let $\prec$ be arbitrary total ordering on $V(S)$. Let $\mathbb{Z}_v(S) = \{\ell | (v,\ell) \in S\}$. Consider now $P(\bigwedge_{a\in S} X_a = x_a)$. We can re-write this as

$$\prod_{u\in V(S)} P(\bigwedge_{\substack{(v,\ell)\in S\\ v=u}} X_{v,\ell} = x_{v,\ell} | \{X_{w,\ell'}\}_{(w,\ell')\in S, w\prec u}) = \prod_{u\in V(S)} P(\bigwedge_{\substack{(v,\ell)\in S\\ v=u}} X_{v,\ell} = x_{v,\ell}) \tag{3.14}$$

where the equality holds by the independence of $X_{u,1}, X_{u,2}, \cdots$ and $X_{v,1}, X_{v,2}, \cdots$. We can further rewrite

$$P(\bigwedge_{\substack{(v,\ell)\in S\\ v=u}} X_{v,\ell} = x_{v,\ell}) = \prod_{\ell\in\mathbb{Z}_v(S)} P(X_{u,\ell} = x_{u,\ell} | \{X_{u,\ell'}\}_{\ell'\in\mathbb{Z}_u, \ell'<\ell}) = \prod_{\ell\in\mathbb{Z}_v(S)} P(X_{u,\ell} = x_{u,\ell}) \tag{3.15}$$

where the second equality holds because (as we have shown above) the virtual hash $h'_\ell(v)$ is independent of $X_{v,1}, X_{v,2}, \cdots, X_{v,\ell-1}$. Putting this together, we have

$$P(\bigwedge_{a\in S} X_a = x_a) = \prod_{a\in S} P(X_a = x_a)$$

which means that the random variables $X_{v,\ell}$ (and thus the events $h_\ell(v) \leq 1-(1-p_N 2^{k_v})^\ell$) are jointly independent.

We now argue that the priorities are equal to the virtual hashes. Whenever a vertex is added to $\mathcal{Q}_k$, its priority is equal to its virtual hash. The only way it may happen that the priorities and virtual hashes are not equal is that the virtual hash of some vertex u changes while there exists $a \in \mathcal{Q}_k$ such that $v(a) = u$. This never happens as the virtual hash of $v(a)$ changes only on line 11 after all $a \in \mathcal{Q}_k$ such that $v(a) = u$ have been removed. Note that we are using the fact that different vertices have different virtual hashes (which we assume without loss of generality as we discussed above), which ensures that all a with $v(a) = u$ are removed on line 5. □

Lemma 27. *Let us have integer parameters $\theta, t \geq 1$. When executed t times, Algorithm 12 returns samples $T_1, \cdots, T_t$. Assume Algorithm 12 is given the priority queues $\{\mathcal{Q}_k\}_{k=1}^{\lg n}$ produced by Algorithm 11. Then there is an event $\mathcal{E}$ with probability at least $1-\delta$ such that, conditioning on this event, for any $i \in [t]$ and any vertex v such that $d(v) \geq \theta$, it holds that*

$$P(v \in T_i) = \min(1, p_N 2^{\lfloor \lg \frac{d(v)}{\theta} \rfloor}) \in [\min(1, \frac{d(v)}{2\theta} p_N), \min(1, \frac{d(v)}{\theta} p_N)]$$

and, conditionally on $\mathcal{E}$, the events $\{e \in T_i\}_{e \in E, i \in [t]}$ are jointly independent. Algorithm 11 has expected time complexity $O(\frac{n \log n \log(n/\delta)}{\theta})$. Algorithm 12 has expected total time complexity $O\big(t\big(1 + \frac{p_N m \log^2 n \log(n/\delta)}{\theta}\big)\big)$.

Proof. We start by specifying the event $\mathcal{E}$ and bounding its probability. The probability that a vertex of degree at least $2^k\theta$ does not have a neighbor in S_k after one sampling of S_k (that is, not considering the repetitions) on line 2is at most

$$(1 - \frac{2^k\theta}{n})^{\frac{n \log(2n/\delta)}{2^k\theta}} \leq \exp\big(-\log(2n/\delta)\big) = \frac{\delta}{2n}$$

Therefore, taking the union bound over all vertices, the probability that there exists an integer k and a vertex with degree at between $2^k\theta$ and $2^{k+1}\theta$ that does not have at least one neighbour in S_k after sampling S_k on line 2, is at most $\delta/2$. We now bound the probability that this holds for some S_k on line line 4.

There are $np_V/2^k = \frac{n \log(2n/\delta)}{2^k\theta}$ vertices sampled on line 2 of Algorithm 11. The expected size of the neighborhood of a vertex picked uniformly at random is $\frac{2m}{n}$. The expectation of $d(S)$ is then $\frac{2m \log(2n/\delta)}{2^k\theta}$. By the Markov's inequality, the probability that $d(S) \geq \frac{4m \log(2n/\delta)}{2^k\theta}$ is at most 1/2. We repeatedly sample S_k until it satisfies $d(S) \leq \frac{4m \log(2n/\delta)}{2^k\theta}$. Its distribution is thus the same as if we conditioned on this being the case. It can be easily checked (by the Bayes theorem) that conditioned on the event $d(S) < \frac{4n \log(2n/\delta)}{2^k\theta}$, any vertex with degree between $2^k\theta$ and $2^{k+1}\theta$ has at least one neighbour in S_k, with probability at least $1-\delta$. Therefore, on line 4, it holds that with probability at least $1-\delta$, there does not exist an integer k and a vertex v such that $2^k \leq d(v) \leq 2^{k+1}$ and v has no neighbor in S_k. We call $\mathcal{E}$ the event that this is the case. In the rest of this proof, we condition on $\mathcal{E}$.

We now prove correctness. Consider a vertex v and let again $k = \lfloor \lg(d(v)/\theta) \rfloor$. Consider the case $p_N 2^k \geq 1$. Conditioned on $\mathcal{E}$, one of its neighbours is in S_k. Since $p_N 2^k \geq 1$, the condition on line 4 is satisfied in the k-th iteration of the loop on line 2. Therefore, the whole neighborhood of S_k is added to T_k and the returned sample thus contains v.

We now consider the case $p_N 2^k < 1$. Conditioned on $\mathcal{E}$, a vertex v is included in T_k when $2^k\theta \leq d(v) < 2^{k+1}\theta$ and $h'(v) \leq 1 - (1 - p_N 2^k)^\ell$. Since

$h'(v)$ is uniformly distributed in $[1-(1-p_N 2^k)^{\ell-1}, 1]$, this happens with probability

$$\frac{1-(1-p_N 2^k)^{\ell}-(1-(1-p_N 2^k)^{\ell-1})}{1-(1-(1-p_N 2^k)^{\ell-1})} = p_N 2^k$$

Let $h'_\ell(v)$ and k_v be defined as in the statement of Lemma 26. We know from that lemma that the events $\{h'_\ell(v) \leq 1-(1-p_N 2^{k_v})^{\ell}\}_{v \in V, \ell \in [t]}$ are jointly independent, conditioned on $\mathcal{E}$. $h'_\ell(v) \leq 1-(1-p_N 2^{k_v})^{\ell}$ is equivalent to $v \in T_\ell$. This implies that the events $\{v \in T_\ell\}_{v \in V, \ell \in [t]}$ are also jointly independent, as we set out to prove.

We now prove the claimed query complexity. We first show the complexity of Algorithm 11. As we argued, the probability of resampling S_k because the condition on line 3 is satisfied, is at most $1/2$. Therefore, the time spent sampling the set S_k (including the repetitions) is $O(|S_k|)$. For every k, $|S_k| = \frac{n \log(2n/\delta)}{2^k \theta}$. Therefore, the total size of the sets S_k is upper-bounded by

$$\sum_{k=0}^{\infty} \frac{n \log(2n/\delta)}{2^k \theta} = \frac{2n \log(2n/\delta)}{\theta}$$

Since all values that are to be inserted into $\mathcal{Q}_k$ in Algorithm 11 are known in advance, we can build the priority queues in time linear with their size. This means that the preprocessing phase (Algorithm 11) takes $O(\frac{n \log(n/\delta)}{\theta})$ time.

We now focus on Algorithm 12. We now prove that an execution of the loop on line 2 of Algorithm 12 takes in expectation $O(1 + \frac{p_N m \log n \log(n/\delta)}{\theta})$ time for any $k \in \{0, \cdots, \log n\}$, from which the desired bound follows. Specifically, we prove that the number of executions of lines 6-8 is $O(\frac{p_N m \log(n/\delta)}{\theta})$ from which this bound follows as every iteration takes $O(\log n)$ time (as the time complexity of an iteration is dominated by the operations of the priority queue).

Lines 6-8 are executed once for each item a in $\mathcal{Q}_k$ such that the priority of a is $\leq 1-(1-p_N 2^k)^{\ell}$ if $p_N 2^k < 1$ or when $p_N 2^k \geq 1$. As we have argued, this happens with probability $\min(1, p_N 2^k) \leq p_N 2^k$. There are $\frac{n \log(2n/\delta)}{2^k \theta}$ vertices in S_k. Since these vertices are chosen at random, there is in expectation $\frac{2m \log(2n/\delta)}{2^k \theta}$ incident edges. We consider an item for such incident edge with probability $\leq p_N 2^k$. This means that we consider on the mentioned lines in expectation $O(\frac{p_N m \log(2n/\delta)}{\theta})$ edges, as we wanted to prove. □

Bernoulli sampling with hash-ordered neighbor access

We now show how to sample each edge independently with some fixed probability p in the hash-ordered neighbor access model. Our approach works by separately sampling light and heavy edges, then taking union of the samples.

In fact, we solve a more general problem of making t Bernoulli samples with time complexity *sublinear* in t for some range of parameters. We will need this for sampling edges with replacement. We first give an algorithm to sample edges, assuming we can sample separately light and heavy edges.

Algorithm 13: Make t Bernoulli samples from E with inclusion probability p

```
1 if $p > 0.9$ then
2     Perform the sampling by a standard linear-time algorithm in time
        $O(n + tpm)$.
3 $\theta \leftarrow \sqrt{\frac{\log(n)\log(n/\delta)}{pt}}$
4 Run Algorithm 11 with parameter $\theta$ to prepare data structures for
    Algorithm 12 (used within Algorithm 15)
5 for $i$ from 1 to $t$ do
6     $S_{L,i} \leftarrow$ sample each light edge with probability $p$ using
        Algorithm 14 with parameter $\theta$
7     $S_{H,i} \leftarrow$ sample each heavy edge with probability $p$ using
        Algorithm 15 with parameter $\theta$
8 return $(S_{L,1} \cup S_{H,1}, \cdots, S_{L,t} \cup S_{H,t})$
```

Theorem 28. *Given a parameter t, with probability at least $1-\delta$, Algorithm 13 returns t samples $T_1, \cdots, T_t$. Each edge is included in T_i with probability p. Furthemore, the events $\{e \in T_i\}_{e\in E, i\in[t]}$ are jointly independent. The expected time complexity is $O\big(\sqrt{tpn}\sqrt{\log(n)\log(n/\delta)}+t\big(1+pm\log^2 n\log(n/\delta)\big)\big)$.*

Proof. If $p > 0.9$, we read the entire graph in $O(n+m)$ and compute the sample in time $O(tm) = O(tpm)$. This is less than the claimed complexity. In the rest, we assume that $p \leq 0.9$.

Correctness follows from lemmas 29 and 30 which imply that light and heavy edges, respectively, are separately sampled independently with the right probability. Moreover, there is no dependency between the samples of the light and heavy edges, as the sample of light edges does not depend on the hashes. The same lemmas give running times of $O(tpn\theta)$ and $O(tpm\log^2 n\log n/\delta)$ spent on lines 6 and 7 respectively. Algorithm 11 takes $\frac{n\log n\log(n/\delta)}{\theta}$ time. Substituting for θ, the expected time complexity is as claimed. □

Sampling light edges

Now we show how to sample from the set of light edges such that each light edge is sampled independently with some specified probability p.

As essentially the same algorithm already appeared in Eden and Rosenbaum [2018a], we defer the proof of the following lemma to the full version.

Algorithm 14: Sample each light edge independently with probability p

```
1  $k \sim Bin(n\theta, p)$
2  $T \leftarrow \emptyset$
3  $M \leftarrow \emptyset$
4  repeat $k$ times
5      $v \leftarrow$ pick vertex uniformly at random
6      $\ell \leftarrow$ random number from $[\theta]$
7      if $(v, \ell) \in M$ then
8          Go to line 5.
9      If $v$ is light and $d(v) \geq \ell$, add the $\ell$-th edge incident to $v$ to $T$
10     $M \leftarrow M \cup \{(v, \ell)\}$
11 return $T$
```

Lemma 29. *Algorithm 14 samples each light edge independently with probability $p \leq 0.9$ and its expected query complexity is $O(pn\theta)$.*

Proof. Consider the set B of pairs (v, ℓ) where $v \in V$ and $\ell \in [\theta]$. We sample $k \sim Bin(n\theta, p)$ such pairs without replacement. By the choice of k and the conditioning principle, it holds that each pair (v, ℓ) has been sampled with probability p, independently of other pairs. Each light edge has exactly one corresponding pair in B, namely a light edge uv where v is the i-th neighbor of u corresponds to (u, i). The algorithm returns all light edges whose corresponding pair was sampled. This happened for each pair independently with probability p, thus implying the desired distribution of T.

By the assumption $p \leq 0.9$, we have $P(k \leq 0.95n\theta) \geq 1 - \exp(-\Omega(n\theta))$. On this event, sampling each edge (that is, sampling a pair (v, ℓ) that is not in M) takes in expectation $O(1)$ queries. Moreover $E(k) = pn\theta$. Therefore, it takes in expectation $O(pn\theta)$ queries to sample the k pairs (combining the expectations using the Wald's equation). It always takes $O(n\theta \log(n\theta))$ time to sample k edges due to coupon collector bounds, so the event $k > 0.95n\theta$ only contributes $o(1)$ to the expectation since $P(k > 0.95n)$ is exponentially small. □

Sampling heavy edges

We now show an algorithm for Bernoulli sampling from the set of heavy edges. The algorithm is based on Algorithm 12.

Lemma 30. *Assume that Algorithm 15 is given $\{Q_k\}_{k=1}^{\lg n}$ as set by Algorithm 11. With probability at least $1 - \delta$, Algorithm 15 samples each heavy edge independently with probability p. Moreover, when executed multiple times,*

Algorithm 15: Sample each heavy edge independently with probability p, given parameter θ

```
1 $S \leftarrow$ Use Algorithm 12 with $p_N = \min(1, 2\theta p)$
2 $S' \leftarrow$ heavy vertices from $S$
3 $T \leftarrow \emptyset$
4 for $v \in S'$ do
5     With probability $\frac{1-(1-p)^{d(v)}}{\min(1, p_N 2^{\lfloor \lg \frac{d(v)}{\theta} \rfloor})}$, skip $v$ and continue on line 3
6     $k \sim (Bin(d(v), p) | k \geq 1)$
7     Sample $k$ edges incident to $v$ without replacement, add them to $T$
8 return $T$
```

the outputs are independent. It has time complexity $O(pm \log^2 n \log n/\delta)$ with high probability.

Proof. We first show that the probability on line 4 is between 0 and 1 (otherwise, the algorithm would not be valid). It is clearly non-negative, so it remains to show that $1-(1-p)^{d(v)} \leq \min(1, p_N 2^{\lfloor \lg \frac{d(v)}{2\theta} \rfloor})$. If $1 \leq p_N = 2\theta p$, then $1 \leq p_N 2^{\lfloor \lg \frac{d(v)}{2\theta} \rfloor}$, and the inequality then clearly holds. Otherwise,

$$p_N 2^{\lfloor \lg \frac{d(v)}{2\theta} \rfloor} \geq 2\theta p 2^{\lg(\frac{d(v)}{\theta})-1} \geq pd(v) \geq 1-(1-p)^{d(v)}$$

We now show correctness. That is, we show that edges are sampled independently with the desired probabilities. By Lemma 27, for heavy v, it holds that $P(v \in S | \mathcal{E}) = \min(1, p_N 2^{\lfloor \lg \frac{d(v)}{2\theta} \rfloor})$. This means that the probability of v being in S and not being skipped is

$$\min(1, p_N 2^{\lfloor \lg \frac{d(v)}{2\theta} \rfloor}) \frac{1-(1-p)^{d(v)}}{\min(1, p_N 2^{\lfloor \lg \frac{d(v)}{\theta} \rfloor})} = 1-(1-p)^{d(v)}$$

This is equal to the probability of $X \geq 1$ for $X \sim Bin(d(v), p)$. Since k is picked from $(Bin(d(v), p) | k \geq 1)$, the distribution of number of edges incident to v that the algorithm picks is distributed as $Bin(d(v), p)$. Let X_v be the number of edges incident to v that are picked. Since each vertex v is picked independently by Lemma 27, we have that these random variables are independent and $X_v \sim Bin(d(v), p)$. Consider the vector $(X_{v_1}, \cdots, X_{v_n})$. Consider experiment where we pick each edge independently with probability p (this is the desired distribution) and let Y_v be the number of edges incident to v that are picked. Then $(Y_{v_1}, \cdots, Y_{v_n}) \sim (X_{v_1}, \cdots, X_{v_n})$. By the conditioning principle, we get that each (directed) edge is sampled independently with probability p.

Assume Algorithm 15 is executed t times, outputting sets $H_1, \cdots, H_t$. By Lemma 27, we know that the events $\{e \in H_i\}_{e \in E, i \in [t]}$ are jointly independent. Algorithm 15 only depends on the virtual hashes in the calls of Algorithm 12 and the rest only depends on independent randomness. The samples $H_1, \cdots, H_t$ are thus independent.

We now argue the time complexity. Algorithm 12 has expected time complexity $O(\frac{p_N m \log^2 n \log(n/\delta)}{\theta}) = O(pm \log^2 n \log(n/\delta))$. We now prove this dominates the complexity of the algorithm. The rest of the algorithm has time complexity linear in $|S| + |T|$. The complexity of Algorithm 12 clearly dominates $|S|$ (as S is the output of this algorithm). It holds $E(|T|) = pm$, so the expected size of T is also dominated by the complexity of Algorithm 12.This completes the proof. □

Sampling edges without replacement with hash-ordered neighbor access

We now show how Bernoulli sampling can be used to sample vertices without replacement.

Algorithm 16: Sample without replacement s edges

```
1 $p \leftarrow 1/n^2$
2 $S \leftarrow$ sample each edges with probability $\min(1, p)$ using Algorithm 13
     with failure probability $\frac{\delta}{3 \lg n}$
3 if $|S| < s$ then
4     $p \leftarrow 2p$
5     Repeat from line 1
6 return random subset of $S$ of size $s$
```

Theorem 31. *Algorithm 16 samples s edges uniformly without replacement with probability at least $1 - \delta$. Moreover, Algorithm 16 has expected query complexity $O(\sqrt{s}\frac{n \log(n) \log(n/\delta)}{\sqrt{m}} + s \log^2 n \log(n/\delta))$.*

Proof. In each iteration, Algorithm 13 fails with probability at most $\frac{\delta}{3 \lg n}$. There are at most $\lg n^2$ iterations in which $p < 1$. After this number of iterations, each additional iteration can happen only if Algorithm 13 fails. This happens with probability $< 1/2$, so we get in expectation < 2 additional iterations. Putting this together by the Wald's equation[13], we see that the

[13] We use the Wald's equation on the indicators that in the i-th iteration fails. The bound is, in fact, on the expected number of failures, which is an upper bound on the probability of failure (by the Markov's inequality).

failure probability is at most $(2\lg+2)\frac{\delta}{3\lg n} < \delta$. In the rest of the analysis, we condition on no errors happening in any of the calls of Algorithm 13.

Condition on the ℓ-th iteration being the first to succeed. What is the conditional distribution of the sample? Note that repeating an experiment until the outcome satisfies some property ϕ results in the outcome of the experiment being distributed as if we conditioned on ϕ. Consider additionally conditioning on $|S| = k$. Then, by symmetry, S is a sample without replacement of size $|k|$. If $k \geq s$, it holds that taking a sample without replacement of size k and taking a random subset of size s, we get a sample with distribution of a sample without replacement of size s. Since this distribution is the same for all values of $k \geq s$, the distribution is unchanged if we only condition on the union of the events $|S| = k$ for $k \geq s$ or, in other words, if we condition on $|S| \geq s$. Similarly, the distribution is the same independently of the value of ℓ. Therefore, the unconditioned distribution is the same. This proves that the algorithm gives a sample from the right distribution. .

Since the probability p increases exponentially, so does the expected time complexity of each iteration. Therefore, the time complexity is dominated by the complexity of the last iteration. Consider an iteration with $p < \frac{2s}{m}$. Then the time complexity of this iteration is no greater than the desired bound. Consider now the case $p \geq \frac{2s}{m}$. By the Chernoff bound, the probability that $|S| < s$ is then at most $1/3$. The probability of performing each additional iteration thus decreases exponentially with base $1/3$. The expected time complexity of an iteration, on the other hand, increases exponentially with base 2. This means that the expected time complexity contributed by each additional iteration decreases exponentially. The asymptotic time complexity is therefore equal to that of the iteration with $\frac{s}{m} \leq p < \frac{2s}{m}$. The complexity of an iteration is dominated by line 2. Therefore, by Theorem 28, the time complexity is as claimed. □

Sampling edges with replacement with hash-ordered neighbor access

The algorithm for Bernoulli sampling can be used to sample multiple edges with replacement. The proof of the theorem below appears in the full version. We now sketch intuition of correctness of the algorithm. Suppose S_1 is nonempty. Picking from each non-empty S_i one edge at random gives us a sample without replacement of size equal to the number of non-empty S_i's. Since the number of S_i's is $2s$, if we pick p large enough, then with high probability at least s of them will be non-empty. In that case, we have a sample without replacement of size $\geq s$ and taking a random subset of size s gives a sample with the desired distribution.

Theorem 32. *Given $s \leq n$, with probability at least $1-\delta$, Algorithm 17 returns s edges sampled with replacement. Moreover, the algorithm runs in*

```
Algorithm 17: Sample $s$ edges with replacement
1 $p \leftarrow 1/n^2$
2 $S_1, \cdots, S_{2s} \leftarrow$ Bernoulli samples with $\min(1, p)$ using Algorithm 13
    with failure probability $\frac{\delta}{3 \lg n}$
3 $T \leftarrow$ from each non-empty $S_i$ pick a random edge
4 if $|T| < s$ then
5     $p \leftarrow 2p$
6     Go to line 2
7 return random subset of $T$ of size $s$
```

time $O(\sqrt{s}\frac{n}{\sqrt{m}} \log n \log n/\delta + s \log^2 n \log(n/\delta))$ *with high probability.*

Proof. By exactly the same argument as in Theorem 31, with probability at least $1 - \delta$, there are no errors in the calls of Algorithm 13. In the rest of the proof, we condition on this being the case.

We now argue that the returned sample has the correct distribution. The argument is again very similar to that in Theorem 31. Consider one of the S_i's. If we condition on $|S_i| = k$, then by symmetry, the distribution is that of sampling k edges without replacement. Picking at random one of those edges (assuming $k \geq 1$), we get one edge uniformly at random. Since this distribution is independent of k for $k \geq 1$, this is also the distribution we get if we only condition on $k \geq 1$. Therefore, T has a distribution of $|T|$ edges picked uniformly at random with replacement at random, where $|T|$ is a random variable. The effect on the distribution of T of repeating the sampling until $|T| \geq s$ is the same as conditioning on $|T| \geq s$. Taking a random subset of size s, it has a distribution of s edges being sampled with replacement.

The time complexity of the algorithm is dominated by line 2. The expected time complexity of each iteration increases exponentially with base 2 as this is the rate at which p increases. The probability that $S_i = \emptyset$ is $(1-p)^m \leq e^{-pm}$. Consider the case $p \geq 2/m$. Then the probability that $|T| < s$ can be upper-bounded by the Chernoff inequality by $1/3$. Therefore, after $p \geq 2/m$, the probability of each additional iteration decreases exponentially with base 3. Therefore, the expected time complexity contributed by each additional iteration then decreases exponentially. The expected time complexity is thus dominated by the first iteration in which $p \geq 2/m$. The expected complexity is thus as claimed by Theorem 28. □

Implementing our algorithms with batched access

Let d denote the average degree rounded up and consider the following setting. Suppose we have access to the following queries: (1) random vertex query and (2) query that, given a vertex v and an index i, returns neighbors $(i -$

$1)d+1,\cdots,i\,d$ of v (where the neighborhoods are assumed to be ordered arbitrarily). As we argued in Chapter 3, this setting is relevant in many practical situations, such as when accessing a graph through a (commonly used) API interface or when the graph is stored on a hard drive.

Consider sampling a vertex uniformly and then querying it to learn its whole neighborhood. In expectation, this takes ≤ 2 queries[14] as the expected neighborhood size is d and we may get $\geq d$ neighbors in one query. Algorithm 14, has the property that it only accesses the neighborhoods of uniformly random vertices. In Algorithm 15, we access neighborhoods on line 7 and within calls of Algorithms 11 and 12 which also have this property. On line 7, we only access in expectation $O(pm)$ neighbors, adding a cost of at most $O(pm)$ queries, thus not increasing the complexity. For the other vertices whose neighborhoods we will be accessing, we may learn their entire neighborhood and simulate the hash-ordered neighbor access at a cost of $O(1)$ queries per vertex. This allows us to implement Algorithms 13 and 16 to 19 in this model without increasing their asymptotic query complexity.

Sampling multiple edges without hash-ordered neighbor access

We now show an algorithm that does not use the hash function, at the cost of a slightly worse running time. The only place where we have used the hash function in the above algorithms is when sampling heavy edges, specifically in Algorithm 12. We show how to simulate Algorithm 12 in the indexed neighbor access model. We can then use this to get Bernoulli sampling as well as sampling with and without replacement. We actually show how to simulate any algorithm from the hash-ordered neighbor access model in the indexed neighbor access model. We then analyze the running time of the simulation of Algorithm 12. Our algorithm improves upon the state of the art when $\varepsilon = \tilde{o}(\sqrt{\frac{n}{m}})$.

Theorem 33. *There are algorithms that, with probability at least $1-\delta$, return sample (1) of edges such that each such edge is sampled independently with probability $\frac{s}{m}$ (where s does not have to be an integer), (2) of s edges sampled without replacement, or (3) of s edges sampled with replacement. Assuming $m = \Omega(n)$, these algorithms run in expected time $O(\sqrt{sn}\log n/\delta + s\log^2 n \log(n/\delta))$ with high probability.*

Proof. We show a general way to simulate hash-ordered neighbor access. The first time the algorithm wants to use a neighborhood query on some vertex v, we look at the whole neighborhood of v, generate virtual hashes $h'(u)$ for all $u \in N(v)$ for which it has not been generated yet, and sort $N(v)$ with respect to the virtual hash values. This clearly allows us to run any algorithm from the hash-ordered neighbor access model in the indexed neighbor access model.

[14] It is 2 and not 1 due to rounding. For example, if $O(1)$ vertices have degree 0 and the rest have degree just above d, then ≈ 2 queries will be needed on average.

In Algorithm 12, we access neighborhoods of the vertices sampled in Algorithm 11, of total size at most $O(\frac{m \log n}{\theta})$. The time complexity is, therefore, by the same argument as in lemmas 27, 29 and 30, at most $O(\frac{m \log n/\delta}{\theta} + sn\theta/m + s\log^2 n \log(n/\delta))$. We now set $\theta = \text{median}(1, \tilde{m}\sqrt{\frac{\log(n/\delta)}{ns}}, n\sqrt{\log(n/\delta)})$.

We may use, for example, the algorithm from Goldreich and Ron [2006] to get $\tilde{m}$ such that it holds $m \le \tilde{m} \le 2m$ with probability at least $1 - \frac{1}{sn^2}$. We call this event $\mathcal{E}$. Conditioning on $\mathcal{E}$, $\theta = \Theta(m\sqrt{\frac{\log(n/\delta)}{ns}})$ and the complexity is as claimed. The time complexity is always $O(m \log n/\delta + sn^2 \log(n/\delta)/m + s\log^2 n \log(n/\delta))$ as $1 \le \theta \le n\sqrt{\log(n/\delta)}$. It holds $P(\mathcal{E}^C) \le \frac{1}{sn^2}$. Therefore, this event contributes only to the expected time complexity only $O(\log^2 n \log(n/\delta)$. Thus, the event $\mathcal{E}^C$ does not increase the asymptotic time complexity. □

Lower bound for sampling multiple edges

As the last result on sampling edges, we prove that algorithms 13, 16 and 17 are optimal up to logarithmic factors.

Theorem 34. *Any algorithm in the hash-ordered neighbor access model that samples pointwise* 0.9*-close to uniform (1) each edge independently with probability* $\frac{s}{m}$*, (2)* s *edges without replacement, or (3)* s *edges with replacement, has to use in expectation* $\Omega(\sqrt{s}\frac{n}{\sqrt{m}} + s)$ *queries.*

Proof. The term s is dominant (up to a constant factor) for $s \gtrsim \frac{n^2}{m}$ and the lower bound holds on this interval as any algorithm that returns s edges has to run in time $\Omega(s)$. Now we consider the case when $s \lesssim \frac{n^2}{m}$.

Let G be a graph consisting of s cliques, each having m/s edges, and the remaining vertices forming an independent set. Due to the assumption on s, the total number of vertices used by the cliques is no more than n and this graph, therefore, exists.

Consider the case of sampling s edges pointwise 0.9-close to uniform at random in either of the three settings. They hit in expectation $\Omega(s)$ distinct cliques. Since the algorithm has to hit each clique from which an edge is sampled, it has to hit in expectation $\Omega(s)$ cliques by uniformly sampling vertices. The probability that a uniformly picked vertex lies in one fixed clique is $O(\frac{s\sqrt{m/s}}{n}) = O(\frac{\sqrt{sm}}{n})$. To hit in expectation s cliques, the number of samples the algorithm has to perform is then $\Omega(s\frac{n}{\sqrt{sm}}) = \Omega(\sqrt{s}\frac{n}{\sqrt{m}})$. □

Estimating the Number of Edges by Sampling

The Bernoulli sampling from Line 13 allows us to estimate the number of edges efficiently. The idea is that if we sample each edge independently with probability p, then the number of sampled edges is concentrated around pm, from which we can estimate m (assuming we know p).

Algorithm 18: Estimate the number of edges by edge sampling

```
1 $p \leftarrow 1/n^2$
2 $S \leftarrow$ sample each edge with probability $p$ using Algorithm 13
3 if $|S| < \frac{6(\log \delta^{-1} + \log 8 \lg n)}{\varepsilon^2}$ then
4     $p \leftarrow 2p$
5     Go to line 2
6 return $|S|/p$
```

Lemma 35. *Given $\varepsilon < 1$, Algorithm 18 returns an estimate $\hat{m}$ of m such that $P(|\hat{m} - m| > \varepsilon m) < \delta$. It runs in time $O(\frac{n}{\varepsilon\sqrt{m}} \log n \log n/\delta + \frac{\log n \log n/\delta}{\varepsilon^2})$.*

Proof. The proof works as follows. We first show that if $p < \frac{3(\log \delta^{-1} + \log 8 \lg n)}{\varepsilon^2 m}$, then with sufficiently high probability $|S| < \frac{6(\log \delta^{-1} + \log 8 \lg n)}{\varepsilon^2 m}$ and the algorithm will continue. We then show that if $p \geq \frac{3(\log \delta^{-1} + \log 8 \lg n)}{\varepsilon^2 m}$, then $|S|$ is sufficiently concentrated. Taking the union bound over all iterations, we get that with good probability, in all iterations with $p < \frac{3(\log \delta^{-1} + \log 8 \lg n)}{\varepsilon^2 m}$, it holds that the algorithm continues and for all $p \geq \frac{3(\log \delta^{-1} + \log 8 \lg n)}{\varepsilon^2 m}$, the estimate is ε-close to the true number of edges. This implies correctness.

Consider the case $p < \frac{3(\log \delta^{-1} + \log 8 \lg n)}{\varepsilon^2 m}$. By the Chernoff bound,

$$P(|S| \geq \frac{3(\log \delta^{-1} + \log 8 \lg n)}{\varepsilon^2 m}) \leq \exp(-\frac{pm}{3}) < \frac{\delta}{2 \lg n^2}$$

Consider now the case $p \geq \frac{3(\log \delta^{-1} + \log 8 \lg n)}{\varepsilon^2 m}$ by the Chernoff bound,

$$P(||S| - E(S)| > \varepsilon E(|S|)) < 2 \exp(-\frac{\varepsilon^2 pm}{3}) \leq \frac{\delta}{2 \lg n^2}$$

□

Using theorems 28 and 33 to sample the edges for X, we get the following theorem.

Theorem 36. *There is an algorithm that uses hash-ordered neighbor access and returns a $1 + \varepsilon$-approximate of the number of edges with probability at least $1 - \delta$ in expected time $O(\frac{n}{\varepsilon\sqrt{m}} \log n \log n/\delta + \frac{\log n \log n/\delta}{\varepsilon^2})$.*

There is an algorithm that does not use *hash-ordered neighbor access and returns a $1 + \varepsilon$-approximate of the number of edges with probability at least $1 - \delta$ in expected time $O(\frac{\sqrt{n}}{\varepsilon} \log n \log n/\delta)$.*

In the algorithm without hash-ordered neighbor access, the second term does not have to be present, as it only becomes dominant when the expected time complexity is $\Omega(n)$, in which case we may use a trivial $O(n)$ algorithm.

Directly Estimating the Number of Edges

We now give two algorithms for approximate edge counting. The first uses hash-ordered neighbor access and runs in time $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}} + \frac{1}{\varepsilon^2})$. This is the same complexity as that of Algorithm 18 which approximates the number of edges by Bernoulli sampling. The algorithm we give now is more straightforward and solves the problem of approximating the number of edges directly. We then give a different algorithm which replaces the need for hash-ordered neighbor access by the more standard pair queries. It has time complexity of $\tilde{O}(\frac{n}{\varepsilon\sqrt{m}} + \frac{1}{\varepsilon^4})$. We then show a lower bound of $\Omega(\frac{n}{\varepsilon\sqrt{m}})$ for $\varepsilon \geq \frac{\sqrt{m}}{n}$. This matches, up to logarithmic factors, the complexity of our algorithms for $\varepsilon \geq \frac{\sqrt{m}}{n}$ and $\varepsilon \geq \frac{m^{1/6}}{n^{1/3}}$, respectively.

Algorithm with hash-ordered neighbor access

We combine our biased sampling procedure with the Horvitz-Thompson estimator. When we appropriately set the parameters of biased sampling, we get an estimator with lower variance than the estimator on which the algorithm in Seshadhri [2015] is built.

It is also possible to simulate this algorithm without hash-ordered neighbor access. This algorithm has the same time complexity as the one of the algorithm from Theorem 36. The simulation can be done by the same approach as in Line 7; we do not repeat the argument.

In our analysis, we assume we have an estimate $\tilde{m}$ such that $m \leq \tilde{m} \leq 2m$. We also prove that even when these inequalities do not hold, the algorithm is unlikely to return an estimate that is more than a constant factor greater than m. As a consequence of this guarantee, the advice of $\tilde{m}$ then can be removed by standard techniques. See Chapter 3 for details, including a sketch of how the advice can be removed.

Theorem 37. *Algorithm 19 returns an estimate $\hat{m}$ such that $P(\hat{m} \geq 8m) \leq 1/3$. It has expected time complexity $O(\frac{n\sqrt{\log n}}{\varepsilon\sqrt{\tilde{m}}} + \frac{\log^2 n}{\varepsilon^2})$. If, moreover, $\tilde{m} \leq m$, then with probability at least $2/3$, it holds $|X - m| \leq \varepsilon m$.*

Proof. We first focus on the time complexity. Line 2 runs in expected time $O(\frac{n \log n}{\theta})$. On line 3, we spend in expectation $O(\frac{n \log n}{\theta}) = O(\frac{n\sqrt{\log n}}{\varepsilon\sqrt{m}} + \frac{\log^2 n}{\varepsilon^2})$ time by Lemma 27 and our choice of p_N. This dominates the complexity of the rest of the algorithm.

Algorithm 19: Estimate the number of edges in a graph

```
1 $\theta = \min(\varepsilon\sqrt{\frac{1}{32}\tilde{m}(\log n + \log 12)}, \frac{1}{64}\varepsilon^2 n/\log n)$
2 $S_L \leftarrow$ sample $k = \frac{2n(\log n + \log 12)}{\theta}$ vertices with replacement, keep those
    with degree $< \theta$
3 $S_H \leftarrow$ sample vertices using Algorithm 12 with parameters $\theta$,
    $p_N = \frac{n}{\tilde{m}\log n}$, $t = 1$, and $\delta = 1/12$
4 Define $P_v = \min(1, p_N 2^{\lfloor \lg \frac{d(v)}{\theta} \rfloor})$
5 return $X = \frac{n}{2k}\sum_{v \in S_L} d(v) + \frac{1}{2}\sum_{v \in S_H} \frac{d(v)}{P_v}$
```

In the rest of the proof, we prove correctness. We break up the estimator X into three estimators X_L, X_M, X_H such that $X_L + X_M + X_H = X$. We first prove separately bounds on each of the three estimators, and then put it together.

We say a vertex is light if $d(v) < \theta$, medium if $\theta \leq d(v) < \frac{2\theta}{p_N}$ and heavy if $d(v) \geq \frac{2\theta}{p_N}$ (note that this definition is different from that in Chapter 3). The threshold between medium and heavy vertices is set such that heavy vertices are sampled with probability 1, whereas the probability that a fixed medium vertex is sampled is strictly less than 1. We call the sets of light, medium, and heavy vertices V_L, V_M, V_H, respectively. We define $X_L = \frac{n}{2k}\sum_{v \in S_L} d(v)$ and $X_\bullet$ for $\bullet \in \{M, H\}$ as $\frac{1}{2}\sum_{v \in S \cap V_\bullet} \frac{d(v)}{P_v}$ (P_v is defined in the algorithm) and $X = X_L + X_M + X_H$.

Light vertices. Let v_i be the i-th vertex sampled on line 1. The expectation of X_L is

$$E(X_L) = E\Big(\frac{n}{2k}\sum_{v \in S_L} d(v)\Big) = \frac{n}{2k}\sum_{i=1}^{k} E\big(\mathbb{I}(d(v_i) \leq \theta) d(v_i)\big) = \frac{n}{2}E\big(\mathbb{I}(d(v_1) \leq \theta) d(v_1)\big) = \frac{1}{2}d(V_L)$$

and the variance can be bounded as[15]

$$\begin{aligned}
Var(X_L) &= Var\Big(\frac{n}{2k}\sum_{i=1}^{k} d(v)\Big) && (3.16)\\
&= \frac{n^2}{4k}Var\big(\mathbb{I}(d(v_1) \leq \theta) d(v_1)\big) && (3.17)\\
&\leq \frac{n^2}{4k}\sup\big(\mathbb{I}(d(v_1) \leq \theta) d(v_1)\big) E\big(\mathbb{I}(d(v_1) \leq \theta) d(v_1)\big) && (3.18)\\
&\leq \frac{n^2 \theta d(V_L)}{4kn} && (3.19)
\end{aligned}$$

[15] We define $\sup(X)$ as the smallest x such that $P(X > x) = 0$.

The first inequality holds because $Var(X) \leq \sup(X)E(X)$ whenever $P(X \geq 0) = 1$. The second holds because $\sup\big(\mathbb{I}(d(v_1) \leq \theta)d(v_1)\big) \leq \theta$ and $E\big(\mathbb{I}(d(v_1) \leq \theta)d(v_1)\big) = d(V_L)/\theta$. Because $\theta \leq \varepsilon\sqrt{\frac{1}{32}\tilde{m}(\log n + \log 12)}$, $k = \frac{2n(\log n + \log 12)}{\theta}$ and $\tilde{m} \leq 2m$, we get the following upper bound:

$$\leq \frac{\theta^2 d(V_L)}{4(\log n + \log 12)} \leq \frac{1}{32}(\varepsilon m)^2 \tag{3.20}$$

where the last inequality holds because $d(V_L) \leq 2m$ and by substituting for the other variables. By the Chebyshev bound, it now holds that

$$P(|X_L - \frac{1}{2}d(V_L)| \geq \varepsilon m/2) \leq \frac{Var(X_L)}{(2\varepsilon m)^2} \leq \frac{1}{8}$$

Medium vertices. For the medium vertices, we consider the conditional expectation and variance, conditioned on $\mathcal{E}$ where $\mathcal{E}$ is the event on which Algorithm 12 succeeds. The expectation is

$$E(X_M|\mathcal{E}) = \frac{1}{2}\sum_{v \in V_M} P_v \frac{d(v)}{P_v} = \frac{1}{2}d(V_M)$$

and the variance is

$$Var(X_M|\mathcal{E}) \leq \frac{1}{4}\sum_{v \in V_M} E\Big((\frac{d(v)}{P_v})^2|\mathcal{E}\Big) \leq \frac{1}{4}\sum_{v \in V_M} P_v(\frac{d(v)}{P_v})^2 \leq \frac{1}{2}\sum_{v \in V_M}\frac{d(v)\theta}{p_N} \leq \frac{m\theta}{p_N} \leq \frac{2m^2\theta\log n^2}{n} \leq \frac{1}{32}(\varepsilon m) \tag{3.21}$$

where the last inequality holds because $\theta \leq \frac{1}{64}\varepsilon^2 n/\log n$ and the one before that because $p_N = \frac{n}{\tilde{m}\log n} \geq \frac{n}{2m\log n}$. By the Chebyshev bound, it now holds that

$$P(|X_M - m_M| \geq \varepsilon m/2|\mathcal{E}) \leq \frac{Var(X_M)}{(\varepsilon m/2)^2} \leq \frac{1}{8}$$

Heavy vertices. Since the heavy vertices are, conditioned on $\mathcal{E}$, sampled with probability 1, it is the case that $P(X_H = m_H|\mathcal{E}) = 1$.

Putting it all together. We first prove $P(\hat{m} \geq 8m) \leq 1/3$. By the Markov's inequality, we have that $P(X_L \geq 8E(X_L)) \leq 1/8$ and $P(X_M + X_H \geq 8E(X_M + X_H|\mathcal{E})|\mathcal{E}) \leq 1/8$. We now have

$$\begin{aligned}
P(\hat{m} \geq 8m) &= P(X_L + X_M + X_H \geq 8(E(X_L) + E(X_M + X_H|\mathcal{E}))) \qquad (3.22)\\
&\leq P(X_L \geq 8E(X_L)) + P(X_M + X_H \geq 8E(X_M + X_H|\mathcal{E})|\mathcal{E}) + P(\mathcal{E}^C) \qquad (3.23)\\
&\leq \frac{1}{8} + \frac{1}{8} + \frac{1}{12} = \frac{1}{3} \qquad (3.24)
\end{aligned}$$

Finally, we prove the concentration. We use the union bound and the bounds on X_L, X_M, X_H that we have proven above.

$$\begin{aligned} P(|X-m| \geq \varepsilon m) \leq & P(|X_L - m_L| \geq \varepsilon m/2) + P(|X_M - m_M| \geq \varepsilon m/2 \vee X_H \neq m_H) & (3.25) \\ \leq & P(|X_L - m_L| \geq \varepsilon m/2) + P(|X_M - m_M| \geq \varepsilon m/2 \vee X_H \neq m_H | \mathcal{E}) + P(\mathcal{E}^C) & (3.26) \\ \leq & P(|X_L - m_L| \geq \varepsilon m/2) + P(|X_M - m_M| \geq \varepsilon m/2 | \mathcal{E}) + (X_H \neq m_H | \mathcal{E}) + P(\mathcal{E}^C & (3.27) \\ \leq & \frac{1}{8} + \frac{1}{8} + 0 + \frac{1}{12} = \frac{1}{3} & (3.28) \end{aligned}$$

□

By using Fact 21, we may remove the need for advice, giving us

Corollary 38. *There is an algorithm that runs in expected time* $O(\frac{n\sqrt{\log n}}{\varepsilon\sqrt{m}} + \frac{\log^2 n}{\varepsilon^2})$ *and returns* $\hat{m}$ *such that with probability at least* $2/3$*, it holds* $|X - m| \leq \varepsilon m$*.*

Algorithm with pair queries

We define $u \prec v$ if $d(u) < d(v)$ or $d(u) = d(v)$ and $id(u) \leq id(v)$. We consider an orientation of edges such that uv is oriented from u to v such that $u \prec v$. We use $d^+(v)$ to denote the out-degree of v.

In this section, we use a notion of light and heavy vertices that is slightly different from the one we have used above. We divide the vertices into a set of *light* V_L and a set of *heavy* vertices V_H. We assume that for every $v \in V_H, d^+(v) \geq \theta/2$ and for every $v \in V_L, d^+(v) \leq 2\theta$. The vertices with $d^+(v)$ between $\theta/2$ and 2θ can be assigned to either V_L or V_H (but not both).

We now describe two algorithms we use to classify vertices between being light or heavy. We need this classification to be consistent. When we first decide whether a vertex is light or heavy, we store this decision and use it if the same vertex is later queried. Assume we are given access to independent Bernoulli trials with bias p. The algorithm of Lipton et al. [1993] give $\hat{p}$ such that $P((1-\varepsilon)p \leq \hat{p} \leq (1+\varepsilon)p) \geq 1 - \delta$ while running in time $O(\frac{\log \delta^{-1}}{p\varepsilon^2})$ [16]. Using this algorithm, we can classify v with high probability in time $O(\frac{d(v)\log n}{d^+(v)})$. Alternatively, using standard Chernoff bounds, one may classify vertices w.h.p. in time $O(\frac{d(v)\log n}{\theta})$. We assume that, with probability $1 - 1/n$, all classifications are correct.

[16] In that paper, the authors in fact solve a more general problem. For presentation of this specific case, see Watanabe [2005]

Algorithm 20: Approximately count edges of G given advice $\tilde{m}$

```
$m_\uparrow = \tilde{m}/2$
$m_\downarrow = 2\tilde{m}$
$\theta \leftarrow \varepsilon\sqrt{m_\downarrow}$
$\tau \leftarrow \sqrt{m_\downarrow/(8\varepsilon)}$

$A_1 \leftarrow 0$
repeat $k = 432\frac{\theta n}{\varepsilon^2 m_\uparrow}$ times
    $v \leftarrow$ random vertex
    $w \leftarrow$ random neighbor of $v$
    if $v \prec w$ and $v$ is light (use the algorithm from Lipton et al. [1993]
      to classify vertices as light/heavy) then
        $A_1 \leftarrow A_1 + d(v)$
$\hat{d}^+_L \leftarrow \frac{nA_1}{k}$

$S \leftarrow$ sample $\frac{48n\log n}{\theta}$ vertices with replacement    // Note that $S, S'$
  are multisets, not sets
$S' \leftarrow$ vertices of $S$ with degree $\geq \theta$
if $|S'| > \frac{576 m_\downarrow \log n}{\theta^2}$ or $d(S') > \frac{1152 m_\downarrow \log n}{\theta}$ then
    return "failure"
for $i$ from 1 to $k_2 = \frac{468}{\varepsilon^2}$ do
    $T \leftarrow$ Sample each edge $\vec{uv}$ incident to $S'$ with probability $p = \frac{\theta}{m_\uparrow}$
    $T' \leftarrow$ Set of all vertices $v$ such that $uv \in T$, $d(v) \leq \tau$ and $v$ is
      heavy (use the standard Chernoff-bound-based algorithm
      described above to classify vertices as light/heavy)
    For each vertex $v \in T'$, let $r(v) = |N(v) \cap S'|$    // Compute by
      using a pair query for each pair $v, w$ for $w \in S'$
    $A_{2,i} \leftarrow 0$
    for $v \in T'$ do
        $w \leftarrow$ random neighbor of $v$
        if $v \prec w$ then
            $A_{2,i} \leftarrow A_{2,i} + d(v)/(1-(1-p)^{r(v)})$
$\hat{d}^+_H \leftarrow \frac{\sum_{i=1}^{k_2} A_{2,i}}{k_2}$
return $\hat{d}^+_L + \hat{d}^+_h$
```

We will again use the algorithm described in Chapter 3 to remove the need for advice $\tilde{m}$. In the following theorem, we prove two deviation bounds. The second one is the one that will give us the approximation guarantee of the final algorithm, while the first one will allow us to remove the need for advice.

We are assuming in the algorithm that conditions in **if** statements are evaluated in order and the evaluation is stopped when the result is already known (e.g., in **if**$(\phi \wedge \psi)$, if ϕ evaluates to false, ψ would not be evaluated).

Theorem 39. *Given $\tilde{m}$ and $\varepsilon > 0$, Algorithm 20 returns $\hat{m}$ such that $P(\hat{m} > 7m) \leq 1/3$ and runs in time $O(\frac{n \log n}{\varepsilon\sqrt{\tilde{m}}} + \frac{\log^2 n}{\varepsilon^4})$. If, moreover, $m \leq \tilde{m} \leq 2m$, then with probability at least $2/3$, $\hat{m} \in (1 \pm \varepsilon)m$.*

Proof. We start by proving that when $m \leq \tilde{m} \leq 2m$, then with probability at least $2/3$, $\hat{m} \in (1\pm\varepsilon)m$ ("correctness"). We then prove that $P(\hat{m} > 4m) \leq 1/3$ ("bounds for advice removal"). We then finish by proving the time complexity ("time complexity").

We now prove correctness. Let d_l^+, d_h^+ be the sum of out-degrees of the light and heavy vertices, respectively. It holds $m = d_L^+ + d_h^+$. Throughout the proof, we condition on all light vertices having out-degree at most 2θ and all heavy vertices having out-degree at least $\theta/2$. As we said above, we are assuming this holds with probability at least $1 - 1/n$ for all vertices. We call this event $\mathcal{E}$.

We now prove that $\hat{d}_l^+$ is a good estimate of d_l^+. Let $A_{1,i}$ be the increment of A_1 in the i-th execution of the loop on line 6.

$$
\begin{aligned}
E(A_{1,i}) &= \sum_{u\in V} \mathbb{I}(\text{u is light}) P(u = v) P(w \succ v) d(v) && (3.29)\\
&= \sum_{u\in V} \mathbb{I}(\text{u is light}) \frac{1}{n} \cdot \frac{d^+(v)}{d(v)} d(v) && (3.30)\\
&= \sum_{u\in V} \mathbb{I}(\text{u is light}) \frac{1}{n} d^+(v) = d_l^+/n && (3.31)
\end{aligned}
$$

$$
\begin{aligned}
Var(A_{1,i}) \leq E(A_{1,i}^2) &= \sum_{u\in V} \mathbb{I}(\text{u is light}) P(u = v) P(w \succ v) d(v)^2 && (3.32)\\
&= \sum_{u\in V} \mathbb{I}(\text{u is light}) \frac{1}{n} d^+(v) d(v) && (3.33)\\
&\leq \sum_{u\in V} \frac{1}{n} 2\theta d(v) = \frac{4\theta m}{n} && (3.34)
\end{aligned}
$$

Therefore, $\hat{d}_l^+$ is an unbiased estimate of d_l^+ (conditioning on the correct classification of all light/heavy vertices). Its variance is $\frac{n^2}{k^2} \cdot k \cdot \frac{4\theta m}{n} \leq \frac{1}{108}\varepsilon^2 m^2$.

By the Chebyshev inequality, we have that

$$P(|\hat{d}_L^+ - d_l^+| > \varepsilon m/3) \leq \frac{\varepsilon^2 m^2/108}{(\varepsilon m/3)^2} \leq 1/12 \tag{3.35}$$

We now focus on the heavy vertices. There are at most $2m/\theta$ heavy vertices. Each one is sampled into S in expectation $\frac{48 \log n}{\theta}$ times. Therefore, there are in expectation at most $\frac{2m}{\theta} \cdot \frac{48 \log n}{\theta} = \frac{96 m \log n}{\theta^2}$ vertices in S'. Similarly, because each vertex is sampled in expectation $\frac{48 \log n}{\theta}$ times, it holds that $E(d(S')) \leq E(d(S)) \leq 2m \frac{48 \log n}{\theta} = \frac{96 m \log n}{\theta}$.

The condition on line 14 is set such that the algorithm only fails on line 15 when $|S'| > 12E(|S'|)$ or $|d(S')| > 12E(|d(S')|)$. By the Markov's inequality and the union bound, with probability at least $1/6$, neither of these inequalities is satisfied. In the rest of the algorithm, it holds $|S'| \leq \frac{576 m_\downarrow \log n}{\theta^2}$ and $|d(S')| \leq \frac{1152 m_\downarrow \log n}{\theta}$. We will use this when arguing the time complexity. When analyzing correctness, we do not condition on the condition on line 14 not being satisfied.

We now argue that $\hat{d}_h^+$ is a good estimate of d_h^+. Specifically, we prove that it holds with probability at least $11/12$ that $\hat{d}_H^+ \in d_H^+ \pm \frac{2}{3}\varepsilon m$. Let u be a heavy vertex with $d(u) \leq \tau$. It holds $d^+(u) \geq \theta/2$. Consider the value $r(u)$. Since each vertex is sampled into S in expectation $\frac{48 \log n}{\theta}$ times, it holds that $E(r(u)) \geq d^+(u) \frac{48 \log n}{\theta} \geq \theta/2 \cdot \frac{48 \log n}{\theta} = 24 \log n$ for any heavy vertex u. It holds with probability at least $1 - 1/n^2$ that $r(u) \geq \frac{24 d^+(u) \log n}{\theta}$ because by the Chernoff bound, we have that

$$P(r(u) < \frac{24 d^+(u) \log n}{\theta}) \leq P(r(u) < E(r(u))/2) \tag{3.36}$$

$$\leq \exp(-\frac{E(r(u))}{12}) \tag{3.37}$$

$$\leq \exp(-2 \log n) = 1/n^2 \tag{3.38}$$

and by the union bound, this inequality holds for all heavy vertices simultaneously with probability at least $1 - 1/n$. We condition on this event in what follows, we call it $\mathcal{E}'$. In fact, we will condition on S', and we assume that this inequality holds for S'. We now analyze the conditional expectation $E(A_{2,1}|S')$ (note that the expectation $E(A_{2,i}|S')$ is the same for all i and we may thus focus on $i = 1$).

$$E(A_{2,1}|S') = \sum_{u\in V} P(u \in T'|S')P(w \succ v)\frac{d(v)}{(1-(1-p)^{r(v)})} \tag{3.39}$$

$$= \sum_{u\in V} \mathbb{I}(d(u) \le \tau \text{ and } u \text{ is heavy})(1-(1-p)^{r(v)}) \cdot \frac{d^+(u)}{d(u)} \cdot \frac{d(v)}{(1-(1-p)^{r(v)})} \tag{3.40}$$

$$= \sum_{u\in V} \mathbb{I}(d(u) \le \tau \text{ and } u \text{ is heavy})d^+(u) \tag{3.41}$$

This counts all heavy edges whose lower-degree endpoint has degree at most τ. All the uncounted edges are therefore in the subgraph induced by these high-degree vertices. There are at most $\frac{2m}{\tau} = \sqrt{\varepsilon m/2}$ vertices with degree $> \tau$. This means that there can be at most $\binom{\sqrt{\varepsilon m/2}}{2} < \varepsilon m/3$ uncounted edges. Therefore, it follows that $|E(A_{2,1}|S') - d^+_h| \le \varepsilon m/3$.

We now analyze the conditional variance. Recall that we are assuming that for S', it holds that all heavy vertices v have $r(v) \ge \frac{24d^+(v)\log n}{\theta}$.

$$Var(A_{2,1}|S') \le E(A_{2,1}^2|S') = \sum_{u\in V} P(u \in T'|S')P(w \succ v)\Big(\frac{d(u)}{(1-(1-p)^{r(u)})}\Big)^2 \tag{3.42}$$

$$\le \sum_{u\in V} \mathbb{I}(u \text{ is heavy})(1-(1-p)^{r(u)})\frac{d^+(u)}{d(u)} \cdot \frac{d(u)^2}{(1-(1-p)^{r(u)})^2} \tag{3.43}$$

$$\le \sum_{u\in V} \mathbb{I}(u \text{ is heavy})\frac{d^+(u)}{d(u)} \cdot \frac{d(u)^2}{(1-(1-p)^{\frac{d^+(u)}{\theta}})} \tag{3.44}$$

$$\le \sum_{u\in V} \frac{2d^+(u)d(u)}{\frac{d^+(u)}{\theta} \cdot \frac{\theta}{m}} \tag{3.45}$$

$$= 2\sum_{u\in V} d(u)m = 4m^2 \tag{3.46}$$

where the second inequality holds because we are conditioning on $r(u) \ge \frac{24d^+(v)\log n}{\theta} \ge \frac{d^+(v)}{\theta}$ [17] and the third holds because for x, y such that $xy < 1, 1 > x > 0, y \ge 1$, it holds that $1-(1-x)^y \ge xy/2$ and $p = \theta/m_\uparrow \ge \theta/m$. On the event $\mathcal{E}'$, the expectation $E(A_{2,1}|S')$ is independent of S'. By the law of total variance, $Var(A_{2,1}|\mathcal{E}') = E(Var(A_{2,1}|S', \mathcal{E}')|\mathcal{E}') \le 4m^2$. Therefore,

[17] We intentionally do not use the tightest possible bound in order to allow us to use the next inequality. It is possible to slightly improve the constants by a more technical analysis.

$Var(\hat{d}_H^+|\mathcal{E}') \leq \frac{\varepsilon^2}{468} Var(A_{2,1}|\mathcal{E}') \leq \varepsilon^2 m^2/117$. It now holds by the (conditional) Chebyshev inequality that

$$P(|\hat{d}_H^+ - E(\hat{d}_h^+)| > \varepsilon m/3 | \mathcal{E}') \leq \frac{\varepsilon^2 m^2/117}{(\varepsilon m/3)^2} \leq 1/13 \qquad (3.47)$$

Putting this together with the union bound with probability bounds on the events of failure on line 15 (probability $\leq 1/6$), event of $|\hat{d}_L^+ - d_l^+| > \varepsilon m/3$ (probability $\leq 1/12$) and the events $\mathcal{E}^C, \mathcal{E}'^C$ (probability $\leq 1/n$), we get that with probability at least $2/3$, it holds $|\hat{d}_L^+ - d_l^+| \leq \varepsilon m/3$, $|\hat{d}_H^+ - E(\hat{d}_h^+)| \leq \varepsilon m/3$, and $|E(\hat{d}_h^+) - \hat{d}_h^+| \leq \varepsilon m/3$. On this event, it holds by the triangle inequality that $|\hat{m} - m| \leq \varepsilon m$. This proves correctness.

We now prove the bounds for advice removal. We have shown that $E(d_l^+) = d_l^+$ and $E(d_H^+|\mathcal{E}') \leq d_h^+$. By the Markov's inequality, $P(\hat{d}_L^+|\mathcal{E}' \geq 7d_k^+) \geq 1/7$ and $P(\hat{d}_H^+|\mathcal{E}' \geq 7d_h^+) \geq 1/7$. By the union bound, both hold with probability at least $2/7$. Adding the probability of $\mathcal{E}^C$ and $\mathcal{E}'^C$, upper bounded by $1/n$, we get that $P(\hat{m} \geq 7m) \leq 1/3$

We now prove the claimed time complexity bound. We first focus on the first part (lines 5 - 13). There are $O(\frac{\theta n}{\varepsilon^2 m}) = O(\frac{n}{\varepsilon\sqrt{m}})$ repetitions. It holds $P(v \prec w) = d^+(v)/d(v)$. Determining whether v is light on line 9 takes $O(\frac{d(v)\log n}{d^+(v)})$. However, we only need to determine whether v is light when $v \prec w$, which happens with probability $\frac{d^+(v)}{d(v)}$. This, therefore, takes in expectation $O(\log n)$ time. This dominates the expected cost of an iteration, leading to total expected running time of $O(\frac{n \log n}{\varepsilon\sqrt{m}})$.

We now focus on the second part of the algorithm (lines 14 - 31). Computing S clearly takes $O(\frac{n\log n}{\theta})$ time. We now analyze one iteration of the loop on line 16. It holds that $d(S') \leq O(\frac{m \log n}{\theta})$. Each of the incident edges is sampled with probability $\leq \frac{2\theta}{m}$. The expected size of T is thus $O(\frac{m\log n}{\theta}) \cdot \frac{2\theta}{m} = O(\log n)$. This is also an upper bound on the size of T' as well as on the time it takes to compute T. In computing T', we classify each endpoint v of an edge $\vec{uv}$ of T with $d(v) \leq \tau$. The running time of this is $O(\frac{d(v)\log n}{\theta}) \leq O(\frac{\log n}{\varepsilon^{3/2}})$. Since $|T| = O(\log n)$, it takes $O(\frac{\log^2 n}{\varepsilon^{3/2}})$ time to compute T'. To calculate $r(v)$ for all v, we must query $|S'||T'|$ vertex pairs. Since $|S'| \leq O(\frac{m \log n}{\theta^2})$, it holds that $E(|S'||T'|) \leq O(\frac{m\log n}{\theta^2})E(|T'|) \leq O(\frac{m \log^2 n}{\theta^2}) = O(\frac{\log^2 n}{\varepsilon^2})$. The rest of the iteration has time complexity $O(|T'|)$, thus not increasing the total time complexity. Since there are $O(\frac{1}{\varepsilon^2})$ iterations, this gives a bound on the time complexity of the second part of $O(\frac{n\log n}{\theta} + \frac{\log^2 n}{\varepsilon^4}) = O(\frac{n \log n}{\varepsilon\sqrt{m}} + \frac{\log^2}{\varepsilon^4})$. This is then also the time complexity of the whole algorithm. □

By using the methods described in Chapter 3, we get the following:

Corollary 40. *There is an algorithm that, given $\varepsilon > 0$, returns $\tilde{m}$ such that with probability at least $2/3$, $\hat{m} \in (1 \pm \varepsilon)m$ and has expected time complexity $O(\frac{n \log n}{\varepsilon\sqrt{\tilde{m}}} + \frac{\log^2 n}{\varepsilon^4})$.*

Lower bound

In this section, we show a lower bound for approximate edge counting of $\Omega(\frac{n}{\varepsilon\sqrt{m}})$ for $\varepsilon \gtrsim \frac{\sqrt{m}}{n}$. This improves on this range over the previously known $\Omega(\frac{n}{\sqrt{\varepsilon m}})$ of Goldreich and Ron [2008]. Similarly to their result, our proof works even for the query complexity in the model where, for each accessed vertex, the algorithm gets the whole connected component of that vertex at unit cost. This is a considerably stronger model than our hash-ordered neighbor access model as well as the full neighborhood access model.

Our improvement comes from the fact that we are using anti-concentration results instead of relying just on the difficulty of hitting once a subset of vertices. Specifically, our proof relies on the following lemma:

Lemma 41. *Suppose we have a finite set A of tuples (i, t) where i is a unique identifier and $t \in \{0, 1\}$. Then any procedure $\mathcal{M}$ that can distinguish with probability at least $2/3$ between the case when of all $(i, t) \in A$, a $\frac{1}{2} - \varepsilon$-fraction have $t = 0$ and $\frac{1}{2} + \varepsilon$-fraction have $t = 1$ and the case when a $\frac{1}{2} + \varepsilon$-fraction have $t = 0$ and $\frac{1}{2} - \varepsilon$-fraction have $t = 1$, has to use in expectation $\Omega(\min(|A|, \frac{1}{\varepsilon^2}))$ uniform samples from A.*

Proof. Assume the existence of $\mathcal{M}$ that uses $o(\min(|A|, \frac{1}{\varepsilon^2}))$ samples. We show that this entails a contradiction.

If we sample k elements without replacement instead of with replacement, it is possible to simulate sampling k elements with replacement, assuming we know $|A|$. Existence of such $\mathcal{M}$ would imply the existence of algorithm $\mathcal{M}'$ with the same guarantees (distinguishing with probability at least $2/3$ between the cases when $\frac{1}{2} - \varepsilon$-fraction have $t = 0$ and $\frac{1}{2} + \varepsilon$-fraction have $t = 1$ and the case when a $\frac{1}{2} + \varepsilon$-fraction have $t = 0$ and $\frac{1}{2} - \varepsilon$-fraction have $t = 1$ using $o(\min(|A|, \frac{1}{\varepsilon^2}))$ samples) which uses the same sample size but samples without replacement.

By symmetry, there is an optimal algorithm which does not use the knowledge of i of the sampled elements. Therefore, if there is such a procedure $\mathcal{M}'$, there also has to be a procedure $\mathcal{M}''$ with the same guarantees that depends only on the number of sampled elements with $t = 1$ and the number of sampled elements with $t = 0$. That is, $\mathcal{M}''$ distinguishes the distributions $H_1 \sim HGeom(n, (\frac{1}{2} - \varepsilon)n, k)$ and $H_2 \sim HGeom(n, (\frac{1}{2} + \varepsilon)n, k)$ with probability at least $2/3$. Specifically, $P(\mathcal{M}''(H_i) = i) \geq 2/3$ for $i \in \{0, 1\}$. We now show that such $\mathcal{M}''$ cannot exist, thus proving the lemma.

Let $B_1 \sim Bin(k, \frac{1}{2} - \varepsilon)$ and $B_2 \sim Bin(k, \frac{1}{2} + \varepsilon)$. It is known ([Diaconis and Holmes, 2004, Theorem 3.2]) that $\|Bin(k, p) - HGeom(n, pn, k)\|_{TV} \leq$

$(k-1)/(n-1)$. Thus, there exists a coupling of B_1, B_2 and H_1, H_2 such that for fixed $i \in \{0,1\}$, it holds that $P(B_i \neq H_i) \leq (k-1)/(n-1)$. For $k \leq n/11$, it then holds that $P(B_1 \neq H_1) \leq 1/10$ for n sufficiently large (specifically, larger than 10). Now

$$P(\mathcal{M}''(B_i) = i) \geq P(\mathcal{M}''(B_i) = i) - P(B_1 \neq H_1) \geq 2/3 - 1/10 \geq 0.55$$

It is known ([Anthony and Bartlett, 1999, Lemma 5.1]) that $Bin(k, \frac{1}{2} - \varepsilon)$ and $Bin(k, \frac{1}{2} + \varepsilon)$ cannot be distinguished with probability at least 0.55 when $k = o(\frac{1}{\varepsilon^2})$. This implies that $\mathcal{M}''$ does not exist. Therefore $\mathcal{M}$ cannot use $o(\min(|A|, \frac{1}{\varepsilon^2}))$ samples cannot exist (note the first branch of the min which comes from the assumption that $k \leq n/11$). □

Using this lemma, we can now prove the following theorem. In the proof, we use the notation $\sim f(x)$, for example saying that the graph has $\sim m$ edges. The meaning in this case is that the graph has m' edges such that $m' \sim m$.

Theorem 42. *For $\varepsilon \geq \frac{4\sqrt{m}}{n}$, any algorithm that with probability at least 2/3 outputs $\tilde{m}$ such that $\tilde{m} = (1 \pm \varepsilon)m$, has to use $\Omega(\frac{n}{\varepsilon\sqrt{m}})$ samples.*

Proof. We construct two graphs G_1, G_2, one with $\sim (1-2\varepsilon)m$ and the other with $\sim (1+2\varepsilon)m$ edges and then show that it is hard to distinguish between them. Define dense chunk S_d as a complete graph with β vertices. Similarly, define a sparse chunk S_s as an independent set on β vertices.

We now describe the graphs G_1, G_2. They both consist of α chunks and an independent set of size $n - \alpha\beta$. The graph G_1 has a $(\frac{1}{2} + \varepsilon)$-fraction of chunks being sparse and the rest being dense, whereas G_2 has a $(\frac{1}{2} - \varepsilon)$-fraction of the chunks being sparse and the rest being dense.

Note that this means that the sparse chunks form, together with the vertices which are not part of any chunk, an independent set. Nevertheless, it will be useful to separately consider the sparse chunks and vertices not part of any chunk.

We set $\beta = \varepsilon\sqrt{m}$ and $\alpha = \frac{4}{\varepsilon^2}$. The graph G_1 now has

$$(\frac{1}{2} - \varepsilon)\alpha\binom{\beta}{2} \sim (1-2\varepsilon)\frac{(\varepsilon\sqrt{m})^2}{\varepsilon^2} = (1-2\varepsilon)m$$

edges. By a similar calculation, G_2 has $\sim (1+2\varepsilon)m$ edges. Note that the number of vetrtices in the chunks is

$$\alpha\beta = \varepsilon\sqrt{m}\frac{4}{\varepsilon^2} = \frac{4\sqrt{m}}{\varepsilon} \leq n$$

where the last inequality holds by the assumption on ε. The described graph, therefore, does exist.

We now define graphs G_1, G_2, corresponding to the two cases from Lemma 41. It then follows from the lemma that any algorithm that can tell apart with probability at least 2/3 between G_1 and G_2 has to use $\Omega(\frac{n}{\varepsilon\sqrt{m}})$ samples. In fact, we prove a stronger statement, namely that this is the case even if the algorithm receives for each sampled vertex information about which chunk the vertex is from as well as the type of the chunk (or that it does not belong to a chunk).

Assuming the algorithm makes k samples, let X_i for $i \in [k]$ be indicator for whether the i-th sample hit one of the chunks. It holds that $E(X_i) = \frac{\alpha\beta}{n}$ for any i. It follows from Lemma 41 that any algorithm satisfying the conditions from the statement has to hit the chunks in expectaton at least $\Omega(\frac{1}{\varepsilon^2})$ times. If k is the number of samples, this implies that

$$E(k)\frac{\alpha\beta}{n} = E(\sum_{i=1}^{k} X_1) \gtrsim \frac{1}{\varepsilon^2}$$

which, solving for $E(k)$, gives

$$E(k) \gtrsim \frac{n}{\varepsilon\sqrt{m}}$$

□

Triangle counting with full neighborhood access

We start by giving several definitions that we use in this section. Given an edge e, we denote by $t(e)$ the number of triangles that contain e. Note that in the full neighborhood model, $t(e)$ may be computed in $O(1)$ queries. Specifically, for $e = uv$, it holds $t(e) = |N(u) \cap N(v)|$. We now define order of edges $\prec$ so that $e_1 \prec e_2$ iff $t(e_1) < t(e_2)$ or $t(e_1) = t(e_2)$ and $id(e_1) \leq id(e_2)$. We assign each triangle to its edge that is minimal with respect to $\prec$. Given an edge e, we let $t(e)$ be the number of triangles assigned to e. Note that, in contrast with $t(e)$, we may not in general compute $t^+(e)$ in $O(1)$ queries. Given a set $S \subseteq V$, we define $t^+_S(e)$ to be the number of triangles assigned to $e = uv$ that have non-empty intersection with $S \setminus \{u, v\}$. If we are given the endpoints of an edge e and the set S, we may compute $t^+_S(e)$ without making any additional queries.

Algorithm with edge sampling

We now prove a bound which we will later need in order to bound variance. Essentially the same bound has been proven in Kallaugher et al. [2019]. We give a slightly different proof, which we later modify to prove Lemma 48.

Lemma 43. *It holds that*

$$\sum_{e \in E} t^+(e)t(e) \leq 46T^{4/3}$$

Algorithm 21: Count triangles approximately, given advice $\tilde{T}$

```
A ← 0
repeat $k = 138\frac{m}{\varepsilon^2 \tilde{T}^{2/3}}$ times
    $uv = e \leftarrow$ pick an edge uniformly at random
    $w \leftarrow$ random vertex from $N(u) \cap N(v)$
    if $uv \prec uw$ and $uv \prec vw$ then
        $A \leftarrow A + t(e)$
return $\frac{mA}{k}$
```

Proof. We start by arguing several inequalities which we will use to bound the variance. We pick a permutation π of the vertices such that $t(e_{\pi(1)}) \geq t(e_{\pi(2)}) \geq \cdots \geq t(e_{\pi(m)})$. Let us have $i \in \{2^{k-1}+1, \cdots, 2^k\}$, it holds (a) that $t(e_{\pi(i)}) \leq \frac{3T}{2^{k-1}}$. Otherwise, the first i edges in the π-ordering would have total of $i\frac{3T}{2^{k-1}} > 3T$ edge-triangle incidencies, which is in contradiction with T being the number of triangles. We now bound $\sum_{i=2^{k-1}+1}^{2^k} t^+(e_{\pi(i)})$. It clearly holds (b) that $\sum_{i=2^{k-1}+1}^{2^k} t^+(e_{\pi(i)}) \leq T$. For any triangle $e_{\pi(i)}e_{\pi(j)}e_{\pi(\ell)}$ assigned to $e_{\pi(i)}$, it holds that $i > j$,$i > \ell$. Therefore, any triangle assigned to an edge $e_{\pi(i)}$ is formed by edges in $\{e_{\pi(j)}\}_{j=1}^{2^k}$. On 2^k edges, there can be at most $\sqrt{2}\,2^{3k/2}$ triangles[18]. Therefore, we have (c) that $\sum_{i=2^{k-1}+1}^{2^k} t^+(e_{\pi(i)}) \leq \sqrt{2}\,2^{3k/2}$.

In what follows, we use (in a slight abuse of notation) the convention $t(e_{\pi(i)}) = t^+(e_{\pi(i)}) = 0$ for $i > m$. We may now use the above bounds to prove

[18] The argument is as follows. Consider a graph on m edges. Order vertices in the order of decreasing degrees. Each vertex has to its left $d^+(v) \leq \sqrt{2m}$ of its neighbors: this is clearly the case for the first $\sqrt{2m}$ vertices; it is also the case for any other vertices as otherwise the graph would necessarily have $> m$ edges. Then for $T \leq \sum_v d^+(v)^2 \leq \sqrt{2m}\sum_v d^+(v) = \sqrt{2}m^{3/2}$.

the desired bound:

$$
\begin{aligned}
\sum_{e\in E} t^+(e)t(e) &= \sum_{i=1}^{m} t^+(e_{\pi(i)})t(e_{\pi(i)}) && (3.48)\\
&= \sum_{k=1}^{\lceil \log_2 m\rceil} \sum_{i=2^{k-1}+1}^{2^k} t^+(e_{\pi(i)})t(e_{\pi(i)}) && (1)\\
&\leq 6 \sum_{k=1}^{\lceil \log_2 m\rceil} \frac{T}{2^k} \sum_{i=2^{k-1}+1}^{2^k} t^+(e_{\pi(i)}) && (2)\\
&\leq 6 \sum_{k=1}^{\lceil \log_2 m\rceil} \frac{T}{2^k} \min(\sqrt{2}\, 2^{3k/2}, \sqrt{2}T) && (3)\\
&\leq \sqrt{2}\, 6 \left(\sum_{k=1}^{\lfloor \frac{2}{3}\log_2 T\rfloor} 2^{k/2} T + \sum_{k=\lceil \frac{2}{3}\log_2 T\rceil}^{\infty} \frac{T^2}{2^k} \right) && (3.49)\\
&\leq \sqrt{2}\, 6 \Big((2+\sqrt{2}) \cdot 2^{\frac{1}{3}\log_2 T} T + 2 \frac{T^2}{2^{\frac{2}{3}\log_2 T}} \Big) && (3.50)\\
&< 46 T^{4/3} && (3.51)
\end{aligned}
$$

where (1) is using the convention $t(e_{\pi(i)}) = t^+(e_{\pi(i)}) = 0$ for $i > m$, (2) holds by inequality (a) and (3) holds by inequalities (b) and (c). Note that in the second branch of min, we are using the upper-bound $\sqrt{2}T$ instead of T for convenience. □

Lemma 44. *Given $\tilde{T}$, Algorithm 21 returns $\hat{T}$ which is an unbiased estimator of T. It has query complexity $O(\frac{m}{\varepsilon^2 \tilde{T}^{2/3}})$. If, moreover, $\tilde{T} \leq T$, then with probability at least $2/3$, it holds $\hat{T} \in (1 \pm \varepsilon)T$.*

Proof. The query complexity is clearly as claimed. We now argue unbiasedness and the deviation bounds. Let A_i be the increment in A in the i-th iteration of the loop. We now compute $E(A_1)$ and upper-bound $Var(A_1)$. It holds

$$E(A_1|e) = P(uv \prec uw \wedge uv \prec vw)t(e) = \frac{t^+(e)}{t(e)}t(e) = t^+(e) \qquad (3.52)$$

By the law of total expectation, $E(A_1) = E(t^+(e)) = T/m$ as each triangle is assigned to one edge, and the average number of triangles assigned per edge is thus T/m. We now analyze the variance.

$$\begin{aligned} Var(A_1) \leq E(A_1^2) &= \frac{1}{m}\sum_{e\in E}\frac{t^+(e)}{t(e)}t(e)^2 && (3.53)\\ &= \frac{1}{m}\sum_{e\in E}t^+(e)t(e) && (3.54)\\ &= \frac{46T^{4/3}}{m} && (3.55)\end{aligned}$$

We thus have $E(A) = kT/m$ and $Var(A) = kVar(A_1)$ and hence $E(\hat{T}) = T$ and

$$Var(\hat{T}) = m\frac{46T^{4/3}}{k} = \frac{1}{3}\varepsilon^2 T^{4/3}\tilde{T}^{2/3} \leq \frac{1}{3}\varepsilon^2 T^2$$

where the last inequality holds for $\tilde{T} \leq T$. The expectation of $\hat{T}$ is thus as desired. By Chebyshev's inequality, we have that

$$P(|\hat{T} - T| > \varepsilon T) < \frac{1/3\varepsilon^2 T^2}{(\varepsilon T)^2} = 1/3$$

□

Algorithm with both vertex and edge sampling

Finding Heavy Subgraph

Definition 45. *Let us be given a parameter $\theta > 0$. A vertex v is heavy with respect to a set S if $\sum_{uv,u\in S} t(uv) \geq 8\sqrt{\theta}\log n$. Otherwise, it is light. Let $V_H(S), V_L(S)$ be the set of heavy and light vertices w.r.t. S, respectively.*

We use just V_H, V_L when the set S is clear from the context. When testing whether a vertex is heavy, we assume we are already given the set S and the vertex v. We then do not make any additional queries to tell whether v is heavy or light (as the sum in the definition of a heavy vertex only depends on the neighborhoods of S, v, which we know since we have already queried v as well as all vertices of S). In our algorithm, we will set S to be a subset of vertices such that each vertex is in S independently with probability $16\log n/\sqrt{\theta}$. We now prove some guarantees on which vertices will be heavy and how many heavy vertices there will be.

Lemma 46. *Let us have a parameter θ and a vertex v. Assume S includes each vertex independently with probability $16\log n/\sqrt{\theta}$. Assume that at least θ triangles are assigned to the edges incident to v. That is, assume $\sum_{e\ni v} t^+(e) \geq \theta$. Then, with probability at least $1 - 1/n^2$, the vertex v is heavy.*

Proof. In the whole proof, we treat v as given and define values based on v without explicitly specifying this throughout the proof. Let $t^+_{max} = \max_{e \ni v} t^+(e)$. For an edge e, we define $t^m(e) = \min(t^+_{max}, t(e))$. Let $T^m_v = \sum_{e \ni v} t^m(e)$.

We will now argue that for any edge e incident to vertex v, it holds $t^m(e) \leq \sqrt{T^m_v}$. Consider the triangles assigned to e. Each of the $t^+(e)$ triangles assigned to e consists of e, one other edge e' incident to v and one edge not incident to v. It holds $t(e') \geq t(e) \geq t^+(e)$ for otherwise the triangle would be assigned to e' instead of e. Thus, it also holds $t^m(e') = \min(t^+_{max}, t(e')) \geq t^+(e)$. Since there are $t^+(e)$ such edges e' (as there is a 1-to-1 correspondence between such edges and triangles assigned to e) they contribute at least $(t^+(e))^2$ to T^m_v and thus $T^m_v \geq (t^+(e))^2$. Rearranging this, we get $t^+(e) \leq \sqrt{T^m_v}$. Since this holds for any $e \ni v$, it also holds $t^+_{max} \leq \sqrt{T^m_v}$ and thus also $t^m(e) = \min(t^+_{max}, t(e)) \leq \sqrt{T^m_v}$.

This bound on $t^m(e)$ allows us to now use the Chernoff bound to prove concentration of $\sum_{uv, u \in S} t^m(uv)$ around $E(\sum_{uv, u \in S} t^m(uv)) = 16 T^m_v \log n / \sqrt{\theta}$. This gives us that $\sum_{uv, u \in S} t(uv)$ is not too small and v is thus heavy. Specifically, we get that

$$P(\sum_{uv, u \in S} t(uv) < 8\sqrt{\theta} \log n) \leq P(\sum_{uv, u \in S} t^m(uv) < 8 T^m_v \log n / \sqrt{\theta}) < \exp(-\frac{16 T^m_v \log n / \sqrt{\theta}}{8\sqrt{T^m_v}}) \leq 1/n^2$$

where the first inequality holds because $T^m_v \geq \sum_{e \ni v} t^+(uv) \geq \theta$ and thus $\sqrt{\theta} \leq T^m_v / \sqrt{\theta}$, the second inequality holds by the Chernoff bound, and the third holds again because $T^m_v \geq \theta$. □

Lemma 47. *Assume S includes each vertex independently with probability $16 \log n / \sqrt{\theta}$. It holds $E(|V_H(S)|) \leq 6T/\theta$.*

Proof. Let T_v be the number of triangles containing vertex v. We can now bound $E(|V_H|)$ as follows

$$E(|V_H|) = \sum_{v \in V} E(\mathcal{I}(v \in V_H)) \tag{3.56}$$

$$= \sum_{v \in V} P(\sum_{u \in N(v) \cap S} t(uv) \geq 8\sqrt{\theta} \log n) \tag{3.57}$$

$$\leq \sum_{v \in V} \frac{E(\sum_{u \in N(v) \cap S} t(uv))}{8\sqrt{\theta} \log n} \tag{3.58}$$

$$= \sum_{v \in V} \frac{16 T_v \log n / \sqrt{\theta}}{8\sqrt{\theta} \log n} = 6T/\theta \tag{3.59}$$

where the inequality holds by the Markov's inequality. □

Algorithm 22: CountTrianglesVertexSampling($\tilde{T}, \varepsilon$)

```
if $\tilde{T} \geq \frac{m^3}{n^3 \log^6 n}$ then
    Use Algorithm 21 instead
$\theta \leftarrow \sqrt{\frac{m\tilde{T}}{n}}$
$S \leftarrow$ sample each vertex with probability $p_1 = 16 \log n / \sqrt{\theta}$
$S_v \leftarrow$ sample each vertex with probability $p_2 = 100\sqrt{\log n}\theta/(\varepsilon^2 \tilde{T})$
$S_t \leftarrow$ sample each vertex with probability $p_3 = \theta \log n / \tilde{T}$
$N \leftarrow$ sample each vertex with probability $p_4 = \theta \log n/(\varepsilon^2 \tilde{T})$
$A_{v,1} \leftarrow 0$
$A_{v,2} \leftarrow 0$
for $uv \in E(G[S_v])$ do
    if both $u$ and $v$ are light w.r.t. $S$ then
        if $t^+_{S_t}(uv) < 162 \log n / \varepsilon^2$ then
            $B \sim Bern(\varepsilon^2 \tilde{T}^2/\theta^3)$
            if $B = 1$ then
                $w \leftarrow$ random vertex from $N(u) \cap N(v)$
                if $uv \prec uw$ and $uv \prec vw$ then
                    $A_{v,1} \leftarrow A_{v,1} + \theta^3 t(uv)/(\varepsilon^2 \tilde{T}^2)$
        else
            $w \leftarrow$ random vertex from $N \cap N(u) \cap N(v)$
            if $uv \prec uw$ and $uv \prec vw$ then
                $A_{v,2} \leftarrow A_{v,2} + t(uv)$
$A_v \leftarrow A_{v,1} + A_{v,2}$
$\hat{T}_L \leftarrow A_v/p^2$
$S_e \leftarrow$ sample $k = \frac{432m(\log(n)+2)}{\varepsilon^2\theta}$ edges with replacement
$A_e \leftarrow 0$ for $uv \in S_e$ do
    if either $u$ or $v$ is heavy w.r.t. $S$ then
        $w \leftarrow$ random vertex from $N(u) \cap N(v)$
        if $uv \prec uw$ and $uv \prec vw$ then
            $A_e \leftarrow A_e + t(uv)$
$\hat{T}_H \leftarrow \frac{mA_e}{k}$
return $\hat{T}_L + \hat{T}_H$
```

Putting it Together

In the analysis of this algorithms, we will need the following inequality. It is similar to Lemma 43 but is tighter when we are only considering edges with non-empty intersection with some small given set of vertices. We will be using this lemma with the "small set of vertices" being the set of heavy vertices V_H.

Lemma 48. *Let us have a set $V' \subseteq V$ and let $\ell = |V'|$. It holds that*

$$\sum_{\substack{e \in E \\ e \cap V' \neq \emptyset}} t^+(e) t(e) \leq 6(2 + \log m)\ell T$$

Proof. Just like in the proof of Lemma 43, we start by arguing several inequalities. We pick a permutation π of the vertices such that $t(e_{\pi_1}) \geq t(e_{\pi_2}) \geq \cdots \geq t(e_{\pi_m})$. Let us have $i \in \{2^{k-1}+1, \cdots, 2^k\}$, it holds (a) that $t(e_{\pi(i)}) \leq \frac{3T}{2^{k-1}}$. Otherwise, the first i edges in the π-ordering would have total of $i\frac{3T}{2^{k-1}} > 3T$ edge-triangle incidencies, which is in contradiction with T being the number of triangles. We now bound $\sum_{i=2^{k-1}+1}^{2^k} t^+(e_{\pi(i)})$. It clearly holds (b) that $\sum_{i=2^{k-1}+1}^{2^k} t^+(e_{\pi(i)}) \leq T$. For any triangle $e_{\pi(i)}e_{\pi(j)}e_{\pi(\ell)}$ assigned to $e_{\pi(i)}$, it holds that $i > j$,$i > \ell$. Therefore, any triangle assigned to an edge $e_{\pi(i)}$ is formed by edges in $\{e_{\pi(j)}\}_{j=1}^{2^k}$. In a graph on 2^k edges, the number of triangles with non-empty intersection with some given subset V' of vertices of size at most ℓ is at most $\ell 2^k$. Therefore, we have (c) that $\sum_{i=2^{k-1}+1}^{2^k} \mathcal{I}(e_{\pi(i)} \cap V' \neq \emptyset) t^+(e_{\pi(i)}) \leq l2^k$. Like in the previous proof, in what follows, we use (in a slight abuse of notation) the convention $t(e_{\pi(i)}) = t^+(e_{\pi(i)}) = 0$ for $i > m$. We may now use these inequalities to bound the variance:

$$
\begin{aligned}
\sum_{\substack{e\in E\\ e\cap V'\neq\emptyset}} t^+(e)t(e) &= \sum_{i=1}^{m} \mathcal{I}(e_{\pi(i)}\cap V'\neq\emptyset)t^+(e_{\pi(i)})t(e_{\pi(i)}) && (3.60)\\
&= \sum_{k=1}^{\lceil\log_2 m\rceil}\ \sum_{i=2^{k-1}+1}^{2^k} \mathcal{I}(e_{\pi(i)}\cap V'\neq\emptyset)t^+(e_{\pi(i)})t(e_{\pi(i)}) && (4)\\
&\le 6\sum_{k=1}^{\lceil\log_2 m\rceil} \frac{T}{2^k} \sum_{i=2^{k-1}+1}^{2^k} \mathcal{I}(e_{\pi(i)}\cap V'\neq\emptyset)t^+(e_{\pi(i)}) && (5)\\
&\le 6\sum_{k=1}^{\lceil\log_2 m\rceil} \frac{T}{2^k}\min(l2^k, T) && (6)\\
&= \left(\sum_{k=1}^{\log_2 T/\ell} \ell T + \sum_{k=\log_2 T/\ell}^{\infty} \frac{T^2}{2^k}\right) && (3.61)\\
&\le 6\Big(\ell T\log m + 2\ell T\Big) && (3.62)\\
&= 6(2+\log m)\ell T && (3.63)
\end{aligned}
$$

where (4) is using the convention $t(e_{\pi(i)}) = t^+(e_{\pi(i)}) = 0$ for $i > m$, (5) holds by inequality (a) and (6) holds by inequalities (b) and (c). □

We now prove three lemmas, one on (conditional) expectation and variance of $A_{v,1}$, one on $A_{v,2}$ and one on (conditional) expectation of $\hat{T}_H$. Before we can state the lemmas, we will need several definitions.

Let E_L be the set of edges whose both endpoints are light and $E_H = E\backslash E_L$ be the set of edges that have at least one heavy endpoint (note the asymmetry in the definitions). Let $E_{L,1}$ be the subset of E_L of edges e that have $t^+_{S_t}(e) < 162\log n$ and $E_{L,2} = E_L \setminus E_{S,2}$. Let T_L be the number of triangles assigned to edges in $G[V_L(S)]$ and let $T_H = T - T_L$ be the number of triangles assigned to edges having at least one vertex in $V_H(S)$. Let

$$
T_{L,1} = \sum_{\substack{u,v\in V_L\\ t^+_{S_t}(uv)<162\log n}} t^+(uv) \qquad T_{L,2} = \sum_{\substack{u,v\in V_L\\ t^+_{S_t}(uv)\ge 162\log n}} t^+(uv) \qquad (3.64)
$$

(3.65)

Note that $T_L = T_{L,1} + T_{L,2}$. Let us define for an edge uv such that $t^+_{S_t}(e) \ge 162\log n$

$$
F_{N,uv} = \frac{|\{w\in N\cap N(u)\cap N(v),\ \text{s.t. } uv\prec uw, uv\prec vw\}|}{|N\cap N(u)\cap N(v)|}
$$

and we define $F_{N,uv} = t^+(e)/t(e)$ otherwise (note that in this case, the value does not depend on N). We now argue concentration of $F_{N,e}$ around $t^+(e)/t(e)$.

Lemma 49. *With probability at least $1 - 3/n$, it holds for all edges e that $F_{N,e} = (1 \pm \varepsilon/3)t^+(e)/t(e)$.*

Proof. For an edge e with $t^+_{S_t}(e) < 162 \log n$, the claim holds by the way we define $F_{N,e}$ for such edge. Consider now the case $t^+_{S_t}(e) \geq 162 \log n$. By a standard argument based on the Chernoff bound over the choice of S_t, if $t^+(e) < 81\tilde{T} \log n/\theta$, then $t^+_{S_t}(e) \geq 162 \log n$ only on some event $\mathcal{E}$ with probability $\leq 1/n^3$. Conditioned on $\bar{\mathcal{E}}$ (that is, assuming $t^+(e) \geq 81\tilde{T} \log n/\theta$), another application of the Chernoff bound gives that $F_{N,uv} = (1 \pm \varepsilon/3)t^+(e)/t(e)$ with probability at least $1 - 2/n^3$. It then holds by the union bound that with probability at least $1 - 3/n$, we have that $F_{N,e} = (1 \pm \varepsilon/3)t^+(e)$ for all edges e. □

Lemma 50. *It holds $E(\hat{T}_L|S) = T_L$ and, with high probability, $E(\hat{T}_L|S, S_t, N) = (1 \pm \varepsilon/3)T_L$ (where the high probability is over the choice of S, S_t, N)*[19].

Proof. We start by analyzing $E(A_{v,1}|S, S_t)$. For any edge $e \in E_{L,1}$, the probability that it is in the induced subgraph $G[S_v]$ is p_2^2. The probability that $B = 1$ is $\varepsilon^2\hat{T}^2/\theta^3$. Conditioned on both of this happening, the probability that w satisfies the condition on line 16 (that is, that the triangle uvw is assigned to e) is $t^+(e)/t(e)$ in which case we increment $A_{v,1}$ by $\theta^3 t(e)/(\hat{T}^2\varepsilon^2)$. In expectation (conditioned on S and S_t), e contributes to $A_{v,1}$ exactly $p_2^2 \cdot \frac{T^2\varepsilon^2}{\theta^3} \cdot \frac{t^+(e)}{t(e)} \cdot \frac{t(e)\theta^3}{T^2\varepsilon^2} = p_2^2 t^+(e)$. Any edge not in $E_{L,1}$ contributes 0 (conditioned on S, S_t). By the linearity of expectation, it holds that $E(A_{v,1}|S, S_t) = p_2^2 T_{L,1}$. Moreover, $A_{v,1}$ is clearly independent of N and it thus also holds $E(A_{v,1}|S, S_t, N) = p_2^2 T_{L,1}$.

We now analyze $E(A_{v,2}|S, S_t, N)$. For any edge $e \in E_{L,2}$, the probability that it is in the induced subgraph $G[S_v]$ is p_2^2. Conditioned on this happening, the probability that w satisfies the condition on line 20 (that is, that the triangle uvw is assigned to e) is $F_{N,uv}$ in which case we increment $A_{v,2}$ by $t(e)$. In expectation (conditioned on S, S_t, N), e contributes to $A_{v,2}$ exactly $C_e = p^2 F_{N,uv} t(e)$. By Lemma 49, we have with probability at least $1 - 3/n$ that for all edges e, $C_e = (1 \pm \varepsilon/3)p_2^2 t^+(e)$. By the linearity of expectation, it then holds that $E(A_{v,2}|S, S_t, N) = (1 \pm \varepsilon/3)p_2^2 T_{L,2}$.

Putting this together, we have $E(A_v|S, S_t, N) = p_2^2 T_{L,1} + (1 \pm \varepsilon/3)p_2^2 T_{L,2} = (1 \pm \varepsilon/3)p_2^2 T_L$ and thus $E(\hat{T}_L|S, S_t, N) = E(A_v|S, S_t, N)/p_2^2 = (1 \pm \varepsilon/3)T_L$.

It holds $E(C_e|S, S_t) = E(F_{N,e}|S, S_t)t(e) = t^+(e)$. It thus holds $E(A_{v,2}|S, S_t) = p_2^2 T_{L,2}$. Putting this together, we have $E(A_v|S, S_t) = p_2^2 T_{L,1} + p_2^2 T_{L,2} = p_2^2 T_L$ and thus $E(\hat{T}_L|S, S_t) = T_L$. □

[19] In fact, it is sufficient to consider the probability over S_t, N but we will not need this.

Lemma 51. *It holds with high probability that* $Var(\hat{T}_L|S, S_t, N) = \frac{1}{8}(\varepsilon T)^2$. *(where the high probability is over the choice of* S, S_t, N*).*

Proof. We now analyze the variance of $A_v = A_{v,1} + A_{v,2}$, conditional on S, S_t, N and assuming $F_{N,e} = (1 \pm \varepsilon/3)\frac{t^+(e)}{t(e)}$ (and thus also $C_e = (1 \pm \varepsilon/3)p_2^2 t^+(e)$); recall that this holds for all edges e with probability at least $1 - 3/n$. We analyze the variance using the law of total variance, conditioning on S_v. We start by upper-bounding the variance of the conditional expectation.

Let $X_{uv} = F_{N,uv}t(uv)$ if $u, v \in S_v$ and $u, v \in V_L$, and let $X_{uv} = 0$ otherwise. If $t^+_{S_t}(uv) < 162 \log n/\varepsilon^2$, then it holds $X_{uv} = t(uv)P(uv \prec uw \wedge uv \prec vw) = \theta^3/(\varepsilon^2\tilde{T}^2)t(uv)P(B = 1)P(uv \prec uw \wedge uv \prec vw)$ for $w \sim N(u) \cap N(v)$. For $t^+_{S_t}(uv) \geq 162 \log n/\varepsilon^2$, it holds $X_{uv} = t(uv)P(uv \prec uw \wedge uv \prec vw)$ for $w \sim N \cap N(u) \cap N(v)$ (note the different distribution of w). We then have

$$E(A_v|S, S_v, S_t, N) = \sum_{\substack{u,v \in S_v \\ u,v \in V_L \\ t^+_{S_t}(e) < 162 \log n/\varepsilon^2}} t(uv)P_{w \sim N(u) \cap N(v)}(uv \prec uw \wedge uv \prec vw) + \tag{3.66}$$

$$\sum_{\substack{u,v \in S_v \\ u,v \in V_L \\ t^+_{S_t}(e) \geq 162 \log n/\varepsilon^2}} t(uv)P_{w \sim N \cap N(u) \cap N(v)}(uv \prec uw \wedge uv \prec vw) = \sum_{e \in E_L} X_e \tag{3.67}$$

Calculating the conditional variance of this expectation, we get

$$Var(E(A_v|S, S_v, S_t, N)|S, S_t, N) = Var(\sum_{e \in E_L} X_e|S, S_t, N) \tag{3.68}$$

$$\leq \sum_{e \in E_L} Var(X_e|S, S_t, N) + \sum_{\substack{e_1,e_2 \in E_L \\ e_1 \cap e_2 \neq \emptyset}} E(X_{e_1}X_{e_2}|S, S_t, N) \tag{3.69}$$

where we are using that X_{e_1} and X_{e_2} are independent when $e_1 \cap e_2 = \emptyset$, conditionally on S_t, S, N. We can further bound

$$\sum_{e \in E_L} Var(X_e|S, S_t, N) \leq p_2^2 \sum_{e \in E_L} ((1 + \varepsilon/3)t^+(e))^2 \leq 92p_2^2 T^{4/3}$$

where the first inequality holds because $Var(X_e|S_t, S, N) \leq E(X_e^2|S_t, S, N) \leq p_2^2((1+\varepsilon/3)t^+(e))^2$ and the second inequality holds by Lemma 43 and because

$(1+\varepsilon/3)^2 < 2$ because $\varepsilon < 1$. We now bound

$$\sum_{\substack{e_1,e_2\in E_L\\ e_1\cap e_2\neq\emptyset}} E(X_{e_1}X_{e_2}|S_t,S,N) = p_2^3 \sum_{\substack{e_1,e_2\in E_L\\ e_1\cap e_2\neq\emptyset}} F_{N,e_1}F_{N,e_2}t(e_1)t(e_2) \tag{3.70}$$

$$\leq 2p_2^3 \sum_{\substack{e_1,e_2\in E_L\\ e_1\cap e_2\neq\emptyset}} \frac{t^+(e_1)}{t(e_1)}\cdot\frac{t^+(e_2)}{t(e_2)}\cdot t(e_1)t(e_2) \tag{3.71}$$

$$= 2p_2^3 \sum_{e_1\in E_L} t^+(e_1) \sum_{e_2\in N(e_1)} t^+(e_2) \tag{3.72}$$

$$\overset{\text{w.h.p.}}{\leq} 2p_2^3\theta \sum_{e\in E_L} t^+(e) = 2p_2^3\theta T_L \tag{3.73}$$

$$\tag{3.74}$$

where the second inequality holds with probability at least $1-1/n^2$ by Lemma 46 since we are assuming both endpoints of e are light. The first holds for the following reason. As we already mentioned, we are assuming that for N, it holds $F_{N,e} = (1\pm\varepsilon/3)\frac{t^+(e)}{t(e)}$ for any edge e (this happens with probability at least $1-O(1/n)$). We thus have

$$F_{N,e_1}F_{N,e_2} \leq (1+\varepsilon/3)^2\frac{t^+(e_1)}{t(e_1)}\cdot\frac{t^+(e_2)}{t(e_2)} \leq 2\frac{t^+(e_1)}{t(e_1)}\cdot\frac{t^+(e_2)}{t(e_2)}$$

Together, this gives us a bound

$$Var(E(A_v|S,S_v,S_t,N)|S,S_t,N) \leq 92p_2^2T^{4/3} + 2p_2^3\theta T$$

We now bound the expectation of variance. Let Y_e be the increment to A_v contributed by an edge $e\in E(G[S_v])$. It holds that

$$Var(A_v|S,S_v,S_t,N) = \sum_{e\in E(G[S_v])} Var(Y_e|S,S_t,N) \tag{3.75}$$

$$= \sum_{\substack{e\in E(G[S_v])\\ t^+_{S_t}(e)<162\log n}} Var(Y_e|S,S_t,N) + \sum_{\substack{e\in E(G[S_v])\\ t^+_{S_t}(e)\geq 162\log n}} Var(Y_e|S,S_t,N) \tag{3.76}$$

$$\leq \sum_{\substack{e\in E(G[S_v])\\ t^+_{S_t}(e)<162\log n}} \frac{\theta^3}{\varepsilon^2\tilde{T}^2}\cdot t^+(e)t(e) + \sum_{\substack{e\in E(G[S_v])\\ t^+_{S_t}(e)\geq 162\log n}} F_{N,e}t(e)^2 \tag{3.77}$$

$$\overset{\text{w.h.p.}}{\leq} 205\,\frac{\theta^2\log n}{\varepsilon^2\tilde{T}} \sum_{\substack{e\in E(G[S_v])\\ t^+_{S_t}(e)<162\log n}} t(e) + (1+\varepsilon/3) \sum_{\substack{e\in E(G[S_v])\\ t^+_{S_t}(e)\geq 162\log n}} t^+(e)t(e) \tag{3.78}$$

where the first inequality holds because $Y_e = t(e)\theta^3/(\varepsilon^2\tilde{T})$ with probability $t^+(e)/t(e)\cdot\varepsilon^2\tilde{T}^2/\theta^3$ and $Y_e = 0$ otherwise. The second inequality holds because by the Chernoff and the union bound, it holds that, with probability at least $1 - 1/n$. for any edge e such that $t^+_{S_t}(e) < 162\log n$, it holds that $t^+(e) < 205\tilde{T}\log n/\theta$. It, therefore, holds by the linearity of expectation that

$$E(Var(A_v|S, S_t, S_v, N)S, S_t, N) \leq 205\,\frac{\theta^2\log n}{\varepsilon^2\tilde{T}} \sum_{\substack{e\in E\\ t^+_{S_t}(e)<162\log n}} p_2^2 t(e) + (1+\varepsilon/3) \sum_{\substack{e\in E\\ t^+_{S_t}(e)\geq 162\log n}} p_2^2 t^+(e)t(e) \tag{3.79}$$

$$\leq 615\,\frac{p_2^2\theta^2 T\log n}{\varepsilon^2\tilde{T}} + 62p_2^2T^{4/3}$$

By the law of total variance, it holds

$$Var(A_v|S, S_t, N) = E(Var(A_v|S, S_t, S_v, N)|S, S_t, N) + Var(E(A_v|S, S_t, S_v, N)|S, S_t, N) \tag{3.80}$$

$$\leq 615\,\frac{p_2^2\theta^2 T\log n}{\varepsilon^2\tilde{T}} + 62p_2^2T^{4/3} + 92p_2^2T^{4/3} + 2p_2^3\theta T \tag{3.81}$$

$$\leq \frac{1}{16}(\varepsilon p^2 T)^2 + \frac{1}{160}(\varepsilon p^2 T)^2 + \frac{1}{100}(\varepsilon p^2 T)^2 + \frac{1}{50}(\varepsilon p^2 T)^2 < \frac{1}{8}(\varepsilon p_2^2 T)^2 \tag{3.82}$$

where the second inequality holds because of the way we set p_2, θ and because $T < m^3/n^3$. We thus have $Var(\hat{T}_L|S) = Var(A_v|S)/p_2^4 = \frac{1}{8}(\varepsilon T)^2$. □

Lemma 52. *It holds $E(\hat{T}_H|S, S_t, N) = T_H$ and $E(Var(T_H|S, S_t, N)|S_t, N) \leq \frac{1}{12}(\varepsilon^2T^2)^2$*

Proof. We now focus on A_e. Let $\Delta_i A_e$ be the i-th increment of A_e. It holds

$$E(\Delta_i A_e|S, S_t, N) = \frac{1}{m}\sum_{e, e\cap V_H\neq\emptyset} \frac{t^+(e)}{t(e)}t(e) = \frac{1}{m}\sum_{e, e\cap V_H\neq\emptyset} t^+(e) = T_H/m$$

Therefore, $E(A_e|S, S_t, N) = E(\sum_{i=1}^k \Delta_i A_e|S, S_t, N) = kT_H/m$ and thus $E(\hat{T}_H|S, S_t, N) = T_H$. Let Y be the number of heavy vertices with respect to S. We have

$$Var(\Delta_i A_e|S, S_t, N) \leq \frac{1}{m}\sum_{\substack{e\in E\\ e\cap V_H\neq\emptyset}} \frac{t^+(e)}{t(e)}t(e)^2 = \sum_{\substack{e\in E\\ e\cap V_H\neq\emptyset}} t^+(e)t(e) \leq 6(2+\log n)YT/m$$

where the last inequality holds by Lemma 48. We thus have $Var(T_H|S) = m^2Var(\Delta_1 A_e|S)/k \leq 6m(2+\log n)YT/k$. It thus holds by Lemma 47 that

$$E(Var(T_H|S, S_t, N)|S_t, N) \leq \frac{36(2+\log n)mT^2}{\theta k} = \frac{1}{12}(\varepsilon^2T^2) \tag{3.83}$$

where the equality holds by our choice of k. □

We are now in position to state and prove the main lemma.

Lemma 53. *Algorithm 22 returns an unbiased estimate $\hat{T}$ of T and has expected query complexity of $O(\min(\frac{m}{\varepsilon^2\tilde{T}^{2/3}}, \frac{\sqrt{nm}\log n}{\varepsilon^2\sqrt{\tilde{T}}}))$. Moreover, if $\tilde{T} \leq T$, then with probability at least $2/3$, it holds $\hat{T} = (1 \pm \varepsilon)T$.*

Proof. We first argue the query complexity. We then argue the last part of the lemma, namely the deviation bounds on $\hat{T}$. Finally, we then prove that the returned estimate is unbiased.

We now argue the complexity of the algorithm. In the case $\tilde{T} \geq m^3/(n^3 \log^6 n)$, the dominant branch of the min is $m/(\varepsilon^2\tilde{T}^{2/3})$. In this case, we execute Algorithm 21 and by Lemma 44, the complexity is $O(\frac{m}{\varepsilon^2\tilde{T}^{2/3}})$ as desired.

We now consider the case $\tilde{T} < m^3/(n^3 \log^6 n)$. On line 4, we perform in expectation $np_1 = O(n \log n/\sqrt{\theta}) \subseteq O(\frac{\sqrt{nm}\log n}{\sqrt{\tilde{T}}})$ queries. On lines 5 to 7, we perform in expectation $O(n\theta \log n/(\varepsilon^2\tilde{T})) = O(\frac{\sqrt{nm}\log n}{\varepsilon^2\sqrt{\tilde{T}}})$ queries. We now calculate the expected number of queries performed on line 15. Each of the m edges is in $G[S_v]$ with probability $p_2^2 = \Theta(\theta^2/(\varepsilon^4\tilde{T}^2))$. For each such edge, we make a query only when $B = 1$, which happens with probability $\varepsilon^2\tilde{T}^2/\theta^3$. This gives us expectation of (up to a constant factor)

$$m \cdot \theta^2/(\varepsilon^4\tilde{T}^2) \cdot \varepsilon^2\tilde{T}^2/\theta^3 = \frac{m}{\varepsilon^2\theta} = \frac{\sqrt{nm}}{\varepsilon^2\sqrt{\tilde{T}}}$$

On line line 19, we only access vertices that have previously been queried, as $w \in N$. This therefore does not increase the query complexity. On line 24, we sample $k = O(\frac{m \log n}{\varepsilon^2\theta}) = O(\frac{\sqrt{nm}}{\varepsilon^2\sqrt{T}})$ edges. On line 27, we sample at most one vertex for each of these sampled edges, thus not increasing the asymptotic complexity. Putting all the query complexities together, the total query complexity is as claimed.

We now prove the deviation bounds. We condition on the high-probability events from Lemmas 50 and 51. We do not make this conditioning explicit in the rest of the proof. We have by Lemma 50 that $E(\hat{T}_L|S, S_t, N) = (1\pm\varepsilon/3)T_L$ and by Lemma 52 that $E(\hat{T}_H|S, S_t, N) = T_H$. Therefore, $E(\hat{T}|S, S_t, N) = E(\hat{T}_L + \hat{T}_H|S, S_t, N) = (1+\varepsilon/3)T$. By Lemma 51, it holds $Var(\hat{T}_L|S, S_t, N) \leq \frac{1}{8}(\varepsilon T)^2$ and by Lemma 52, we have $Var(\hat{T}_H|S, S_t, N) \leq \frac{1}{8}(\varepsilon T)^2$. The random variables $\hat{T}_L, \hat{T}_H$ can be easily seen to be independent conditionally on S. Since $\hat{T}_H$ is independent of S_t and N, it holds that $\hat{T}_L$ and $\hat{T}_H$ are also independent conditionally on S, S_t, N. It thus holds that $Var(\hat{T}|S, S_t, N) = Var(\hat{T}_L|S, S_t, N) + Var(\hat{T}_H|S, S_t, N)$. We now use the law of total variance, conditioning on S, to bound $Var(\hat{T}|S_t, N)$. Since $E(\hat{T}|S, S_t, N) = (1 \pm \varepsilon/3)T$, it holds $Var(E(\hat{T}|S,S_t,N)|S_t,N) \leq (2\varepsilon T/3)^2/4 = 1/9(\varepsilon T)^2$. It then holds

$$Var(\hat{T}|S_t,N) = E(Var(\hat{T}|S,S_t,N)|S_t,N) + Var(E(\hat{T}|S,S_t,N)|S_t,N) \tag{3.84}$$

$$\leq \frac{1}{8}(\varepsilon T)^2 + \frac{1}{12}(\varepsilon T)^2 + \frac{1}{9}(\varepsilon T)^2 \leq \frac{72}{23}(\varepsilon T)^2 \tag{3.85}$$

where the first of the three terms of the bound comes from Lemma 51, the second comes from Lemma 52, and the third from the above bound on $Var(E(\hat{T}|S,S_t,N)|S_t,N)$. Therefore, by the Chebyshev inequality, we have that

$$P(|\hat{T}-T| > \varepsilon T) \leq \frac{Var(\hat{T})}{(\varepsilon T)^2} \leq 72/23$$

Adding probability of $O(1/n)$ of the complement of the events we are conditioning on, we have that $P(|\hat{T}-T| > \varepsilon T) \leq 1/3$ for n large enough.

We now argue unbiasedness. We now argue that the algorithm gives an unbiased estimate of T. It holds

$$E(\hat{T}) = E(E(\hat{T}_L + \hat{T}_H|S)) = T_L + T_H = T$$

□

By standard advice removal (as we discussed in Chapter 3), we get the following theorem.

Theorem 54. *There is an algorithm in the full neighborhood access model with random vertex and edge queries, that returns $\hat{T}$ such that, with probability at least $2/3$, it holds $\hat{T} = (1 \pm \varepsilon)T$ and has expected query complexity of $O(\min(\frac{m}{\varepsilon^2 T^{2/3}}, \frac{\sqrt{nm}\log n}{\varepsilon^2\sqrt{T}}))$.*

We may modify the algorithm so that Algorithm 21 is called on line 2 whenever $\hat{T} \geq m^3/(\varepsilon^3 n^3)$. We simulate the random edge queries using Algorithm 17. The number of random edge queries in the case $\hat{T} \geq m^3/(\varepsilon^3 n^3)$ is then $s = m/(\varepsilon^2 T^{2/3})$ and the complexity is thus $O(n\sqrt{s/m} + s) = O(\frac{n}{\varepsilon T^{1/3}})$. In the case $\hat{T} < m^3/(\varepsilon^3 n^3)$, we make $s = O(\frac{\sqrt{nm}\log n}{\varepsilon^2\sqrt{T}})$ queries and the complexity is thus $O(n\sqrt{s/m} + s) = O(\frac{\sqrt{nm}\log n}{\varepsilon^2\sqrt{T}})$. This gives total complexity of $O(\frac{n}{\varepsilon T^{1/3}} + \frac{\sqrt{nm}\log n}{\varepsilon^2\sqrt{T}})$:

Theorem 55. *There is an algorithm in the full neighborhood access model with random vertex queries, that returns $\hat{T}$ such that, with probability at least $2/3$, it holds $\hat{T} = (1 \pm \varepsilon)T$ and has query complexity of $O(\frac{n}{\varepsilon T^{1/3}} + \frac{\sqrt{nm}\log n}{\varepsilon^2\sqrt{T}}))$.*

Lower Bound

Proving $\Omega(m/T^{2/3})$ and $\Omega(n/T^{1/3})$

Theorem 56. *Any algorithm that returns $\hat{T}$ such that $\hat{T} = (1 \pm \varepsilon)T$ with probability at least $2/3$ and uses (1) only random vertex queries and solves, (2) only random edge queries, (3) both random vertex and edge queries has to have query complexity at least (1) $\Omega(n/T^{1/3})$, (2) $\Omega(m/T^{2/3})$, (3) $\Omega(\min(n/T^{1/3}, m/T^{2/3}))$.*

Proof. Given n and m, we construct two graphs with $\Theta(n)$ vertices and $\Theta(m)$ edges with the same number of vertices and edges. We than prove that the two graphs are hard to distinguish using $o(n/T^{1/3})$ random vertex samples and $o(m/T^{2/3})$ random edges. The three lower bounds follow. Let H be a triangle-free graph on n vertices and m edges. We let G_1 be a disjoint union of H with a clique of size $k = \Theta(T^{1/3})$ that has at least T triangles and ℓ edges. Let G_2 be the disjoint union of H with a bipartite graph on k vertices with ℓ edges. This means that G_1 has $\geq T$ triangle while G_2 is triangle-free.

Consider sampling a pair of vertices $v_1 \in G_1, v_2 \in G_2$ as follows. With probability $n/(n+k)$, we sample a vertex v uniformly from H and set $v_1 = v_2 = v$. Otherwise, we sample v_1 independently uniformly from $G_1 \setminus H$ and v_2 from $G_2 \setminus H$. One can easily verify that v_1 is sampled uniformly from G_1 and v_2 from G_2. It holds $P(v_1 \neq v_2) \leq k/(n+k) = \Theta(T^{1/3}/n)$ (the equality holds because $k = \Theta(T^3) \leq O(n)$).

Similarly, we sample $e_1 \in G_1$ and $e_2 \in G_2$ as follows. We sample with probability $m/(m+\ell)$ an edge e uniformly from H and set $e_1 = e_2 = e$. Otherwise, we sample e_1, e_2 independently from $G_1 \setminus H, G_2 \setminus H$, respectively. It holds $P(e_1 \neq e_2) \leq \ell/(n+\ell) = \Theta(T^{2/3}/m)$ (the equality holds because $\ell = \Theta(T^{2/3}) \leq O(m)$). Making $o(n/T^{1/3})$ vertex samples and $o(m/T^{2/3})$ edge samples from the above couplings, it holds by the union bound that, with probability $1 - o(1)$, the samples are equal. This means that any algorithm executed (with the same randomness) on these two samples, has to give the same answer on G_1 and G_2 with probability at least $1 - o(1)$. This is in contradiction with the algorithm being correct with probability $2/3$. □

Proving $\Omega(\sqrt{nm/T})$

The proof is via reduction from the OR problem: given $(x_1, \cdots, x_n) \in \{0,1\}^n$, the OR problem asks for the value $\bigvee_{i=1}^n x_i$. It is known that any algorithm that solves the OR problem with probability at least $2/3$ has complexity $\Omega(n)$. See for example Assadi for a proof of this.

Theorem 57. *Assume $T \leq m^3/n^3$. Any algorithm that returns $\hat{T}$ such that $\hat{T} = (1 \pm \varepsilon)T$ with probability at least $2/3$ and uses both random vertex and edge queries must have query complexity at least $\Omega(\sqrt{nm/T})$.*

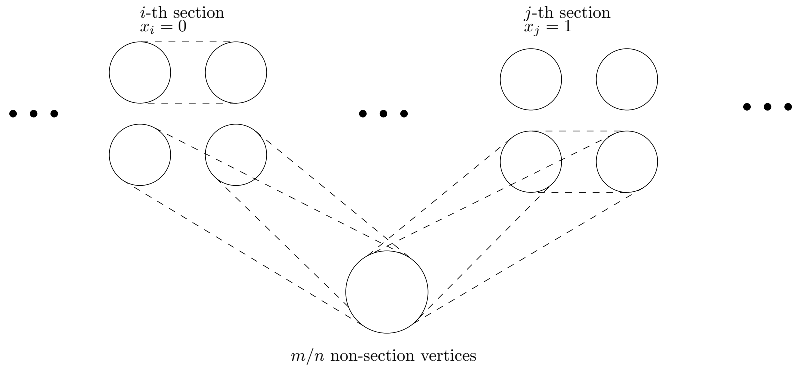

Figure 3.1: The hard instance used in the proof of Theorem 57.

Proof. The proof is by reduction from the OR problem of size $\ell = \sqrt{nm/T}$. Specifically, we show that if we have an algorithm that solves the triangle counting problem in expectation using Q full neighborhood queries, then we can also solve the OR problem of size ℓ in Q queries. This implies the bound as one needs $\Omega(\ell) = \Omega(\sqrt{nm/T})$ queries to solve the OR problem of size ℓ.

Given a vector $x = x_1, \cdots, x_\ell$, we define a graph $G_x = (V, E_x)$. The graph consists of ℓ *sections* $s_1, \cdots s_\ell$ and m/n *non-section vertices*, where each section consists of $4\sqrt{nT/m}$ *section vertices*. Each section is divided into four *subsections*, with each subsection having $\sqrt{nT/m}$ vertices. We call the subsections top-left, top-right, bottom-left, and bottom-right. We order vertices within a section such that vertices in the top-left subsection come first, then top-right, bottom-left, and bottom-right in this order. We order the sections arbitrarily. Together with the orders on the vertices within a section, this induces an order on all section vertices. We put the non-section vertices at the end of this order.

We have specified the vertex set; we now specify the edges of G_x. There is an edge between each right subsection vertex and each non-section vertex. Consider the i-th section in the ordering. If $x_i = 0$, then there is an edge between each top-left vertex and top-right vertex. That is, for $x_i = 0$, the top subsections induce a complete bipartite graph and the bottom subsections induce an independent set. If $x_i = 1$, the top subsections induce an independent set and there is an edge between each bottom-left and bottom-right vertex. We represent each edge e consisting of vertices u and v as a pair (u, v) such that u comes in the order before v. We order edges contained within a section lexicographically, with edges incident to the non-section vertices coming at the end.

In the rest of the proof, we argue that (1) the graph G_x has $\Theta(n)$ vertices, $\Theta(m)$ edges (2) $\bigvee_{i=1}^{\ell} x_i = 0$ iff G_x is triangle-free and $\bigvee_{i=1}^{\ell} x_i = 1$ iff G_x has at least T triangles, and (3) that we can simulate a query on G_x by a single query on x. Proving these three things means that any algorithm that solves the approximate triangle counting problem can be used to give an algorithm for the OR problem of size ℓ with the same query complexity. This then implies the lower bound.

The number of vertices is $m/n+\sqrt{\frac{nm}{T}}\cdot 4\sqrt{nT/m}=\Theta(n)$. The number of edges is $m/n\cdot\sqrt{\frac{nm}{T}}\cdot\sqrt{nT/m}+\sqrt{\frac{nm}{T}}\cdot nT/m=\Theta(m)+\frac{n^{3/2}T^{1/2}}{m^{1/2}}\leq\Theta(m)$ where the last inequality is true by assumption $T\leq m^3/n^3$. This proves property (1).

Consider the case $\bigvee_{i=1}^{\ell} x_i=0$. Each section induces a disjoint union of a complete bipartite graph and an independent set. Each section only neighbors the non-section vertices. Moreover, the non-section vertices induce an independent set. Thus, any triangle has to contain one non-section vertex and two section vertices. Since the non-section vertices only neighbor the bottom subsections, these two section vertices have to be in the bottom subsections. If $x_i=0$, there are no edges between the bottom subsections of the i-th section. Therefore, these two vertices cannot be from the i-th section. If $x_i=0$ for all $i\in[\ell]$, there cannot be any triangles in the graph.

If $\bigvee_{i=1}^{\ell} x_i=1$, there exists i such that $x_i=1$. Consider the i-th section. There is a triangle for each triplet of vertices u,v,w for u in the bottom-left subsection, v being in the bottom-right subsection and w being a non-section vertex from the same unit. This means that the graph contains at least $(\sqrt{nT/m})^2\cdot m/n=T$ triangles. This proves property (2).

We finally argue that any full neighborhood query on G can be simulated using a query on x_j for some $j\in[\ell]$. Specifically, we prove that, given $i\in[|V|]$ ($i\in[|E_x|]$), the neighborhood of v_i (of v,w for $e_i=vw$) only depends on the value of x_j for some j (respectively). The neighborhood of any non-section vertex is the same regardless of x (and the query can thus be answered without making any queries on x). The neighborhood of a vertex in the j-th section is determined by the value of x_i. Vertex queries can thus be simulated by a single query on x. We now argue that this is also the case for edge queries. The edges are ordered such that for $j<j'$, all edges incident to the j-th section are in the ordering before those incident to the j'-th section. The number of edges within a section is independent of x. We call this number k. The i-th edge is then the $(i-k\lfloor(i-1)/k\rfloor)$-th vertex in the $(\lfloor(i-1)/k\rfloor+1)$-th section and it thus only depends on $x_{\lfloor(i-1)/k\rfloor+1}$. This proves the property (3).

□

Chapter 4

Sampling an Edge in Sublinear Time Exactly and Optimally

Talya Eden[1]
talyaa01@gmail.com
Bar Ilan University

Shyam Narayanan[2]
shyamsn@mit.edu
MIT

Jakub Tětek
j.tetek@gmail.com
BARC, Univ. of Copenhagen

Abstract

Sampling edges from a graph in sublinear time is a fundamental problem and a powerful subroutine for designing sublinear-time algorithms. Suppose we have access to the vertices of the graph and know a constant-factor approximation to the number of edges. An algorithm for pointwise ε-approximate edge sampling with complexity $O(n/\sqrt{\varepsilon m})$ has been given by Eden and Rosenbaum [SOSA 2018]. This has been later improved by Tětek and Thorup [STOC 2022] to $O(n\log(\varepsilon^{-1})/\sqrt{m})$. At the same time, $\Omega(n/\sqrt{m})$ time is necessary. We close the problem, by giving an algorithm with complexity $O(n/\sqrt{m})$ for the task of sampling an edge exactly uniformly.

Introduction

Suppose we have a graph too big to even read the whole input. We then need an algorithm running in time sublinear in the input size. Such algorithms have recently received a lot of attention. In the sublinear-time settings, one usually has direct access to the vertices of the input graph, but not to the edges. Because of this, one tool commonly used for designing sublinear-time graph algorithms is an algorithm for sampling edges. This allows us to design an algorithm that uses random edge queries, as we can simulate these queries by the edge sampling algorithm.

The task and query access. Our goal is to sample an edge uniformly, i.e., to return an edge so that each edge is returned with exactly equal probability. We assume that the algorithm may (i) ask for the i-th vertex of the input graph, (ii) ask for the degree of a given vertex, and (iii) ask for the j-th neighbor of a given vertex. We assume the algorithm has (approximate) knowledge of the number of edges m. This assumption of knowing m was not made in the previous work, and we think getting rid of this assumption is a very interesting open problem. This assumption is, however, not a barrier to using our algorithm as a subroutine for implementing random edge queries, as we discuss below.

Ours and previous results. In past work only algorithms for sampling ε-*approximately* uniformly in the pointwise distance (or equivalently approximately in the ℓ_∞ metric) were given, where the state-of-the-art complexity for that problem was $O(n \log(\varepsilon^{-1})/\sqrt{m})$ given by Tětek and Thorup [2022b]. Our algorithm is not only exact, but also more efficient. Specifically, the expected complexity of our algorithm is $O(n/\sqrt{m})$. This is known to be the best possible. If we have a graph with $n - \Theta(\sqrt{m})$ isolated vertices and a clique over $\Theta(\sqrt{m})$ vertices with m edges, we need to sample $\Omega(n/\sqrt{m})$ vertices before we expect to see a single edge, giving us a simple matching lower bound.

Using our algorithm as a subroutine and the necessity of knowing m. As we said above, our algorithm needs to have a constant-factor approximation of the number of edges. This was not necessary in previous works. The reason is that the previous state-of-the-art has complexity in which one can also afford to independently estimate the number of edges; the possibility of the estimate being incorrect is then added to the bias of the edge sampling algorithm. However, as our algorithm is more efficient, we are not able to estimate m in that complexity, and since we want to sample exactly, we cannot accept that the estimate could be wrong.

Suppose we have an algorithm A that performs random edge queries. We may then use our algorithm in a black box manner to implement these queries (unlike, for example, the algorithm for sampling multiple edges from Eden et al. [2021b] which has polynomial dependency on ε). Specifically, if A uses q random edge queries, then we may set $\varepsilon = 1/(10q)$ and it will only decrease the success probability of A by at most $1/10$. [3]

If the goal is to get an algorithm with a constant success probability (which can then be amplified) that uses our edge sampling algorithm as a subroutine, then we may remove the need for having an a priori constant-factor approximation $\tilde{m}$ of m by computing it using the algorithm from Feige [2006b], only adding a constant to the failure probability. The algorithm from Feige [2006b] has expected complexity $O(n/\sqrt{m})$. The complexity of our algorithm will also still be as desired: it follows from our analysis that the complexity is $O(\frac{n\sqrt{\tilde{m}}}{m})$. It holds by the Jensen inequality that $\mathbb{E}[\frac{n\sqrt{\tilde{m}}}{m}] \leq \frac{n\sqrt{\mathbb{E}[\tilde{m}]}}{m} + \log \varepsilon^{-1} = O(n/\sqrt{m})$ since it holds $\mathbb{E}[\tilde{m}] = O(m)$.

To summarize, we may remove the assumption of a priori knowledge of m when sampling multiple edges at the cost of adding a constant failure probability. This means that we may use our algorithm as a subroutine in an algorithm with constant probability of error, even without knowing m a priori.

Technical overview

The starting point of our algorithm is the algorithm by Eden and Rosenbaum [2018d], which we now shortly recall.

The algorithm by Eden and Rosenbaum [2018d].

Consider each undirected edge as two directed edges, and let θ be a degree threshold. We refer to vertices with degree at most θ as *light* vertices, and to all other vertices as *heavy*. We refer to edges originating in light vertices as *light edges*, and to all other edges as *heavy edges*. Using rejection sampling, light edges can be sampled with probability *exactly* $\frac{1}{n\theta}$: by sampling a uniform vertex v, then sampling one of its incident edges u.a.r., and then returning that edge with probability $\frac{d(v)}{\theta}$. Sampling heavy vertices is done by first sampling a light edge uv as described above, and if the second endpoint v of the sampled light edge is heavy, sampling one of its incident edges. This procedure results in every heavy edge vw being sampled with probability $\frac{d_\ell(v)}{n\theta} \cdot \frac{1}{d(v)}$, where $d_\ell(v)$ is the number of light neighbors of v. In (Eden and Rosenbaum [2018d]), θ is set to $\sqrt{2m/\varepsilon}$ which implies that for every heavy vertex v, $d_\ell(v) \in$

[3] This holds because the total variation distance from uniform of each query is at most $1/(10q)$, so the total variation distance from uniform of the sequence of q queries is at most $1/10$, meaning that the output from the algorithm has total variation distance at most $1/10$ from the distribution the output would have if the queries were answered exactly.

$[(1-\varepsilon)d(v), d(v)]$.[4] Hence, each (heavy) edge is sampled with probability in $[\frac{(1-\varepsilon)}{n\theta}, \frac{1}{n\theta}]$. The total probability of sampling some edge (with the algorithm failing otherwise) is thus at least $\frac{(1-\varepsilon)m}{n\theta} \approx \frac{\sqrt{\varepsilon m}}{n}$. Therefore, the number of attempts needed before we expect to sample an edge is $O(\frac{n}{\sqrt{\varepsilon m}})$, implying a multiplicative dependence on ε.

Improving the dependency on ε.

In order to avoid the multiplicative dependency in ε, we instead set the threshold θ to $\sqrt{cm}$ for some constant c. Considering the same sampling procedures as before, light edges can still be sampled with probability exactly $\frac{1}{n\theta}$. For heavy edges, however, the values $d_\ell(v)/d(v)$ can vary up to a constant factor between the different heavy vertices, leading to a large bias towards heavy edges originating in vertices v with higher values of $\frac{d_\ell(v)}{d(v)}$. If for each vertex v, we knew the value of $d_\ell(v)$, we could use rejection sampling with probability q that is inversely proportional to $p = \frac{d_\ell(v)}{d(v)}$, e.g., $q = \frac{d(v)}{2d_{\ell(v)}} = \frac{1}{2p}$ (we may assume that, say, $p \geq 2/3$ and thus $q < 1$, by making c large enough). This would result in each heavy edge being sampled with exactly equal probability $\frac{d_{\ell(v)}}{n\theta} \cdot \frac{d(v)}{2d_\ell(v)} = \frac{1}{2n\theta}$.

While we do not know the exact value of $d_\ell(v)$, we can approximate it up to a $(1 \pm \Theta(\varepsilon))$-multiplicative factor using $O(1/\varepsilon^2)$ neighbor queries. This results in $(1 \pm \Theta(\varepsilon))$-approximation of q and thus leads to a distribution that is ε-close to uniform. Note that we only need to approximate q when the algorithm samples a heavy edge. Moreover, when we do that, we return the edge with constant probability. Thus, in expectation, we only need to approximate q a constant number of times. This means that the total expected time complexity is $O(n/\sqrt{m} + 1/\varepsilon^2)$.

To remove the dependence on $1/\varepsilon^2$, our main observation is that we do not actually need to (approximately) learn the value of p, in order to reject with probability proportional to $q = 1/(2p)$. Rather, we can "simulate" a Bernoulli trial with probability exactly $\frac{1}{2p}$ by using the results of only $O(1)$ many $Bern(p)$ trials in expectation (though possibly more in the worst case), using the Bernoulli Factory technique of Nacu and Peres [2006].

When we sample a uniform neighbor of a heavy vertex v, we see a light neighbor of v with probability exactly $p = \frac{d_\ell(v)}{d(v)}$, where, as discussed above, we can set θ so that $p > 2/3$. Therefore, we have access to a Bernoulli trial that succeeds with probability $Bern(p)$ for $p > 2/3$. As previously explained, in order to achieve uniformity, we need to perform rejection sampling (corresponding to a Bernoulli trial) that succeeds with probability $= \frac{1}{2p}$. We can

[4] Since, denoting by $\mathcal{H}$ the set of heavy vertices, and by $d_h(v)$ the number of heavy neighbors of vertex v, we have the following. For every $v \in \mathcal{H}$, $d_h(v) \leq |H| \leq \frac{2m}{\theta} = \sqrt{2\varepsilon m} = \varepsilon\theta \leq \varepsilon d(v)$. Therefore, $d_\ell(v) > (1-\varepsilon)d(v)$.

simulate $Bern(1/(2p))$ by relying on the results of an expected $O(1)$ independent copies of $Bern(p)$. Namely, we perform an expected $O(1)$ neighbor queries where each results in a light neighbor with probability p, giving us an independent copy of a random variable distributed as $Bern(p)$. If we knew that p is bounded away from 1, we could directly use the result of Nacu and Peres [2006]. We can ensure that this is the case by first rejecting with probability, say, 1/2. The probability of getting a light neighbor and not rejecting is then $p/2 \in [1/3, 2/3]$. We then use the result of Nacu and Peres [2006] with the function $1/(4x)$, thus simulating $Bern(1/(4p/2)) = Bern(1/(2p))$.

Related work

Using uniform edge samples as a basic query in the sublinear time setting was first suggested by Aliakbarpour et al. [2018b] in the context of estimating the number of s-stars in a graph, where they showed that this access allows to circumvent lower bounds that hold in the standard adjacency list access. It was later used for the more general tasks of estimating and uniformly sampling arbitrary subgraphs in sublinear time (Assadi et al. [2019a], Fichtenberger et al. [2020b], Biswas et al. [2021a], Tětek and Thorup [2022b]).

As mentioned in the introduction, sampling edges from a distribution that is pointwise close to uniform in sublinear time was first suggested by Eden and Rosenbaum [2018d] who gave an algorithm with complexity $O\left(n/\sqrt{\varepsilon m}\right)$. This was later improved by Tětek and Thorup [2022b] to an algorithm with complexity $O(n \log(\varepsilon^{-1})/\sqrt{m})$. They also considered two additional access models (full neighborhood access and hash-ordered access) and gave new lower and upper bounds for these settings. Eden et al. [2021b] gave an upper bound for the problem of sampling k edges from a pointwise close to uniform distribution. The complexity of their algorithm is $O\left(\sqrt{k} \cdot \frac{n}{\sqrt{m}} \cdot \frac{\log^2 n}{\varepsilon^{2.5}} + k\right)$. This was later shown to be essentially optimal (i.e., up to the dependencies on ε and $\log n$) by Tětek and Thorup [2022b]. Eden et al. [2019b] gave an $O\left(\frac{n\alpha}{m} \cdot \frac{\log^3 n}{\varepsilon}\right)$ algorithm for sampling edges in graphs with arboricity at most α. They also showed their algorithm is optimal up to the $\text{poly}(\log n, \varepsilon^{-1})$ dependencies.

The task of sampling close to uniform edges was also recently considered in the setting where the access to the graph is given via Bipartite Independent Set (BIS) queries (Lapinskas et al. [2019], Bhattacharya et al. [2022], Addanki et al. [2022]).

Preliminaries

The query model: Since we do not have time to read the whole input (and thus to change its representation), it matters how exactly we are able to query it. Throughout this paper, we assume that the graphs' vertices are labeled arbitrarily in $[n]$, the edges of every vertex v are labeled arbitrarily by $[d(v)]$, and that the algorithm knows n. We then assume the following standard set of queries:

- **Uniform vertex queries:** given $i \in [n]$, return the i-th vertex
- **Degree queries:** given a vertex v, return its degree $d(v)$
- **Neighbor queries:** given a vertex v and $j \in [d(v)]$, return the j-th neighbor of v

This setting has been previously called the adjacency list or indexed neighbor access model and is among the most-studied settings for sublinear-time algorithms.

A model with an additional query

- **Uniform edge queries:** given $i \in [m]$, return vertices u, v such that uv is the i-th edge of the graph

has also been considered. Our algorithm can be thought of as a reduction between the two models. One can show (by randomly permuting the edges) that up to a logarithmic factor this setting is equivalent to just assuming random edge queries (with replacement). Our algorithm then allows us to simulate random edge queries in the indexed neighborhood access model.

Sampling an edge

In this section, we give our algorithm for sampling an edge. We start by stating a result of Nacu and Peres [2006] about "Bernoulli factories". We recall that a function f from a closed interval I to $\mathbb{R}$ is said to be *real analytic* if for any point x_0 in the interior of I, $f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} \cdot (x - x_0)^n$ where $f^{(n)}(x_0)$ represents the nth derivative of f at x_0. Equivalently, f matches its Taylor series about x_0 for all of I.

Theorem 58. *Let $I \subset (0, 1)$ be a closed interval and $f : I \to (0, 1)$ be a real analytic function. Let p be a number in I. Then, there exists an algorithm independent of p that performs in expectation $O(1)$ independent trials from $Bern(p)$ and returns one Bernoulli trial with distribution $Bern(f(p))$. In addition, the probability of using more than k independent trials is at most $C\rho^k$, for some constants $C \geq 1, 0 < \rho < 1$ that only depend on I.*

Theorem 58 gives us the following corollary.

Corollary 59. *There is an algorithm that for any $p \in [2/3, 1]$ performs in expectation $O(1)$ trials from $Bern(p)$ and returns one Bernoulli trial distributed as $Bern(1/(2p))$. (Note that the algorithm may also use its own randomness, independent of the $Bern(p)$'s given.)*

Algorithm 23: Sample an edge pointwise $\Theta(1)$-close to uniform

```
1  $u \leftarrow$ uniformly random vertex
2  $j \leftarrow Unif([\theta])$, where $\theta = \lceil\sqrt{6m}\rceil$.
3  Fail if $d(v) > \theta$ or $d(v) \leq j$
4  $v \leftarrow j$-th neighbor of $u$
5  $B \sim Bern(1/3)$
6  if $B = 1$ then
7      return $uv$
8  else if $B = 0$ and $v$ is heavy then
9      $w \leftarrow$ random neighbor of $v$
10     return $vw$
11 return Fail
```

Proof. First, note that for any p we can simulate $Bern(p/2)$ from a single $Bern(p)$, by simulating an independent $Bern(1/2)$ and considering the event where both $Bern(p)$ and $Bern(1/2)$ equal 1. Then, it is well-known that $\frac{1}{4x}$ is real analytic on the interval $[1/2, 2/3]$, and $\frac{1}{4x}$ is contained in $[3/8, 1/2]$ for $x \in [1/2, 2/3]$. Since we can generate $Bern(p/2)$, we can therefore apply Theorem 58 to get a trial from $Bern\left(\frac{1}{4(p/2)}\right) = Bern\left(\frac{1}{2p}\right)$. □

We now give an algorithm for sampling an edge. The algorithm closely follows the approach from Eden and Rosenbaum [2018d] but uses the Bernoulli factory of Nacu and Peres [2006] to reduce the sampling error in a significantly more efficient way than in Eden and Rosenbaum [2018d]. We first give an algorithm for $\Theta(1)$-approximate edge sampling.

Throughout this section, we assume for sake of simplicity that we know the number of edges exactly. The analysis of correctness only uses that we have an upper bound, while the analysis of the complexity needs that we have a lower bound up to a constant factor. Putting this together, it is in fact sufficient to have a constant-factor approximation.

Lemma 60. *Let e be the edge returned by Algorithm 23 if successful. Then for any light edge e', it holds that $\mathbb{P}(e = e') = 1/(3n\theta)$, and for any heavy edge, it holds that $\mathbb{P}(e = e') = \frac{2}{3} \cdot \frac{d_\ell(v)}{d(v)} \cdot \frac{1}{n\theta}$.*

Proof. Fix a light edge $e' = uv$. Recall that by definition, uv is light iff $d(u) \leq \theta$ for $\theta = \lceil\sqrt{6m}\rceil$. The edge uv is returned only in the case that (1) u is sampled in Step 1, (2) the chosen index j in Step 2 is the label of v, and (3) $B = 1$ in Step 5. Therefore, $\Pr[e = e'] = \frac{1}{n} \cdot \frac{1}{\lceil\sqrt{6m}\rceil} \cdot \frac{1}{3} = \frac{1}{3n\theta}$.

Now fix a heavy edge $e' = vw$. The edge vw is returned in the event that (1) the sampled vertex u in Step 1 is a light neighbor of v, (2) the chosen index j in Step 2 is the label of v, (3) $B = 0$ in Step 5, and (4) w is the sampled

Algorithm 24: Sample an edge $1 \pm \varepsilon$-pointwise-close to uniform

```
1  repeat
2      $vw \leftarrow$ Algorithm 23
3      if $v$ is light then
4          return $vw$
5      if $v$ is heavy then
6          Let $w_1, \ldots,$ be random neighbors of $v$
7          $Y \leftarrow$ use Corollary 59 on Bernoulli trials defined as
             $B_i = [d(w_i) \leq \sqrt{6m}]$
8          if $Y = 0$ then
9              return $vw$
10 until;
```

neighbor in Step 9. Therefore, if we define $\Gamma_L(v)$ to be the set of light neighbors of v, then $\Pr[e = e'] = \sum_{u \in \Gamma_L(v)} \frac{1}{n} \cdot \frac{1}{\lceil\sqrt{6m}\rceil} \cdot \frac{2}{3} \cdot \frac{1}{d(v)} = \frac{2}{3} \cdot \frac{d_\ell(v)}{d(v)} \cdot \frac{1}{n\theta}$. □

We are now able to give an algorithm for sampling an edge perfectly uniformly. Simply re-running the above algorithm until it succeeds would result in $\Theta(1)$-pointwise close to uniform sampling. The algorithm below differs in that if Algorithm 23 returns a heavy edge (which has some bias), we use rejection sampling based on Bernoulli factories to reduce the bias.

Theorem 61. *Algorithm 24 returns a perfectly uniform edge. Its expected complexity is $O(n/\sqrt{m})$.*

Proof. We start with proving the correctness of the algorithm. By Lemma 60, each invocation of Algorithm 23 returns each light edge with probability $\frac{1}{3n\theta}$, and each heavy edge with probability $\frac{2}{3} \cdot \frac{d_\ell(v)}{d(v)} \cdot \frac{1}{n\theta}$. If Algorithm 23 returns a heavy edge vw, then for every w_i sampled in Step 6 in Algorithm 24, it holds that the indicator of the event $[d(w_i) \leq \lceil\sqrt{6m}\rceil]$ is the result of a Bernoulli trial $Bern(p)$ with $p = \frac{d_\ell(v)}{d(v)}$. Let H denote the set of vertices with degree greater than $\lceil\sqrt{6m}\rceil$. Then $deg_H(v) \leq |H| \leq \frac{2m}{\lceil\sqrt{6m}\rceil} \leq \sqrt{\frac{2}{3}m} \leq \frac{1}{3}d(v)$, where the last is since v is heavy (so $d(v) > \lceil\sqrt{6m}\rceil$). Therefore, $p = \frac{d_\ell(v)}{d(v)} \in [\frac{2}{3}, 1]$. Hence, by Corollary 59, the value Y returned by Algorithm 24 has distribution $Y \sim \mathrm{Bern}(\frac{1}{2p})$. Therefore, in a single iteration of the repeat loop, every fixed heavy edge vw is returned with probability $\frac{2}{3} \cdot \frac{d_\ell(v)}{d(v)} \cdot \frac{1}{n\theta} \cdot \left(\frac{1}{2p}\right) \in \frac{1}{3n\theta}$, as $p = \frac{d_\ell(v)}{d(v)}$.

Therefore, every edge is sampled with probability exactly $\frac{1}{3n}$, so conditioning on some edge being returned, each edge is returned with probability in $\frac{1}{m}$, as claimed.

We turn to analyze the complexity of the algorithm. By the above analysis, every invocation of the loop returns an edge with probability at least $m \cdot \frac{1}{3n\theta} \geq \frac{\sqrt{m}}{10n}$. Also, note that each invocation is independent. Therefore, the expected number of iterations until an edge is returned is $O(n/\sqrt{m})$. Furthermore, each invocation of the loop invokes Algorithm 23 once, and Corollary 59 at most once. Algorithm 23 clearly takes a constant number of queries. If Algorithm 23 returns a heavy edge, then sampling the w_i neighbors in Step 6 takes $O(1)$ queries in expectation. Therefore, the expected number of queries in each loop is constant. Hence, the expected query complexity is $\Theta\left(n/\sqrt{m}\right)$. □

Acknowledgments

We thank Nima Anari and Peter Occil for informing us about the existence of Bernoulli factories in the literature, and Nima for pointing us to the reference Nacu and Peres [2006].

Chapter 5

Better Differentially Private Approximate Histograms and Heavy Hitters using the Misra-Gries Sketch

Christian Janos Lebeda
chle@itu.dk
BARC, IT Univ. of Copenhagen

Jakub Tětek
j.tetek@gmail.com
BARC, Univ. of Copenhagen

Abstract

We consider the problem of computing differentially private approximate histograms and heavy hitters in a stream of elements. In the non-private setting, this is often done using the sketch of Misra and Gries [Science of Computer Programming, 1982]. Chan, Li, Shi, and Xu [PETS 2012] describe a differentially private version of the Misra-Gries sketch, but the amount of noise it adds can be large and scales linearly with the size of the sketch; the more accurate the sketch is, the more noise this approach has to add. We present a better mechanism for releasing a Misra-Gries sketch under (ε, δ)-differential privacy. It adds noise with magnitude independent of the size of the sketch; in fact, the maximum error coming from the noise is the same as the best known in the private non-streaming setting, up to a constant factor. Our mechanism is simple and likely to be practical. We also give a simple post-processing step of the Misra-Gries sketch that does not increase the worst-case error guarantee. It is sufficient to add noise to this new sketch with less than twice the magnitude of the non-streaming setting. This improves on the previous result for ε-differential privacy where the noise scales linearly to the size of the sketch. Finally, we consider a general setting where users can contribute multiple distinct elements. We present a new sketch with maximum error matching the Misra-Gries sketch. For many

parameters in this setting our sketch can be released with less noise under (ε, δ)-differential privacy.

Introduction

Computing the histogram of a dataset is one of the most fundamental tasks in data analysis. At the same time, releasing a histogram may present significant privacy issues. This makes the efficient computation of histograms under privacy constraints a fundamental algorithmic question. Notably, differential privacy has become in recent years the golden standard for privacy, giving formal mathematical privacy guarantees. It would thus be desirable to have an efficient way of (approximately) computing histograms under differential privacy.

Histograms have been investigated thoroughly in the differentially private setting (Dwork et al. [2006b], Ghosh et al. [2012], Geng et al. [2015], Cormode et al. [2012], Korolova et al. [2009], Balcer and Vadhan [2019], Aumüller et al. [2022]). These algorithms start by computing the histogram exactly and they then add noise to ensure privacy. However, in practice, the amount of data is often so large that computing the histogram exactly would be impractical. This is, for example, the case when computing the histogram of high-volume streams such as when monitoring computer networks, online users, financial markets, and similar. In that case, we need an efficient streaming algorithm. Since the streaming algorithm would only compute the histogram approximately, the above-mentioned approach that first computes the exact histogram is infeasible. In practice, non-private approximate histograms are often computed using the Misra-Gries (MG) sketch (Misra and Gries [1982]). The MG sketch of size k returns at most k items and their approximate frequencies $\hat{f}$ such that $\hat{f}(x) \in [f(x) - n/(k+1), f(x)]$ for all elements x where $f(x)$ is the true frequency and n is the length of the stream. This error is known to be optimal (Bose et al. [2003]). In this work, we develop a way of releasing a MG sketch in a differentially private way while adding only a small amount of noise. This allows us to efficiently and accurately compute approximate histograms in the streaming setting while not violating users' privacy. This can then be used to solve the heavy hitters problem in a differentially private way. Our result improves upon the work of Chan et al. [2012] who also show a way of privately releasing the MG sketch, but who need a greater amount of noise; we discuss this below.

In general, the issue with making approximation algorithms differentially private is that although we may be approximating a function with low global sensitivity, the algorithm itself (or rather the function it implements) may have a much larger global sensitivity. This increases the amount of noise required to achieve privacy using standard techniques. We get around this issue by exploiting the structure of the difference between the MG sketches for neighboring inputs. This allows us to prove that the following simple mechanism ensures (ε, δ)-differential privacy: (1) We compute the Misra-Gries sketch, (2) we add to each counter independently noise distributed as Laplace$(1/\varepsilon)$, (3) we add to all counters the same value, also distributed as Laplace$(1/\varepsilon)$, (4) we remove all counters smaller than $1 + 2\ln(3/\delta)/\varepsilon$. Specifically, we show that this algorithm satisfies the following guarantees:

Theorem 73 (simplified). *The above algorithm is (ε, δ)-differentially private, uses $2k$ words of space, and returns a frequency oracle $\hat{f}$ with maximum error of $n/(k+1) + O(\log(1/\delta)/\varepsilon)$ with high probability for δ being sufficiently small.*

A construction for a differentially private Misra-Gries sketch has been given before by Chan et al. [2012]. However, the more accurate they want their sketch to be (and the bigger it is), their approach has to add *more* noise. The reason is that they directly rely on the global ℓ_1-sensitivity of the sketch. Specifically, if the sketch has size k (and thus error $n/(k+1)$ on a stream of n elements), its global sensitivity is k, and they thus have to add noise of magnitude k/ε. Their mechanism ends up with an error of $O\left(k \log(d)/\varepsilon\right)$ for ε-differential privacy with d being the universe size. This can be easily improved to $O\left(k \log(1/\delta)/\varepsilon\right)$ for (ε, δ)-differential privacy with a thresholding technique similar to what we do in step (4) of our algorithm above. This also means that they cannot get more accurate than error $\Theta\left(\sqrt{n \log(1/\delta)/\varepsilon}\right)$, no matter what value of k one chooses. We achieve that the biggest error, as compared to the values from the MG sketch, among all elements is $O(\log(1/\delta)/\varepsilon)$ assuming δ is sufficiently small (we show more detailed bounds including the mean squared errors in Theorem 73). This is the same as the best private solution that starts with an exact histogram (Korolova et al. [2009]). In fact, for any mechanism that outputs at most k heavy hitters there exists input with error at least $n/(k+1)$ in the streaming setting (Bose et al. [2003]) and input with error at least $O(\log(\min(d, 1/\delta))/\varepsilon)$ (Balcer and Vadhan [2019]) under differential privacy. In Section 8 we discuss how to achieve ε-differential privacy with error $n/(k+1) + O(\log(d)/\varepsilon)$. Therefore the error of our mechanisms is asymptotically optimal for approximate and pure differential privacy, respectively. The techniques used in Section 8 could also be used to get approximate differential privacy, but the resulting sketch would not have strong competitiveness guarantees with respect to the non-private Misra-Gries sketch, unlike the sketch that we give in Section 5.

Chan et al. Chan et al. [2012] use their differentially private Misra-Gries sketch as a subroutine for continual observation and combine sketches with an untrusted aggregator. Those settings are not a focus of our work but our algorithm can replace theirs as the subroutine, leading to better results also for those settings. However, the error from noise still increases linearly in the number of merges when the aggregator is untrusted. As a side note, we show that in the case of a trusted aggregator, the approach of Chan et al. [2012]

can handle merge operations without increasing error. While that approach adds significantly more noise than ours if we do not merge, it can with this improvement perform better when the number of merges is very large (at least proportional to the sketch size).

Furthermore, we consider the setting where users contribute multiple elements to the stream. The algorithms discussed above can be adjusted for this setting by scaling the noise linearly proportional to the number of elements per user. This is the optimal scaling for ε-differential privacy or when users can contribute many copies of the same element. However, under (ε, δ)-differential privacy in the non-streaming setting the magnitude of noise scales only with the square root of the number of elements if they are distinct Wilkins et al. [2024]. We give a generalized version of the Misra-Gries sketch that requires less noise for many parameters when users contribute multiple distinct elements. It remains an open problem to achieve optimal error in this setting under (ε, δ)-differential privacy.

Another approach that can be used is to use a randomized frequency oracle to recover heavy hitters. However, it seems hard to do this with the optimal error size. In its most basic form ([Ghazi et al., 2019, Appendix D]), this approach needs noise of magnitude $\Theta(\log(d)/\varepsilon)$, even if we have a sketch with sensitivity 1 (the approach increases the sensitivity to $\log(d)$, necessitating the higher noise magnitude), leading to maximum error at least $\Omega(\log(k)\log(d)/\varepsilon)$. Bassily et al. [2017] show a more involved approach which reduces the maximum error coming from the noise to $\Theta((\log(k) + \log(d))/\varepsilon)$, but at the cost of increasing the error coming from the sketch by a factor of $\log(d)$. This means that even if we had a sketch with error $\Theta(n/k)$ and sensitivity 1, neither of these two approaches would lead to optimal guarantees, unlike the algorithm we give in this paper.

Relation to Böhler and Kerschbaum Böhler and Kerschbaum [2021] Essentially the same result as Theorem 73 has been claimed in (Böhler and Kerschbaum [2021]). However, their approach ignores the discrepancy between the global sensitivity of a function we are approximating and that of the function the algorithm actually computes. Their mechanism adds noise scaled to the sensitivity of the exact histogram which is 1 when a user contributes a single element to the stream. But as shown by Chan et al. [2012] the sensitivity of the Misra-Gries sketch scales linearly with the number of counters in the sketch. The algorithm from Böhler and Kerschbaum [2021] thus does not achieve the claimed privacy parameters. Moreover, it seems unlikely this could be easily fixed – not without doing something along the lines of what we do in this paper.

See https://github.com/JakubTetek/Differentially-Private-Misra-Gries for sample implementations of the algorithms we present in this paper.

Technical overview

Misra-Gries sketch Since our approach depends on the properties of the MG sketch, we describe it here. Readers familiar with the MG sketch may wish to skip this paragraph. We describe the standard version; in Section 5 we use a slight modification, but we do not need that here.

Suppose we receive a sequence of elements from some universe. At any time, we will be storing at most k of these elements. Each stored item has an associated counter, other elements have implicitly their counter equal to 0. When we process an element, we do one of the following three updates: (1) if the element is being stored, increment its counter by 1, (2) if it is not being stored and the number of stored items is $< k$, store the element and set its counter to 1, (3) otherwise decrement all k counters by 1 and remove those that reach 0. The exact guarantees on the output will not be important now, and we will discuss them in Section 5.

Our contributions We now sketch how to release an MG sketch in a differentially private way.

Consider two neighboring data streams $S = (S_1, \cdots, S_n)$ and $S' = (S_1, \cdots, S_{i-1}, S_{i+1}, \cdots, S_n)$ for some $i \in [n]$. At step $i - 1$, the state of the MG sketch on both inputs is exactly the same. MG_S then receives the item S_i while $MG_{S'}$ does not. This either increments one of the counters of MG_S (possibly by adding an element and raising its counter from 0 to 1) or decrements all its counters. In ℓ_1 distance, the vector of the counters thus changes by at most k. One can show by induction that this will stay this way: at any point in time, $\|MG_S - MG_{S'}\|_1 \leq k$. By a standard global sensitivity argument, one can achieve pure DP by adding noise of magnitude k/ε to each count. This is the approach used in (Chan et al. [2012]). Similarly, we could achieve (ε, δ)-DP by using the Gaussian mechanism (Dwork and Roth [2014]) with noise magnitude proportional to the ℓ_2-sensitivity, which is $\sup_{S,S'} \|MG_S - MG_{S'}\|_2 \leq \sqrt{k}$. We want to instead achieve noise with magnitude $O(1/\varepsilon)$ at each count. To this end, we need to exploit the structure of $MG_S - MG_{S'}$.

What we just described requires that we add the noise to the counts of all items in the universe, also to those that are not stored in the sketch. This results in the maximum error of all frequencies depending on the universe's size, which we do not want. However, it is known that this can be easily solved under (ε, δ)-differential privacy by only adding noise to the stored items and then removing values smaller than an appropriately chosen threshold (Korolova et al. [2009]). This may introduce additional error – for this reason, we end up with error $O(\log(1/\delta)/\varepsilon)$. As this is a somewhat standard technique, we ignore this in this section, we assume that the sketches MG_S and $MG_{S'}$ store the same set of elements; the thresholding allows us to remove this assumption, while allowing us to add noise only to the stored items, at the cost of only getting approximate differential privacy.

We now focus on the structure of $MG_S - MG_{S'}$. After we add to MG_S the element S_i, it either holds (1) that $MG_S - MG_{S'}$ is a vector of all 0's and one 1 or (2) that $MG_S - MG_{S'} = -\mathbf{1}^k$ (We use $\mathbf{1}^k$ the denote the dimension k vector of all ones). We show by induction that this will remain the case as more updates are done to the sketches (note that the remainders of the streams are the same). We do not sketch the proof here, as it is quite technical.

How do we use the structure of $MG_S - MG_{S'}$ to our advantage? We add noise twice. First, we independently add to each counter noise distributed as Laplace$(1/\varepsilon)$. Second, we add to all counters the same value, also distributed as Laplace$(1/\varepsilon)$. That is, we release $MG_S + \text{Laplace}(1/\varepsilon)^{\otimes k} + \text{Laplace}(1/\varepsilon)\mathbf{1}^k$ (For D being a distribution, we use $D^{\otimes k}$ to denote the k-dimensional distribution consisting of k independent copies of D). Intuitively speaking, the first noise hides the difference between S and S' in case (1) and the second noise hides the difference in case (2). We now sketch why this is so for worse constants: $2/\varepsilon$ in place of $1/\varepsilon$. When proving this formally, we use a more technical proof which leads to the better constant.

We now sketch why this is differentially private. Let m_S be the mean of the counters in MG_S for S being an input stream. We may represent MG_S as $(MG_S - m_S\mathbf{1}, m_S)$ (note that there is a bijection between this representation and the original sketch). We now argue that the ℓ_1-sensitivity of this representation is < 2 (treating the representation as a $(k+1)$-dimensional vector for the sake of computing the ℓ_1 distances). Consider the first case. In that case, the averages $m_S, m_{S'}$ differ by $1/k$. As such, $MG_S - m_S\mathbf{1}^k$ and $MG_{S'} - m_{S'}\mathbf{1}^k$ differ by $1/k$ at $k-1$ coordinates and by $1-1/k$ at one coordinate. The overall ℓ_1 change of the representation is thus

$$(k-1)\cdot\frac{1}{k} + (1-1/k) + 1/k = 2 - 1/k < 2.$$

Consider now the second case when $MG_S - MG_{S'} = -\mathbf{1}^k$. Thus, $MG_S - m_S = MG'_S - m_{S'}$. At the same time $|m_S - m_{S'}| = 1$. This means that the ℓ_1 distance between the representations is 1. Overall, the ℓ_1-sensitivity of this representation is < 2.

This means that adding noise from Laplace$(2/\varepsilon)^{\otimes k+1}$ to this representation of MG_S satisfies ε-DP. The resulting value after adding the noise is $(MG_S - m_S\mathbf{1}^k + \text{Laplace}(2/\varepsilon)^{\otimes k}, m_S + \text{Laplace}(2/\varepsilon))$. In the original vector representation of MG_S, this corresponds to $MG_S + \text{Laplace}(2/\varepsilon)^{\otimes k} + \text{Laplace}(2/\varepsilon)\mathbf{1}^k$ and, by post-processing, releasing this value is also differentially private. But this is exactly the value we wanted to show is differentially private!

Preliminaries

Setup of this paper We use $\mathcal{U}$ to denote a universe of elements. We assume that $\mathcal{U}$ is a totally ordered set of size d. That is, $\mathcal{U} = [d]$ where $[d] = \{1, \ldots, d\}$.

Given a stream $S \in \mathcal{U}^{\mathbb{N}}$ we want to estimate the frequency in S of each element of $\mathcal{U}$. Our algorithm outputs a set $T \subseteq \mathcal{U}$ of keys and a frequency estimate c_i for all $i \in T$. The value c_j is implicitly 0 for any $j \notin T$. Let $f(x)$ denote the true frequency of x in the stream S. Our goal is to minimize the largest error between c_x and $f(x)$ among all $x \in \mathcal{U}$. In Section 8 we consider a more general setting where each item in the stream is a set i.e. $S_i \subseteq \mathcal{U}$. In that setting we want to estimate the frequency of sets containing each element such that $f(x) = \sum_{i=1}^{|S|}[x \in S_i]$ for any $x \in \mathcal{U}$, where $[x \in S_i]$ equals 1 if $x \in S_i$ and 0 otherwise.

Differential privacy Differential privacy is a rigorous definition for describing the privacy loss of a randomized mechanism introduced by Dwork et al. [2006b]. Intuitively, differential privacy protects privacy by restricting how much the output distribution can change when replacing the input from one individual. This is captured by the definition of neighboring datasets. We use the add-remove neighborhood definition for differential privacy.

Definition 62 (Neighboring Streams). *Let S be a stream of length n. Streams S and S' are neighboring denoted $S \sim S'$ if there exists an i such that $S = (S'_1, \ldots, S'_{i-1}, S'_{i+1}, \ldots, S'_{n+1})$ or $S' = (S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_n)$.*

Definition 63 (Differential Privacy Dwork and Roth [2014]). *A randomized mechanism $\mathcal{M} : \mathcal{U}^{\mathbb{N}} \to \mathcal{R}$ satisfies (ε, δ)-differential privacy if and only if for all pairs of neighboring streams $S \sim S'$ and all measurable sets of outputs $Z \subseteq \mathcal{R}$ it holds that*

$$\Pr[\mathcal{M}(S) \in Z] \leq e^{\varepsilon} \Pr[\mathcal{M}(S') \in Z] + \delta \ .$$

Samples from a Laplace distribution are used in many differentially private algorithms, most notably the Laplace mechanism Dwork et al. [2006b]. We write Laplace(b) to denote a random variable with a Laplace distribution with scale b centered around 0. We sometimes abuse notation and write Laplace(b) to denote the value of a random variable drawn from the distribution. Our mechanism also works with other noise distributions. We briefly discuss this in Section 8.

Definition 64 (Laplace distribution). *The probability density and cumulative distribution functions of the Laplace distribution centered around 0 with scale parameter b are $f_b(x) = \frac{1}{2b}e^{-|x|/b}$, and $\Pr[\text{Laplace}(b) \leq x] = \frac{1}{2}e^{x/b}$ if $x < 0$ and $1 - \frac{1}{2}e^{-x/b}$ for $x \geq 0$.*

The sensitivity of a deterministic function taking a stream as input restricts the distance between the outputs for neighboring streams. We use the term sensitivity to describe any predicate that holds for the outputs for all pairs of neighboring streams. Two commonly used sensitivities in differential privacy are the ℓ_1/ℓ_2-sensitivities. They are special cases of ℓ_p-sensitivity which bounds the distance between outputs for neighboring streams in the ℓ_p-norm.

Definition 65 (ℓ_p-sensitivity)**.** *Let $g\colon \mathcal{U}^{\mathbb{N}} \to \mathbb{R}^d$ be a deterministic function mapping a stream to a vector of real values. The ℓ_p-sensitivity of g for any $p \geq 1$ is*

$$\Delta_p = \max_{S \sim S'} \|g(S) - g(S')\|_p$$

where $\|g(S) - g(S')\|_p$ is the ℓ_p-distance between $g(S)$ and $g(S')$ defined as

$$\|g(S) - g(S')\|_p := \left(\sum_{i=1}^{d} |g(S) - g(S')|^p \right)^{1/p}$$

Related work

Chan et al. Chan et al. [2012] show that the global ℓ_1-sensitivity of a Misra-Gries sketch is $\Delta_1 = k$. (They actually show that the sensitivity is $k + 1$ but they use a different definition of neighboring datasets that assumes n is known. Applying their techniques under our definition yields sensitivity k.) They achieve privacy by adding noise with scale k/ε to all elements in the universe and keep the top-k noisy counts. This gives an expected maximum error of $O(k \log(d)/\varepsilon)$ with ε-DP for d being the universe size. They use the algorithm as a subroutine for continual observation and merge sketches with an untrusted aggregator. Those settings are not a focus of our work but our algorithm can replace theirs as the subroutine.

Böhler and Kerschbaum Böhler and Kerschbaum [2021] worked on differentially private heavy hitters with no trusted server by using secure multi-party computation. One of their algorithms adds noise to the counters of a Misra-Gries sketch. They avoid adding noise to all elements in the universe by removing noisy counts below a threshold which adds an error of $O(\log(1/\delta)/\varepsilon)$. This is a useful technique for hiding differences in keys between neighboring sketches that removes the dependency on d in the error. Unfortunately, as stated in the introduction their mechanism uses the wrong sensitivity. The ℓ_1-sensitivity of the sketch is k. If the magnitude of noise and the threshold are increased accordingly the error of their approach is $O(k \log(k/\delta)/\varepsilon)$.

If we ignore the memory restriction in the streaming setting, the problem is the same as the top-k problem Mir et al. [2011], Durfee and Rogers [2019], Carvalho et al. [2020], Qiao et al. [2021]. The problem we solve can also be seen as a generalization of the sparse histogram problem. This has been investigated in Cormode et al. [2012], Korolova et al. [2009], Balcer and Vadhan [2019], Aumüller et al. [2022]. Notably, Balcer and Vadhan Balcer and Vadhan [2019] provides a lower bound showing that for any (ε, δ)-differentially private mechanism that outputs at most k counters, there exists input such that the expected error for some elements is $\Omega(\min(\log(d/k)/\varepsilon, \log(1/\delta)/\varepsilon, n))$

(assuming $\varepsilon^2 > \delta$). The noise that we add in our main contribution in fact matches the second branch of the min over all elements.

A closely related problem is that of implementing frequency oracles in the streaming setting under differential privacy. This has been studied in e.g. Zhao et al. [2022b], Pagh and Thorup [2022], Ghazi et al. [2019]. These approaches do not directly return the heavy hitters. The simplest approach for finding the heavy hitters is to iterate over the universe which might be infeasible. However, there are constructions for finding heavy hitters with frequency oracles more efficiently (see Bassily et al. Bassily et al. [2017]). However, as we discussed in the introduction, the approach of Bassily et al. [2017] leads to worse maximum error than what we get unless the sketch size is very large and the universe size is small.

The heavy hitters problem has also received a lot of attention in local differential privacy, starting with the paper introducing the RAPPOR mechanism Erlingsson et al. [2014] and continuing with Qin et al. [2016], Zhao et al. [2022a], Wang et al. [2019], Bun et al. [2019], Wu and Wirth [2022], Bassily et al. [2017]. This problem is practically relevant, it is used for example by Apple to find commonly used emojis Apple. The problem has also been recently investigated when using cryptographic primitives Zhu et al. [2020].

Blocki et al. [2022], Tětek [2022] have recently given general frameworks for designing differentially private approximation algorithms; however, if used naively, they are not very efficient for releasing multiple values (not more efficient than using composition) and they are thus not suitable for the heavy hitters problem.

In Section 8 we consider a more general setting where each item in the stream is a set of size at most m instead of a single element. Both Chan et al. [2012] and Böhler and Kerschbaum [2021] also consider the setting where multiple elements differ between neighboring sketches and they achieve privacy by scaling the noise linearly to m. This scaling is required for mechanisms based on Laplace noise even if we ignore the memory restriction of the streaming setting. However, when the items are distinct we can instead add noise from a normal distribution. The magnitude of the noise scales only with the square root of the number of differing counts. We use such a mechanism in Section 8 based on work by Karrer, Kifer, Wilkins, and Zhang Wilkins et al. [2024]. We discuss their results further in Theorem 80.

Differentially Private Misra-Gries Sketch

In this section, we present our algorithm for privately releasing Misra-Gries sketches. We first present our variant of the non-private Misra-Gries sketch in Algorithm 25 and later show how we add noise to achieve (ε, δ)-differential privacy. The algorithm we use differs slightly from most implementations of MG in that we do not remove elements that have weight 0 until we need to re-use the counter. This will allow us to achieve privacy with slightly lower error.

At all times, k counters are stored as key-value pairs. We initialize the sketch with dummy keys that are not part of $\mathcal{U}$. This guarantees that we never output any elements that are not part of the stream, assuming we remove the dummy counters as post-processing.

The algorithm processes the elements of the stream one at a time. At each step one of three updates is performed: (1) If the next element of the stream is already stored the counter is incremented by 1. (2) If there is no counter for the element and all k counters have a value of at least 1 they are all decremented by 1. (3) Otherwise, one of the elements with a count of zero is replaced by the new element.

In case (3) we always remove the smallest element with a count of zero. This allows us to limit the number of keys that differ between sketches for neighboring streams as shown in Lemma 67. The choice of removing the minimum element is arbitrary but the order of removal must be independent of the stream so that it is consistent between neighboring datasets. The limit on differing keys allows us to obtain a slightly lower error for our private mechanism. However, it is still possible to apply our mechanism with standard implementations of MG. We discuss this in Section 8.

Algorithm 25: Misra-Gries (MG)

```
Input: Positive integer $k$ and stream $S \in \mathcal{U}^{\mathbb{N}}$
$T \leftarrow \{d+1, \dots, d+k\}$ // Start with $k$ dummy counters
$c_i \leftarrow 0$ for all $i \in T$
foreach $x \in S$ do
    if $x \in T$ then // Branch 1
        $c_x \leftarrow c_x + 1$
    else if $c_i \geq 1$ for all $i \in T$ then // Branch 2
        $c_i \leftarrow c_i - 1$ for all $i \in T$
    else // Branch 3
        Let $y \in T$ be the smallest key satisfying $c_y = 0$
        $T \leftarrow (T \cup \{x\}) \setminus \{y\}$
        $c_x \leftarrow 1$
return $T, c$
```

The same guarantees about correctness hold for our version of the MG sketch, as for the original version. This can be easily shown, as the original version only differs in that it immediately removes any key whose counter is zero. Since the counters for items not in the sketch are implicitly zero, one can see by induction that the estimated frequencies by our version are exactly the same as those in the original version. We still need this modified version, as the set of keys it stores is different from the original version, which we use below. The fact that the returned estimates are the same however allows us to use the following fact

Fact 66 (Bose et al. Bose et al. [2003]). *Let $\hat{f}(x)$ be the frequency estimates given by an MG sketch of size k for n being the input size. Then for all $x \in \mathcal{U}$, it holds that $\hat{f}(x) \in [f(x) - n/(k+1), f(x)]$, where $f(x)$ is the true frequency of x in S.*

Note that this is optimal for any mechanism that returns a set of at most k elements. This is easy to see for an input stream that contains $k+1$ distinct elements each with frequency $n/(k+1)$ since at least one element must be removed.

We now analyze the value of $MG_S - MG_{S'}$ for S, S' being neighboring inputs (recall Definition 62). We will then use this in order to prove privacy. As mentioned in Section 5, Chan et al. Chan et al. [2012] showed that the ℓ_1-sensitivity for Misra-Gries sketches is k. They show that this holds after processing the element that differs for neighboring streams and use induction to show that it holds for the remaining stream. Our analysis follows a similar structure. We expand on their result by showing that the sets of stored elements for neighboring inputs differ by at most two elements when using our variant of Misra-Gries. We then show how all this can be used to get differential privacy with only a small amount of noise.

Lemma 67. *Let $T, c \leftarrow \mathrm{MG}(k, S)$ and $T', c' \leftarrow \mathrm{MG}(k, S')$ be the outputs of Algorithm 25 on a pair of neighboring streams $S \sim S'$ such that S' is obtained by removing an element from S. Then $|T \cap T'| \geq k - 2$ and all counters not in the intersection have a value of at most 1. Moreover, it holds that either (1) $c_i = c'_i - 1$ for all $i \in T'$ and $c_j = 0$ for all $j \notin T'$ or (2) there exists an $i \in T$ such that $c_i = c'_i + 1$ and $c_j = c'_j$ for all $j \neq i$.*

Proof. Let $S \sim S'$ be pair of neighboring streams where S' is obtained by removing one element from S. We show inductively that the Lemma holds for any such S and S'. Let $w = T - T'$ and $w' = T' - T$ denote the set of keys that are only in one sketch. Let c_0 and c'_0 denote the smallest element with a zero count in the respective sketch when such an element exists. Then at any point during execution after processing the element removed from S the sketches are in one of the following states:

(S1) $T = T'$ and $c_i = c'_i - 1$ for all $i \in T$.

(S2) There exist $x_1, x_2 \in \mathcal{U}$ such that $w = \{x_1\}$ and $w' = \{x_2\}$, $c_{x_1} = 0$, $c'_{x_2} = 1$ and $c_i = c'_i - 1$ for all $i \in T \cap T'$.

(S3) $T = T'$ and there exists $x_1 \in \mathcal{U}$ such that $c_{x_1} = c'_{x_1} + 1$ and $c_i = c'_i$ for all $i \in T \setminus \{x_1\}$.

(S4) There exist $x_1, x_2 \in \mathcal{U}$ such that $w = \{x_1\}$ and $w' = \{x_2\}$, $c_{x_1} = 1$, $c'_{x_2} = 0$ and $c_i = c'_i$ for all $i \in T \cap T'$.

(S5) There exist $x_1, x_2, x_3 \in \mathcal{U}$ such that $c_{x_1} = c'_{x_1} + 1$, $w = \{x_2\}$, $w' = \{x_3\}$, $c_{x_2} = 0$, $c'_{x_3} = 0$ and $c_i = c'_i$ for all $i \in T \cap T' \setminus \{x_1\}$.

(S6) There exist $x_1, x_2, x_3, x_4 \in \mathcal{U}$ such that $w = \{x_1, x_2\}$ and $w' = \{x_3, x_4\}$, $c_{x_1} = 1$, $c_{x_2} = c'_{x_3} = c'_{x_4} = 0$, $c_i = c'_i$ for all $i \in T \cap T'$ and $x_4 = c'_0$.

Let $x = S_i$ be the element in stream S which is not in stream S'. Since the streams are identical in the first $i-1$ steps the sketches are clearly the same before step i. If there is a counter for x in the sketch we execute Branch 1 and the result corresponds to state S3. If there is no counter for x and no zero counters we execute Branch 2 and the result corresponds to state S1. Otherwise, the 3rd branch of the algorithm is executed and c_0 is replaced by x which corresponds to state S4. Therefore we must be in one of the states S1, S3, or S4 for $T, c \leftarrow \mathrm{MG}(k, (S_1, \ldots, S_i))$ and $T', c' \leftarrow \mathrm{MG}(k, (S'_1, \ldots, S'_{i-1}))$.

We can then prove inductively that the Lemma holds since the streams are identical for the elements $(S_{i+1}, \ldots, S_n)$. We have to consider the possibility of each of the branches being executed for both sketches. The simplest case is when the element has a counter in both sketches and Branch 1 is executed on both inputs. This might happen in all states and we stay in the same state after processing the element. But many other cases lead to new states.

Below we systematically consider all outcomes of processing an element $x \in \mathcal{U}$ when the sketches start in each of the above states. When processing each element, one of the three branches is executed for each sketch. This gives us up to 9 combinations to check, although some are impossible for certain states. Furthermore, when Branch 3 is executed we often have to consider which element is replaced as it leads to different states. We refer to T, c and T', c' as sketch 1 and sketch 2, respectively.

State S1: If $x \in T$ then $x \in T'$ and Branch 1 is executed for both sketches. Similarly, if Branch 2 is executed for sketch 1 it must also be executed for sketch 2 as all counters are strictly larger. Therefore we stay in state S1 in both cases. It is impossible to execute Branch 3 for sketch 2 since all counters are non-zero by definition. As such the final case to consider is when $x \notin T$ and there is a counter with value 0 in sketch 1. In this case, we execute Branch 3 for sketch 1 and Branch 2 for sketch 2. This results in state S4.

State S2: If $x \in T$ we execute Branch 1 on sketch 1 and there are two possible outcomes. If $x \neq x_1$ we also execute Branch 1 on sketch 2 and remain in state S2. If $x = x_1$ we execute Branch 2 on sketch 2 in which case there are no changes to T or T' but now $c_x = 1$ and $c_i = c'_i$ for all $i \in T \cap T'$. As such, we transitioned to state S4.

Since $c_{x_1} = 0$ by definition Branch 2 is never executed on sketch 1 and Branch 3 is never executed on sketch 2 as all counters are non-zero. If $x = x_2$ Branch 3 is executed on sketch 1 and Branch 1 is executed for sketch 2. If $c_0 = x_1$ the sketches have the same keys after processing x and transition to state S1, otherwise if $c_0 \neq x_1$ the sketches still differ for one key and remain in state S2.

Finally, if Branch 3 is executed on sketch 1 and Branch 2 is executed on sketch 2 we again have two possibilities. In both cases, the sketches store the same count on all elements from $T \cap T'$ after processing x. If $c_0 = x_1$ it is removed from T and replaced by x with $c_x = 1$ which corresponds to state S4. If $c_0 \neq x_1$ we must have that $c_0 \in T \cap T'$. The two sketches now differ on exactly two keys after processing x. One of the two keys stored in sketch 2 that are not in sketch 1 must be the minimum zero key since the elements c_0 and x_2 now have counts of zero in sketch 2 and c_0 was the minimum zero key in $T \cap T'$. Therefore we transition to state S6.

State S3: The simplest case is $x \in T$ since then $x \in T'$ and Branch 1 is executed for both sketches. If Branch 2 is executed for sketch 1 and $c'_{x_1} \neq 0$ Branch 2 is also executed for sketch 2. For both cases, we remain in state S3. Instead, if $c'_{x_1} = 0$ Branch 3 is executed for sketch 2. Since all counters are decremented for sketch 1 and x_1 is replaced in sketch 2 we transition to state S2. Lastly, if Branch 3 is executed for sketch 1 it is also executed for sketch 2 and there are two cases. If the same element is removed we remain in state S3. Otherwise, if x_1 is replaced in sketch 2 we transition to state S4.

State S4: Sketch 2 contains a counter with a value of zero in states S4, S5, and S6. Therefore Brach 2 is never executed on sketch 2 in the rest of the proof. If Branch 1 is executed for both sketches we stay in the same state as always but if $x = x_1$ Branch 1 is executed for sketch 1 and Branch 3 is executed for sketch 2. If $c'_0 = x_2$ then $T = T'$ after processing x and we transition to state S3. If $c'_0 \neq x_2$ another element is removed from sketch 2 which must also have a count of zero in sketch 1 and we go to state S5.

If Branch 2 is executed on sketch 1 we know that c'_{x_2} must be the only zero counter in sketch 2. Therefore it does not matter if Branch 1 or 3 is executed on sketch 2. For both cases, we set $c_x = 1$ and the sketches differ in one key which corresponds to state S2.

Finally, if Branch 3 is executed on sketch 1 we again have two cases that lead to the same state. If $x = x_2$ or $c'_0 = x_2$ the counter c'_{x_2} is updated or replaced but the counter that was removed from sketch 1 still remains in sketch 2. Otherwise, we have $c_0 = c'_0$ and we replace the same counter in sketches 1 and 2. Therefore we remain in state S4 in both cases.

State S5: Since by definition both sketches contain counters with a value of zero, Branch 2 is never executed while in this state. If $x \in T \cap T'$ we remain in the same state as always. We have to consider the cases where $x = x_2$, $x = x_3$, and $x \notin T \cup T'$. The resulting state depends on the elements that are replaced in the sketch. For $x = x_2$ we transition to state S3 if $c'_0 = x_3$ and remain in state S5 otherwise. The same argument shows that we transition to state S3 or S5 based on c_0 if $x = x_3$. When $x \notin T \cup T'$ we execute Branch 3 on both sketches. We transition to state S3 only if $c_0 = x_2$ and $c'_0 = x_3$ since otherwise both sketches still have a zero counter that is not stored in the other sketch and we stay in state S5.

State S6: Similar to state S5, Branch 2 is never executed from this state. Here we have to consider the five cases where $x \in T \cap T'$, $x = x_1$, $x = x_2$, $x \in w'$, and $x \notin T \cup T'$. We know that x_4 is replaced whenever $x \notin T'$. If $x \in T \cap T'$ we execute Branch 1 on both sketches and remain in state S6. If $x = x_1$ we transition to state S5 and for $x = x_2$ we transition to state S4. When $x \in w'$ there are two possibilities. We always have $c_x = c'_x$ after updating. If $c_0 = x_2$ the sketches will share $k - 1$ keys and transition to state S4. If $c_0 \neq x_2$ then another element that has a count of zero in both sketches is replaced in sketch 1. We know that either this element or the remaining zero-valued element of w' must be the smallest zero-valued element in sketch 2. Therefore we remain in state S6.

The final case to consider is when $x \notin T \cup T'$. In this case Branch, 3 is executed for sketch 2 and x_4 is replaced with x in T'. If $c_0 = x_2$ we transition to state S4. Otherwise, either x_3 or the element that was replaced from sketch 1 must be the minimum element with a count of zero in sketch 2. As such, we remain in state S6.

□

Next, we consider how to add noise to release the Misra-Gries sketch under differential privacy. Recall that Chan et al. Chan et al. [2012] achieves privacy by adding noise to each counter which scales with k. We avoid this by utilizing the structure of sketches for neighboring streams shown in Lemma 67. We sample noise from Laplace$(1/\varepsilon)$ independently for each counter, but we also sample one more random variable from the same distribution which is added to all counters. Small values are then discarded using a threshold to hide differences in the sets of stored keys between neighboring inputs. This is similar to the technique used by e.g. Korolova et al. [2009]. The algorithm takes the output from MG as input. We sometimes write PMG(k, S) as a shorthand for PMG(MG(k, S)).

Algorithm 26: Private Misra-Gries (PMG)

```
Parameters: $\varepsilon, \delta > 0$
Input     : Output from Algorithm 25:
            $T, c \leftarrow \mathrm{MG}(k, S)$
$\tilde{T} \leftarrow \emptyset$
Sample $\eta \sim \mathrm{Laplace}(1/\varepsilon)$
foreach $x \in T$ do
    $c_x \leftarrow c_x + \eta + \mathrm{Laplace}(1/\varepsilon)$
    if $c_x \geq 1 + 2\ln(3/\delta)/\varepsilon$ then
        $\tilde{T} \leftarrow \tilde{T} \cup \{x\}$
        $\tilde{c}_x \leftarrow c_x$
return $\tilde{T}, \tilde{c}$
```

We prove the privacy guarantees in three steps. First, we show that changing either a single counter or all counters by 1 does not change the output distribution significantly (Corollary 69). This assumes that, for neighboring inputs, the set of stored elements is exactly the same. By Lemma 67, we have that the difference between the sets of stored keys is small and the corresponding counters are ≤ 1. Relying on the thresholding, we bound the probability of outputting one of these keys (Lemma 70). Finally, we combine these two lemmas to show that the privacy guarantees hold for all cases (we do this in Lemma 71).

Lemma 68. *Let us have $x, x' \in \mathbb{R}^k$ such that one of the following three cases holds*

1. *$\exists i \in [k]$ such that $|x_i - x'_i| = 1$ and $x_j = x'_j$ for all $j \neq i$.*
2. *$x_i = x'_i - 1$ for all $i \in [k]$.*
3. *$x_i = x'_i + 1$ for all $i \in [k]$.*

Then we have for any measurable set Z that

$$\Pr[x + \mathrm{Laplace}(1/\varepsilon)^{\otimes k} + \mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k \in Z] \leq e^{\varepsilon} \Pr[x' + \mathrm{Laplace}(1/\varepsilon)^{\otimes k} + \mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k \in Z]$$

Proof. Throughout the proof, we construct sets by applying a translation to all elements of another set. That is, for any $\phi \in \mathbb{R}^k$ and measurable set Z we define $Z - \phi = \{a \in \mathbb{R}^k | a + \phi \in Z\}$. We first focus on the simpler case (1). It holds by the law of total expectation that

$$\begin{aligned} \Pr[x + \mathrm{Laplace}(1/\varepsilon)^{\otimes k} + \mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k \in Z] = \\ \mathbb{E}_{N \sim \mathrm{Laplace}(1/\varepsilon)}\left[\Pr[\mathrm{Laplace}(1/\varepsilon)^{\otimes k} \in Z - x - N\mathbf{1}^k | N]\right] \leq \\ e^{\varepsilon}\mathbb{E}_{N \sim \mathrm{Laplace}(1/\varepsilon)}\left[\Pr[\mathrm{Laplace}(1/\varepsilon)^{\otimes k} \in Z - x' - N\mathbf{1}^k | N]\right] = \\ e^{\varepsilon} \Pr[x' + \mathrm{Laplace}(1/\varepsilon)^{\otimes k} + \mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k \in Z] \end{aligned}$$

where to prove the inequality, we used that for any measurable set A, it holds

$$\Pr[\mathrm{Laplace}(1/\varepsilon)^{\otimes k} \in A] \leq e^{\varepsilon} \Pr[\mathrm{Laplace}(1/\varepsilon)^{\otimes k} \in A - \phi]$$

for any $\phi \in \mathbb{R}^k$ with $\|\phi\|_1 \leq 1$ (see Dwork et al. [2006b]). Specifically, we have set $A = Z - x - N\mathbf{1}^k$ and $\phi = x - x'$ such that $\|\phi\|_1 = 1$.

We now focus on the cases $(2), (3)$. We will prove below that for x, x' satisfying one of the conditions $(2), (3)$ and for any measurable A, Z and $N_1 \sim \mathrm{Laplace}(1/\varepsilon)^{\otimes k}$, it holds

$$\Pr[x + N_1 + \mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k \in Z | N_1 \in A] \leq e^{\varepsilon} \Pr[x' + N_1 + \mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k \in Z | N_1 \in A]$$

This allows us to argue like above:

$$\begin{aligned}
\Pr[x + \mathrm{Laplace}(1/\varepsilon)^{\otimes k} + \mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k \in Z] &= \\
\mathbb{E}_{N_1 \sim \mathrm{Laplace}(1/\varepsilon)^{\otimes k}}\left[\Pr[x + N_1 + \mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k \in Z | N_1]\right] &\leq \\
e^{\varepsilon}\mathbb{E}_{N_1 \sim \mathrm{Laplace}(1/\varepsilon)^{\otimes k}}\left[\Pr[x' + N_1 + \mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k \in Z | N_1]\right] &= \\
e^{\varepsilon} \Pr[x' + \mathrm{Laplace}(1/\varepsilon)^{\otimes k} + \mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k \in Z]
\end{aligned}$$

which would conclude the proof. Let $g : \mathbb{R} \to \mathbb{R}^k$ be the function $g(a) = a\mathbf{1}^k$ and define $g^{-1}(B) = \{a \in \mathbb{R} | g(a) \in B\}$ and note that g is measurable. We focus on the case (2); the same argument works for (3) as we discuss below. It holds

$$\begin{aligned}
\Pr[x + N_1 + \mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k \in Z | N_1 \in A] &= \\
\Pr[\mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k \in Z - x - N_1 | N_1 \in A] &= \\
\Pr[\mathrm{Laplace}(1/\varepsilon) \in g^{-1}(Z - x - N_1) | N_1 \in A] &= \\
\Pr[\mathrm{Laplace}(1/\varepsilon) \in g^{-1}(Z - x' - \mathbf{1}^k - N_1) | N_1 \in A] &= \\
\Pr[\mathrm{Laplace}(1/\varepsilon) \in g^{-1}(Z - x' - N_1) - 1 | N_1 \in A] &\leq \\
e^{\varepsilon} \Pr[\mathrm{Laplace}(1/\varepsilon) \in g^{-1}(Z - x' - N_1) | N_1 \in A] &= \\
e^{\varepsilon} \Pr[\mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k \in Z - x' - N_1 | N_1 \in A] &= \\
e^{\varepsilon} \Pr[x' + N_1 + \mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k \in Z | N_1 \in A].
\end{aligned}$$

To prove the inequality, we again used the standard result that for any measurable A, it holds that $\Pr[\mathrm{Laplace}(1/\varepsilon) \in A] \leq e^{\varepsilon} \Pr[\mathrm{Laplace}(1/\varepsilon) \in A - 1]$. The same holds for $A + 1$; this allows us to use the exact same argument in case (3), in which the proof is the same except that -1 on lines 4,5 of the manipulations is replaced by $+1$. □

Corollary 69. *Let T, c and T', c' be two sketches such that $T = T'$ and one of following holds:*

1. *$\exists i \in T$ such that $|c_i - c'_i| = 1$ and $c_j = c'_j$ for all $j \neq i$.*

2. $c_i = c'_i - 1$ *for all* $i \in T$.

3. $c_i = c'_i + 1$ *for all* $i \in T$.

Then for any measurable set of outputs Z*, we have:*

$$\Pr[\mathrm{PMG}(T, c) \in Z] \leq e^{\varepsilon} \Pr[\mathrm{PMG}(T', c') \in Z]$$

Proof. Consider first a modified algorithm PMG′ that does not perform the thresholding: that is, if we remove the condition on line 5. It can be easily seen that PMG′ only takes the vector c and releases $c + \mathrm{Laplace}(1/\varepsilon)^{\otimes k} + \mathrm{Laplace}(1/\varepsilon)\mathbf{1}^k$. We have just shown in Lemma 68 that this means that for any measurable Z',

$$\Pr[\mathrm{PMG}'(T, c) \in Z'] \leq e^{\varepsilon} \Pr[\mathrm{PMG}'(T', c') \in Z'].$$

Let $\tau(x) = x$ for $x \geq 1 + 2\ln(3/\delta)/\varepsilon$ and 0 otherwise. Since $\mathrm{PMG}(T, c) = \tau(\mathrm{PMG}'(T, c))$, it then holds

$$\Pr[\mathrm{PMG}(T, c) \in Z] = \Pr[\mathrm{PMG}'(T, c) \in \tau^{-1}(Z)] \leq \\ e^{\varepsilon} \Pr[\mathrm{PMG}'(T', c') \in \tau^{-1}(Z)] = e^{\varepsilon} \Pr[\mathrm{PMG}(T', c') \in Z]$$

as we wanted to show. □

Next, we bound the effect on the output distribution from keys that differ between sketches by δ.

Lemma 70. *Let* T, c *and* T', c' *be two sketches of size* k *and let* $\hat{T} = T \cap T'$*. If we have that* $|\hat{T}| \geq k - 2$*,* $c_i = c'_i$ *for all* $i \in \hat{T}$*, and for all* $x \notin \hat{T}$*, it holds* $c_x, c'_x \leq 1$*. Then for any measurable set* Z*, it holds*

$$\Pr[\mathrm{PMG}(T, c) \in Z] \leq \Pr[\mathrm{PMG}(T', c') \in Z] + \delta$$

Proof. Let $\mathrm{PMG}'(T, c)$ denote a mechanism that executes $\mathrm{PMG}(T, c)$ and post-processes the output by discarding any elements not in $\hat{T}$. It is easy to see that (a) $\Pr[\mathrm{PMG}'(T, c) \in Z] = \Pr[\mathrm{PMG}'(T', c') \in Z]$ since the input sketches are identical for all elements in $\hat{T}$. Moreover, for any output $\tilde{T}, \tilde{c} \leftarrow \mathrm{PMG}(T, c)$ for which $\tilde{T} \subseteq \hat{T}$, the post-processing does not affect the output. This gives us the following inequalities: (b) $\Pr[\mathrm{PMG}(T, c) \in Z] \leq \Pr[\mathrm{PMG}'(T, c) \in Z] + \Pr[\tilde{T} \not\subseteq \hat{T}]$ and (c) $\Pr[\mathrm{PMG}'(T', c') \in Z] \leq \Pr[\mathrm{PMG}(T, c) \in Z] + \Pr[\tilde{T}' \not\subseteq \hat{T}]$. Combining equations $(a)-(c)$, we get the inequality $\Pr[\mathrm{PMG}(T, c) \in Z] \leq \Pr[\mathrm{PMG}(T', c') \in Z] + \Pr[\tilde{T} \not\subseteq \hat{T}] + \Pr[\tilde{T}' \not\subseteq \hat{T}]$.

As such, the Lemma holds if $\Pr[\tilde{T} \not\subseteq \hat{T}] + \Pr[\tilde{T}' \not\subseteq \hat{T}] \leq \delta$. That is, it suffices to prove that with probability at most δ any noisy count for elements not in $\hat{T}$ is at least $1 + 2\ln(3/\delta)/\varepsilon$. The noisy count for such a key can only exceed the threshold if one of the two noise samples added to the key is at least $\ln(3/\delta)/\varepsilon$. From Definition 64 we have $\Pr[\mathrm{Laplace}(1/\varepsilon) \geq \ln(3/\delta)/\varepsilon] = \delta/6$.

There are at most 4 keys not in $\hat{T}$ which are in $T \cup T'$ and therefore at most 6 noise samples affect the probability of outputting such a key (the 4 individual Laplace noise samples and the 2 global Laplace noise samples, one for each sketch). By a union bound the probability that any of these samples exceeds $\ln(3/\delta)/\varepsilon$ is at most δ. □

We are now ready to prove the privacy guarantee of Algorithm 26.

Lemma 71. *Algorithm 26 is (ε, δ)-differentially private for any k.*

Proof. The Lemma holds if and only if for any pair of neighboring streams $S \sim S'$ and any measurable set Z we have:

$$\Pr[\mathrm{PMG}(T, c) \in Z] \leq e^{\varepsilon} \Pr[\mathrm{PMG}(T', c') \in Z] + \delta,$$

where $T, c \leftarrow \mathrm{MG}(k, S)$ and $T', c' \leftarrow \mathrm{MG}(k, S')$ denotes the non-private sketches for each stream.

We prove the guarantee above using an intermediate sketch that "lies between" T, c and T', c'. The sketch has support T' and we denote the counters as $\hat{c}$. By Lemma 67, we know that $|T \cap T'| \geq k - 2$ and all counters in c and not in $T \cap T'$ are at most 1. We will now come up with some conditions on $\hat{c}$ such that if these conditions hold, the lemma follows. We will then prove the existence of such $\hat{c}$ below. Assume that $\hat{c}_i = c_i$ for all $i \in T \cap T'$ and $\hat{c}_j \leq 1$ for all $j \in T' \setminus T$. Lemma 70 then tells us that

$$\Pr[\mathrm{PMG}(T, c) \in Z] \leq \Pr[\mathrm{PMG}(T', \hat{c}) \in Z] + \delta.$$

Assume also that one of the required cases for Corollary 69 holds between $\hat{c}$ and c'. We have

$$\Pr[\mathrm{PMG}(T', \hat{c}) \in Z] \leq e^{\varepsilon} \Pr[\mathrm{PMG}(T', c') \in Z].$$

Therefore, if such a sketch $T', \hat{c}$ exists for all S and S' the lemma holds since

$$\Pr[\mathrm{PMG}(T, c) \in Z] \leq \Pr[\mathrm{PMG}(T', \hat{c}) \in Z] + \delta \leq e^{\varepsilon} \Pr[\mathrm{PMG}(T', c') \in Z] + \delta \ .$$

It remains to prove the existence of $\hat{c}$ such that $\hat{c}_i = c_i$ for all $i \in T \cap T'$ and $\hat{c}_j \leq 1$ for all $j \in T' \setminus T$ and such that one of the conditions (1) – (3) of Corollary 69 holds between $\hat{c}$ and c'. We first consider neighboring streams where S' is obtained by removing an element from S. From Lemma 67 we have two cases to consider. If $c_i = c'_i - 1$ for all $i \in T'$ we simply set $\hat{c} = c$. Recall that we implicitly have $c_i = 0$ for $i \notin T$. Therefore the sketch satisfies the two conditions above since $\hat{c}_i = c_i$ for all $i \in \mathcal{U}$ and condition (2) of Corollary 69 holds. In the other case where $c_i = c'_i + 1$ for exactly one $i \in T$ there are two possibilities. If $i \in T'$ we again set $\hat{c} = c$. When $i \notin T'$ there must exist at least one element $j \in T'$ such that $c'_j = 0$ and $j \notin T$. We set $\hat{c}_j = 1$ and $\hat{c}_i = c'_i$ for all $i \neq j$. In both cases $\hat{c}_i = c_i$ for all $i \in T \cap T'$ and $\hat{c}_j$ is at most one for $j \notin T$. There is exactly one element with a higher count in $\hat{c}$ than c' which means that condition (1) of Corollary 69 holds.

If S is obtained by removing an element from S' the cases from Lemma 67 are flipped. If $c_i - 1 = c'_i$ for all $i \in T$ and $c'_j = 0$ for $j \notin T$ we set $\hat{c}_i = c_i$ if $i \in T$ and $\hat{c}_i = 1$ otherwise. It clearly holds that $\hat{c}_i = c_i$ for all $i \in T \cap T'$ and $\hat{c}_j \leq 1$ for all $j \notin T$. Since $\hat{c}_i = c'_i + 1$ for all $i \in T'$ condition (3) of Corollary 69 holds. Finally, if $c_i + 1 = c'_i$ for exactly one $i \in T'$ we simply set $\hat{c} = c$. $\hat{c}_i = c_i$ clearly holds for all $i \in T \cap T'$, $\hat{c}_j = 0$ for all $j \notin T$, and condition (1) of Corollary 69 holds between $\hat{c}$ and c'. □

Next, we analyze the error compared to the non-private sketch. We state the error in terms of the largest error among all elements of the sketch. Recall that we implicitly say that the count is zero for any element not in the sketch.

Lemma 72. *Let $\tilde{T}, \tilde{c} \leftarrow \mathrm{PMG}(T, c)$ denote the output of Algorithm 26 for any sketch T, c with $|T| = k$. Then with probability at least $1 - \beta$ we have*

$$\tilde{c}_x \in \left[c_x - \frac{2\ln\left(\frac{k+1}{\beta}\right)}{\varepsilon} - 1 - \frac{2\ln\left(3/\delta\right)}{\varepsilon}, c_x + \frac{2\ln\left(\frac{k+1}{\beta}\right)}{\varepsilon} \right]$$

for all $x \in T$ and $\tilde{c}_x = 0$ for all $x \notin T$.

Proof. The two sources of error are the noise samples and the thresholding step. We begin with a simple bound on the absolute value of the Laplace distribution.

$$\Pr\left[|\mathrm{Laplace}(1/\varepsilon)| \geq \frac{\ln((k+1)/\beta)}{\varepsilon} \right] = 2\cdot\Pr\left[\mathrm{Laplace}(1/\varepsilon) \leq -\frac{\ln((k+1)/\beta)}{\varepsilon} \right] = \beta/(k+1) \ .$$

Since $k+1$ samples are drawn we know by a union bound that the absolute value of all samples is bounded by $\ln((k+1)/\beta)/\varepsilon$ with probability at least $1-\beta$. As such the absolute error from the Laplace samples is at most $2\ln((k+1)/\beta)/\varepsilon$ for all $x \in T$ since two samples are added to each count. Removing noisy counts below the threshold potentially adds an additional error of at most $1 + 2\ln(3/\delta)/\varepsilon$. It is easy to see that $\tilde{c}_x = 0$ for all $x \notin T$ since the algorithm never outputs any such elements. □

Theorem 73. $\mathrm{PMG}(k, S)$ *satisfies (ε, δ)-differential privacy. Let $f(x)$ denote the frequency of any element $x \in \mathcal{U}$ in S and let $\hat{f}(x)$ denote the estimated frequency of x from the output of* $\mathrm{PMG}(k, S)$. *For any x with $f(x) = 0$ we have $\hat{f}(x) = 0$ and with probability at least $1 - \beta$ we have for all $x \in \mathcal{U}$*

$$\hat{f}(x) \in \left[f(x) - \frac{2\ln\left(\frac{k+1}{\beta}\right)}{\varepsilon} - 1 - \frac{2\ln(3/\delta)}{\varepsilon} - \frac{|S|}{k+1}, f(x) + \frac{2\ln\left(\frac{k+1}{\beta}\right)}{\varepsilon} \right]$$

Moreover, the algorithm outputs all x, such that $\hat{f}(x) > 0$ and there are at most k such elements. PMG(k, S) *uses* $2k$ *words of memory. The mean squared error for any fixed element $x \in \mathcal{U}$ is bounded by* $\mathbb{E}[(\hat{f}(x) - f(x))^2] \leq 3\left(1 + \frac{2+2\ln(3/\delta)}{\varepsilon} + \frac{|S|}{k+1}\right)^2$.

Proof. The space complexity is clearly as claimed, as we are storing at any time at most k items and counters. We focus on proving privacy and correctness.

If $f(x) = 0$ we know that $x \notin T$ where T is the keyset after running Algorithm 25. Since Algorithm 26 outputs a subset of T we have $\hat{f}(x) = 0$. The first part of the Theorem follows directly from Fact 66 and Lemmas 71 and 72.

We now bound the mean squared error. There are three sources of error. Let r_1 be the error coming from the Laplace noise, r_2 from the thresholding, and r_3 the error made by the MG sketch. Then

$$\mathbb{E}[(\hat{f}(x) - f(x))^2] = \mathbb{E}[(r_1 + r_2 + r_3)^2] \leq 3(\mathbb{E}[r_1^2] + \mathbb{E}[r_2^2] + \mathbb{E}[r_3^2])$$

by equivalence of norms (for any dimension n vector v, $\|v\|_1 \leq \sqrt{n}\|v\|_2$). The errors r_2, r_3 are deterministically bounded $r_2 \leq 1 + 2\ln(3/\delta)/\varepsilon$ and $r_3 \leq |S|/(k+1)$. $\mathbb{E}[r_1^2]$ is the variance of the Laplace noise; we added two independent noises each with scale $1/\varepsilon$ and thus variance $2/\varepsilon^2$ for a total variance of $4/\varepsilon^2$. This finishes the proof. □

Privatizing standard versions of MG

The privacy of our mechanism as presented in Algorithm 26 relies on our variant of the Misra-Gries algorithm. Our sketch can contain elements with a count of zero. However, elements with a count of zero are removed in the standard version of the sketch. As such, sketches for neighboring datasets can differ for up to k keys if one sketch stores k elements with a count of 1 and the other sketch is empty. It is easy to change Algorithm 26 to handle these implementations. We simply increase the threshold to $1 + 2\ln\left(\frac{k+1}{2\delta}\right)/\varepsilon$ since the probability of outputting any of the k elements with a count of 1 is bounded by δ.

Tips for practitioners

Here we discuss some technical details to keep in mind when implementing our mechanism.

The output of the Misra-Gries algorithm is an associative array. In Algorithm 26 we add appropriate noise such that the associative array can be released under differential privacy. However, for some implementations of associative arrays such as hash tables the order in which keys are added affects the data structure. Using such an implementation naively violates differential privacy but it is easily solved either by outputting a random permutation of the key-value pairs or using a fixed order e.g. sorted by key.

We present our mechanism with noise sampled from the Laplace distribution. However, the distribution is defined for real numbers which cannot be represented on a finite computer. This is a known challenge and precision-based attacks still exist on popular implementations Haney et al. [2022]. Since the output of MG is discrete the distribution can be replaced by the Geometric mechanism Ghosh et al. [2012] or one of the alternatives introduced in Balcer and Vadhan [2019]. Our mechanism would still satisfy differential privacy but it might be necessary to change the threshold in Algorithm 26 slightly to ensure that Lemma 70 still holds. Our proof of Lemma 70 works for the Geometric mechanism from Ghosh et al. [2012] when increasing the threshold to $1 + 2\lceil \ln(6e^{\varepsilon}/((e^{\varepsilon}+1)\delta))/\varepsilon \rceil$.

Lastly, it is worth noting that the analysis for Lemma 70 is not tight. We bound the probability of outputting a small key by bounding the value of all relevant samples by $\ln(3/\delta)/\varepsilon$ which is sufficient to guarantee that the sum of any two samples does not exceed $2\ln(3/\delta)/\varepsilon$. This simplifies the proof and presentation significantly however one sample could exceed $\ln(3/\delta)/\varepsilon$ without any pair of samples exceeding $2\ln(3/\delta)/\varepsilon$. A tighter analysis would improve the constant slightly which might matter for practical applications.

Pure Differential Privacy

In this section, we discuss how to achieve ε-differential privacy. We cannot use our approach from Section 5 where we add the same noise to all keys because the set of stored keys can differ between sketches for neighboring datasets. Instead, we achieve privacy by adding noise to all elements of $\mathcal{U}$ scaled to the ℓ_1-sensitivity. Chan et al. Chan et al. [2012] showed that the sensitivity of Misra-Gries sketches scales with the number of counters. We show that a simple post-processing step reduces the sensitivity of the sketch to 2 and the worst-case error of the sketch is still $n/(k+1)$ where $n = |S|$. This allows us to achieve an error of $n/(k+1) + O(\log(d)/\varepsilon)$.

The ℓ_1-sensitivity scales with the size of the sketch since the counts can differ by 1 for all k elements between neighboring datasets. This happens when the decrement step is executed on a given input one fewer or one more time than on a neighboring input. We get around this case by post-processing the sketch before adding noise. We first run the Misra-Gries algorithm on the stream but we count how many times the counters were decremented. That is, we count the number of times Branch 2 of Algorithm 25 was executed and denote this count as γ. The Misra-Gries algorithm decrements the counters at most $\lfloor n/(k+1) \rfloor$ times. We use this fact by first adding γ and then subtracting $n/(k+1)$ from each counter in the sketch. We then remove all elements with negative counters. Although we increase the error of the sketch for some datasets, the worst-case error guarantee is still the same as at most $n/(k+1)$ has been subtracted from each count. Next, we show how this post-processing step reduces the ℓ_1-sensitivity to 2.

Let $S \sim S'$ denote any pair of neighboring streams where S' is obtained by removing one element from S. Consider the effect of running the following procedure on the Misra-Gries sketches for both streams (1) add γ and γ' to the counters of MG_S and $\mathrm{MG}_{S'}$, respectively (2) subtract $|S|/(k+1)$ from the counters in both sketches (3) remove any negative counters from both sketches. It can be shown that the new sketches are either identical or differ by 1 in a single counter. Specifically, we may use the argument from the proof of Lemma 67 to argue that we end in one of the 6 states introduced in that proof before running the procedure. One may verify that the claim holds in all 6 states. Specifically, we get $\gamma = \gamma' + 1$ in the first 2 states and $\gamma = \gamma'$ for the final 4 states. The post-processing step we introduced in the previous paragraph uses the length of the stream which differs by 1 between S and S'. As such, there is an additional difference of $1/(k+1)$ for each counter. The ℓ_1-sensitivity is bounded by 2 since $1 + k/(k+1) < 2$.

We achieve ε-differential privacy by adding noise to our new sketch. We essentially use the same technique as Chan et al. Chan et al. [2012] but the noise no longer scale linearly in k as the sensitivity is bounded by 2. Specifically, we add noise sampled from Laplace$(2/\varepsilon)$ independently to the count of each element from $\mathcal{U}$ and release the top-k noisy counts. A simple union bound shows us that with probability at least $1-\beta$ the absolute value of all samples is bounded by $2\ln(d/\beta)/\varepsilon$. Note that it might be infeasible to actually sample noise for each element when $\mathcal{U}$ is large; we refer to previous work on how to implement this more efficiently Chan et al. [2012], Cormode et al. [2012], Balcer and Vadhan [2019].

It is worth noting that the low sensitivity of the post-processed sketch can also be utilized under (ε, δ)-differential privacy. We can use an approach similar to Korolova et al. [2009]. They add noise to all non-zero counters and remove noisy counts below a threshold to hide small counters. Applying the standard approach for histograms would require a threshold with a small dependence on k as neighboring sketches might disagree on all keys. However, Aumüller et al. [2022] extended the technique to real-valued vectors by probabilistically rounding elements with a value less than the ℓ_1-sensitivity. If we apply their technique directly we get a threshold of $4 + 2\ln(1/\delta)/\varepsilon$. This approach has error guarantees that match those from Theorem 73 up to constant factors. However, this approach has worse guarantees than Algorithm 26 when comparing to the non-private Misra-Gries sketch. By Lemma 72 the error of Algorithm 26 is $O(\log(1/\delta)/\varepsilon)$ with high probability (for sufficiently small δ). Here the error is $n/(k+1) + O(\log(1/\delta)/\varepsilon)$ since we subtract up to $n/(k+1)$ from the counters before adding noise.

Privatizing merged sketches

In practice, it is often important that we may merge sketches. This is for example commonly used when we have a dataset distributed over many servers. Each dataset consists of multiple streams in this setting, and we want to compute an aggregated sketch over all streams. We say that datasets are neighboring if we can obtain one from the other by removing a single element from one of the streams. If the aggregator is untrusted we must add noise to each sketch before performing any merges. This is the setting in Chan et al. [2012] and we can run their merging algorithm. However, since we add noise to each sketch the error scales with the number of sketches. In particular, the error from the thresholding step of Algorithm 26 scales linearly in the number of sketches for worst-case input. In the rest of this section, we consider the setting where aggregators are trusted. We can apply the post-processing step from the previous section to each sketch before aggregating the counters of each element. The ℓ_1-sensitivity of the aggregated sketch is still bounded by 2 so we can use the approach from the previous section. However, the aggregated sketch might have much more than k counters. This approach increases the memory requirement of the aggregator. Next we consider a merging algorithm where aggregators never store more than $2k$ counters.

Agarwal, Cormode, Huang, Phillips, Wei, and Yi Agarwal et al. [2013] introduced the following simple merging algorithm in the non-private setting. Given two Misra-Gries sketches $T_1, c_1 \leftarrow \mathrm{MG}(k, S^{(1)})$ and $T_2, c_2 \leftarrow \mathrm{MG}(k, S^{(2)})$ they first compute the sum of all counters $c_1 + c_2$. There are up to $2k$ counters at this point. They subtract the value of the $k+1$'th largest counter from all elements. Finally, any non-positive counters are removed leaving at most k counters. They show that merged sketches have the same worst-case guarantee as non-merged Misra-Gries sketches. That is, if we compute a Misra-Gries sketch for each stream $(S^{(1)}, \ldots, S^{(l)})$ and merge them into a single sketch, the frequency estimate of all elements is at most $N/(k+1)$ less than the true frequency. Here N is the total length of all streams. This holds for any order of merging and the streams do not need to have the same length.

Unfortunately, the structure between neighboring sketches where either a single counter or exactly k counters differ by 1 breaks down when merging. Therefore we cannot run Algorithm 26 on the merged sketch. However, as we show below, the global sensitivity of merged sketches is independent of the number of merges. The sensitivity only depends on the number of counters. We first show a property for a single merge operation; this will allow us to bound the sensitivity for any number of merges. Note that unlike in the previous section, we do not limit the number of keys that differ between sketches and we do not store keys with a count of zero.

Lemma 74. *Let T_1, c_1, T_1', c_1' and T_2, c_2 denote Misra-Gries sketches of size k*

and denote the sketches merged with the algorithm above as $\hat{T}, \hat{c} \leftarrow \mathrm{Merge}(T_1, c_1, T_2, c_2)$ *and* $\hat{T}', \hat{c}' \leftarrow \mathrm{Merge}(T_1', c_1', T_2, c_2)$*. If* $T_1' \subseteq T_1$ *and* $c_{1i} - c_{1i}' \in \{0, 1\}$ *for all* $i \in \mathcal{U}$ *then at least one of the following holds (1)* $\hat{T}' \subseteq \hat{T}$ *and* $\hat{c}_i - \hat{c}_i' \in \{0, 1\}$ *for all* $i \in \mathcal{U}$ *or (2)* $\hat{T} \subseteq \hat{T}'$ *and* $\hat{c}_i' - \hat{c}_i \in \{0, 1\}$ *for all* $i \in \mathcal{U}$*.*

Proof. Let $\bar{c}$ and $\bar{c}'$ denote the merged counters before subtracting and removing values. Then clearly $\bar{c}_i - \bar{c}_i' \in \{0, 1\}$ for all $i \in \mathcal{U}$. Therefore we have that $\bar{c}_{k+1} - \bar{c}_{k+1}' \in \{0, 1\}$ where $\bar{c}_{k+1}$ denotes the value of the $k+1$'th largest counter in $\bar{c}$. Note that it does not matter if the $k + 1$'th largest counter describes a different element. If $\bar{c}_{k+1} = \bar{c}_{k+1}'$ we subtract the same value from each sketch and we have $\hat{c}_i - \hat{c}_i' \in \{0, 1\}$ for all $i \in \mathcal{U}$. If $\bar{c}_{k+1} - \bar{c}_{k+1}' = 1$ we subtract one more from each count in $\hat{c}$ and we have $\hat{c}_i' - \hat{c}_i \in \{0, 1\}$ for all $i \in \mathcal{U}$. □

Corollary 75. *Let* $(S^{(1)}, \ldots, S^{(l)})$ *and* $(S'^{(1)}, \ldots, S'^{(l)})$ *denote two sets of streams such that* $S^{(i)} \sim S'^{(i)}$ *for one* $i \in [l]$ *and* $S^{(j)} = S'^{(j)}$ *for any* $j \neq i$*. Let* T, c *and* T', c' *be the result of merging Misra-Gries sketches computed on both sets of streams in any fixed order. Then* c *and* c' *differ by 1 for at most* k *elements and agree on all other counts.*

Proof. It is clearly true for sketches of a pair of neighboring datasets by Lemma 67. It holds by induction after each merging operation by Lemma 74. □

Since the ℓ_1-sensitivity is k we can use the algorithm in Chan et al. [2012] that adds noise with magnitude k/ε to all elements in $\mathcal{U}$ and keeps the top-k noisy counts. If we only add noise to non-zero counts we can hide that up to k keys can change between neighboring inputs with a threshold. The two approaches have expected maximum error compared to the non-private merged sketch of $O(k \log(d)/\varepsilon)$ and $O(k \log(k/\delta)/\varepsilon)$, respectively. For (ε, δ)-differential privacy we can also utilize that the ℓ_2-sensitivity is $\sqrt{k}$ because counters only differ by 1. This allows us to add noise that scales slower with k using the Gaussian Sparse Histogram Mechanism Wilkins et al. [2024] which we discuss further in the next section.

User-level Differential Privacy

So far we considered the setting where a user has exactly one element. In this section, we consider the more general setting where a user contributes up to m distinct elements. Specifically, this means that the input is a stream $S = (S_1, S_2, \ldots, S_{n-1}, S_n)$ where for all $i \in [n]$ we have $S_i \subseteq \mathcal{U}$ and $|S_i| \leq m$. Our goal is to estimate the frequency of each element that is $f(x) = \sum_{i \in [n]} [x \in S_i]$. We denote the total length of the stream as $N = \sum_{i \in [n]} |S_i|$. If we want to compute a MG sketch in this setting we have to flatten the input. We denote by $\hat{S}$ a stream created by processing the sets in S one at a time by iterating over each element in the set in some fixed order (e.g. ascending order).

We first show that Algorithm 26 satisfies differential privacy in this setting. The magnitude of noise and the threshold increase as a function of m. We then introduce a new sketch with similar error guarantees to the MG sketch that can be released with less noise than Algorithm 26 for many parameters in this setting. Note that throughout this section we assume that $k > m$. This is a fair assumption because the error guarantees of the MG sketch are meaningless when $m > k$ because $N/(k+1) = (n \cdot m)/(k+1) \geq n$ if $|S_i| = m$ for all i.

We start by restating the group privacy property of differential privacy.

Lemma 76 (Group Privacy (Following Dwork and Roth [2014]))**.** *Let $\mathcal{M}$ be a mechanism satisfying (ε, δ)-differential privacy where one stream of a neighboring pair of streams is obtained by removing 1 element from the other. Then $\mathcal{M}$ satisfies $(m\varepsilon, me^{m\varepsilon}\delta)$-differential privacy for neighboring streams where one stream is obtained by adding or removing at most m elements from the other.*

This allows us to upper bound the privacy parameters when streams differ by multiple elements.

Lemma 77. *Let $\hat{S}$ be the flattened version of the stream S. Then releasing $\mathrm{PMG}(k, \hat{S})$ with parameters $\varepsilon = \varepsilon'/m$ and $\delta = \delta'/(me^{\varepsilon'})$ satisfies (ε', δ')-differential privacy.*

Proof. $\mathrm{PMG}(k, \hat{S})$ satisfies (ε, δ)-differential privacy for streams differing in a single element by Lemma 71. For any neighboring streams $S \sim S'$ we have that wither $\hat{S}$ can be created by removing at most m elements from $\hat{S}'$ or vice-versa. It follows from Lemma 76 that $\mathrm{PMG}(k, \hat{S})$ satisfies (ε', δ')-differential privacy as $\varepsilon' = m\varepsilon$ and $\delta' = me^{m\varepsilon}\delta$. □

Since the proof above does not depend on the fact that the elements of a user are distinct and appear in consecutive order the algorithm can be used in an even more general setting.

Corollary 78. *Algorithm 26 with the parameters of Lemma 77 satisfies (ε', δ')-differential privacy under a definition of neighboring streams where one stream is obtained from the other by adding or removing up to m (possibly duplicate) elements.*

We can use a similar argument to achieve pure differential privacy when streams differ by multiple elements using our technique from Section 8.

Lemma 79. *Let $S \in \mathcal{U}^{\mathbb{N}}$ be a stream of elements where a neighboring stream S' is obtained from S by adding or removing up to m elements. If we compute a MG sketch of S followed by the post-processing step from Section 8, we can release the sketch under ε-DP using the technique of Chan et al. [2012] when noise is sampled from* $\mathrm{Laplace}(2m/\varepsilon)$.

Proof. It follows from Lemma 76 since the mechanism satisfies (ε/m)-differential privacy for streams differing by 1 element because the ℓ_1-sensitivity is bounded by 2 as discussed in Section 8. □

Algorithm 26 and the mechanism by Chan et al. [2012] relies on noise from the Laplace distribution. This choice works well for the settings of Sections 5 and 8. However, when users have multiple distinct items we can instead use Gaussian noise under (ε, δ)-differential privacy. The magnitude of Gaussian noise required for differential privacy scales with the ℓ_2-sensitivity rather than the ℓ_1-sensitivity which is the case for Laplace noise. If we consider this setting of this section without the memory constraints it is clear that a user changes at most m distinct counts by 1 each. The ℓ_1-sensitivity is m but the ℓ_2-sensitivity is only $\sqrt{m}$.

We next discuss a general mechanism for releasing sketches using Gaussian noise. The mechanism is similar to the technique with Laplace noise by Korolova et al. [2009]. The idea is to add noise from a Gaussian distribution to all non-zeroes counters and then remove small noisy counts. This is known as the Gaussian Sparse Histogram Mechanism. Karrer, Kifer, Wilkins, and Zhang Wilkins et al. [2024] gave an exact analysis of the parameters required for the mechanism to satisfy (ε, δ)-DP. They consider a more general version of the mechanism, and we restate their result for our setting in Theorem 80.

Theorem 80 (Gaussian Sparse Histogram Mechanism (GSHM) - Following [Wilkins et al., 2024, Theorem 5.4])**.** *Let T, c and T', c' be frequency sketches for streams S and S', such that for any neighboring pair $S \sim S'$, the counters c and c' differ for at most l counts and agree on the count for all other elements. The differing counts are either (a) all 1 higher in c than in c' or (b) all 1 lower in c than in c'. Then the Gaussian Sparse Histogram Mechanism, denoted* $\mathrm{GSHM}(T, c, l, \sigma, \tau)$, *is the mechanism that adds noise from $\mathcal{N}(0, \sigma^2)$ independently to each non-zero count in c and removes all noisy counts below the threshold $1 + \tau$. The mechanism satisfies (ε, δ)-differential privacy if and only if the following inequality holds*

$$\begin{aligned}\max\Bigg[&1 - \Phi\left(\frac{\tau}{\sigma}\right)^{l}, \\ &\max_{j\in[l]} 1 - \Phi\left(\frac{\tau}{\sigma}\right)^{l-j} + \Phi\left(\frac{\tau}{\sigma}\right)^{l-j}\left[\Phi\left(\frac{\sqrt{j}}{2\sigma} - \frac{(\varepsilon-\gamma)\sigma}{\sqrt{j}}\right) - e^{\varepsilon-\gamma}\Phi\left(-\frac{\sqrt{j}}{2\sigma} - \frac{(\varepsilon-\gamma)\sigma}{\sqrt{j}}\right)\right], \\ &\max_{j\in[l]} \Phi\left(\frac{\sqrt{j}}{2\sigma} - \frac{(\varepsilon+\gamma)\sigma}{\sqrt{j}}\right) - e^{\varepsilon+\gamma}\Phi\left(-\frac{\sqrt{j}}{2\sigma} - \frac{(\varepsilon+\gamma)\sigma}{\sqrt{j}}\right)\Bigg] \le \delta,\end{aligned}$$

where $\gamma = (l - j)\log\Phi\left(\frac{\tau}{\sigma}\right)$.

The inequality of Theorem 80 can be difficult to parse. We present simpler parameters for the Gaussian Sparse Histogram Mechanism below. Note that our analysis is far from tight and a deployment of the GSHM should preferably set parameters using the exact analysis presented in Theorem 80. We present the version with worse parameters only because it is easier to read off the asymptotic behavior of the mechanism.

Lemma 81. *Let T, c, T', c' and* $\mathrm{GSHM}(T, c, l, \sigma, \tau)$ *all be defined as in Theorem 80.* $\mathrm{GSHM}(T, c, l, \sigma, \tau)$ *satisfies (ε, δ)-differential privacy when* $\varepsilon < 1$ *for* $\sigma = \sqrt{l 2 \ln(2.5/\delta)}/\varepsilon$ *and* $\tau = \sqrt{2 \ln(2l/\delta)}\sigma$.

Proof. We use a proof similar to that currently used for the Google Differential Privacy library Google Anonymization Team where the budget for δ is split between the noise and the threshold. Let E_0 denote the event where only counters that are in both sketches are above the threshold. We use $\mathcal{M}(x)$ and $\mathcal{M}(x')$ to denote $\mathrm{GSHM}(T, c, l, \sigma, \tau)$ and $\mathrm{GSHM}(T', c', l, \sigma, \tau)$, respectively. We have that

$$\begin{aligned}\Pr[\mathcal{M}(x) \in Z] &= \Pr[\mathcal{M}(x) \in Z | E_0] \Pr[E_0] + \Pr[\mathcal{M}(x) \in Z | \neg E_0] \Pr[\neg E_0] \\ &\leq \Pr[\mathcal{M}(x) \in Z | E_0] \Pr[E_0] + \delta/2 \\ &\leq (e^{\varepsilon} \Pr[\mathcal{M}(x') \in Z | E_0] + \delta/2) \Pr[E_0] + \delta/2 \\ &\leq e^{\varepsilon} \Pr[\mathcal{M}(x') \in Z \text{ and } E_0] + \delta/2 + \delta/2 \\ &\leq e^{\varepsilon} \Pr[\mathcal{M}(x') \in Z] + \delta \ .\end{aligned}$$

The first inequality follows from a union bound and the fact that $\Pr[\mathcal{N}(0, \sigma^2) > t] \leq e^{-(t/\sigma)^2/2}$ for any positive t. The second inequality follows from [Dwork and Roth, 2014, Theorem A.1] since $\Delta_2 = \sqrt{l}$. □

As seen above the noise and threshold for the Gaussian Sparse Histogram Mechanism scales with the ℓ_2-sensitivity rather than the ℓ_1-sensitivity which is the case for the Laplace noise. Adding or removing a user to or from the stream only changes the true frequency of any $x \in \mathcal{U}$ by 1. As such the ℓ_2 distance between the frequencies for any pair of neighboring streams $S \sim S'$ is $\sqrt{\sum_{x \in \mathcal{U}} (f(x) - f(x'))^2} \leq \sqrt{m}$. We might hope that the ℓ_2-sensitivity of the MG sketch is low, or perhaps that we can use a technique similar to Algorithm 26 with Gaussian noise. Alternatively, we could remove small values using the post-processing from Section 8 if that guaranteed low ℓ_2-sensitivity. Unfortunately, that is not the case because there exist neighboring streams where a single counter differs by m. The statement is true even if we perform the post-processing from Section 8.

Lemma 82. *Let T, c and T', c' denote MG sketches of size k for a pair of neighboring streams $S \sim S'$. There exist neighboring streams such that $c_x - c'_x = m$ for some element $x \in \mathcal{U}$.*

Proof. There are many such pairs of streams, here we give an example of a pair where all counters for elements other than x are zero in both sketches. Let S_{k+1} be the user that is removed from S to obtain S'. Let $(S_1, S_2, \ldots, S_{k-1}, S_k)$

contain exactly m copies of k distinct elements that does not include x. This is always possible for any $m \leq k$ by cycling through the same order of k elements and taking m elements at a time. E.g, for $m = 2$ and items $\{1, 2, 3\}$ the stream could be $(\{1, 2\}, \{3, 1\}, \{2, 3\})$. If S_{k+1} contains m elements not in the sketch all counters are decremented and the sketch for S is empty after processing S_{k+1}. Now, the rest of the stream contains only copies of the singleton $\{x\}$. After processing S_{k+1+m} the sketch for S has a single counter $c_x = m$ while the sketch for S' is empty. Furthermore, if we extend the streams further we have $c_x = |S| - k - 1$ and $c'_x = |S| - k - 1 - m$. □

Since there exist neighboring streams for which the MG sketch differs by m for a single counter we must add noise with magnitude scaled to m to satisfy differential privacy. We also cannot hope to avoid these problematic inputs using some post-processing on small counters similar to Section 8 because the counter that differs by m can be arbitrarily large for long streams. We must use a different sketching algorithm to remove the linear dependency on m. In Algorithm 27 we present a novel frequency sketch. Our sketch is similar in spirit to the MG sketch as it is a generalization of the original MG sketch for sets. The key difference is that we always increment all counters for the elements of a user and decrement at most once per user instead of once per element.

Algorithm 27: Privacy-Aware Misra-Gries Sketch (PAMG)

```
Input: Positive integer $k$ and stream
       $S = (S_1, S_2, \ldots, S_{n-1}, S_n)$
$T \leftarrow \emptyset$
foreach $S_i \in S$ do
    foreach $x \in S_i$ do
        if $x \in T$ then
            $c_x \leftarrow c_x + 1$
        else
            $T \leftarrow T \cup \{x\}$
            $c_x \leftarrow 1$
    if $|T| > k$ then
        foreach $x \in T$ do
            $c_x \leftarrow c_x - 1$
            if $c_x = 0$ then
                $T \leftarrow T \setminus \{x\}$
return $T, c$
```

We start our analysis by showing that the error guarantees of Algorithm 27 match the MG sketch.

Lemma 83. *Let $\hat{f}(x)$ be the frequency estimates given by Algorithm 27 with size k for N being the input size. Then for all $x \in \mathcal{U}$, it holds that $\hat{f}(x) \in [f(x) - \lfloor N/(k+1) \rfloor, f(x)]$, where $f(x)$ is the true frequency of x in S.*

Proof. The proof is similar to the analysis of the MG sketch by Bose et al. [2003]. Notice that similar to the MG sketch, we only introduce errors when decrementing counters. The error of any estimate $\hat{f}(x)$ is exactly the number of times the counter for x was decremented on line 11. As such, we can bound the maximum error of all counters by the number of times the condition on line 9 evaluates to true. Notice that whenever the condition is true the sum of all counters is decreased by at least $k+1$. Since the sum is incremented exactly N times and counters are never negative this happens at most $\lfloor N/(k+1) \rfloor$ times. □

Next, we bound the sensitivity of Algorithm 27.

Lemma 84. *Let $T, c \leftarrow \mathrm{PAMG}(k, S)$ and $T', c' \leftarrow \mathrm{PAMG}(k, S')$ be the outputs of Algorithm 27 on a pair of neighboring streams $S \sim S'$. Then it holds that either, (1) $T' \subseteq T$ and $c_i - c'_i = \{0, 1\}$ for all $i \in T$ or (2) $T \subseteq T'$ and $c'_i - c_i = \{0, 1\}$ for all $i \in T'$.*

Proof. We first show that the condition holds after processing the user that is only present in one of the streams. Without loss of generality consider the case where S' is obtained by removing S_i from S. Since the two cases of the lemma are symmetric the proof is the same if S is obtained by removing some user S'_i from S'. Since the sketch is deterministic the sketches for the two streams are identical after processing S_{i-1}. The sketch for S then processes S_i. After running the loop on Lines 3-8 the count for each element in S_i is 1 higher compared to the sketch for S'. If the condition on Line 9 evaluates to false we finished processing S_i and the state corresponds to case (1) of the lemma. If the condition is true we decrement all counters by 1 and remove elements with a count of 0. The state then corresponds to case (2) of the lemma.

We now show that if one of the conditions holds before processing some user S_j one of the conditions holds after processing S_j. This proves the lemma by induction. Assume that case (1) of the lemma holds before processing S_j. The proof is identical for case (2) due to symmetry. In each iteration of the loop on Lines 3-8 we increment the same counter in both sketches which does not affect the condition of case (1). As such, we only have to consider the effect of decrementing counters. If we either do not decrement the counters in any sketch or we decrement the counters in both sketches the state after processing S_j still corresponds to case (1). However, if $|T| > k \geq |T'|$ we only decrement the counters in c. The state then corresponds to case (2). Note that we never decrement the counters in c' without decrementing the counters in c since we have that $|T| \geq |T'|$. □

As mentioned in Section 8 it is often important in practice that we can merge sketches. A nice property of Algorithm 27 is that the sensitivity has the structure we used to bound the sensitivity of merged sketches. The PAMG sketch can be seen as a special case of the merging algorithm discussed in Section 8. As such, merging the sketches from Algorithm 27 does not increase sensitivity.

Corollary 85. *Let $(S^{(1)}, \dots, S^{(l)})$ and $(S'^{(1)}, \dots, S'^{(l)})$ denote sets of streams such that $S^{(i)} \sim S'^{(i)}$ for one $i \in [l]$ and $S^{(j)} = S'^{(j)}$ for any $j \neq i$. Let T, c and T', c' be the result of merging* PAMG *sketches computed on both sets of streams in any fixed order using the merging algorithm from Section 8. Then it holds that either, (1) $T' \subseteq T$ and $c_i - c'_i = \{0, 1\}$ for all $i \in T$ or (2) $T \subseteq T'$ and $c'_i - c_i = \{0, 1\}$ for all $i \in T'$.*

Proof. It holds by induction and Lemma 74. The condition required by Lemma 74 holds for sketches of any pair of neighboring streams by Lemma 84. □

Lemma 86. *Let T, c be the result of merging* PAMG *sketches of size k computed on a set of streams using the merging algorithm from Section 8. Let $f(x)$ be the true frequency of x across all streams and let M be the total number of elements across all streams. Then for all $x \in \mathcal{U}$ it holds that*

$$c_x \in \left[f(x) - \frac{M}{k+1}, f(x) \right]$$

Proof. Agarwal et al. Agarwal et al. [2013] show that this holds for the MG sketch in Lemma 1 and Theorem 1 of their paper. They used the properties of the MG sketch that error is only introduced when decrementing counters, and counters for $k+1$ distinct elements are decremented each time. Since at least $k+1$ counters for distinct elements are decremented in Algorithm 27 the lemma follows from their proof. □

We are now ready to state our theorem for this section.

Theorem 87. *Let $(S^{(1)}, \dots, S^{(l)})$ and $(S'^{(1)}, \dots, S'^{(l)})$ denote sets of streams such that $S^{(i)} \sim S'^{(i)}$ for one $i \in [l]$ and $S^{(j)} = S'^{(j)}$ for any $j \neq i$. Let T, c and T', c' be the result of merging* PAMG *sketches of size k computed on both sets of streams in any fixed order using the merging algorithm from Section 8. Then releasing the output of* GSHM(T, c, k, σ, τ) *where σ and τ are chosen according to Theorem 80 satisfies (ε, δ)-differential privacy. Let $\hat{f}(x)$ be the estimate of the true frequency $f(x)$ across all streams and let M denote the* *total number of elements across all streams. Then we have for all* $x \in \mathcal{U}$ *with probability at least* $1 - 2\delta$ *that*

$$\hat{f}(x) \in \left[f(x) - \frac{M}{k+1} - 2\tau - 1, f(x) + \tau \right]$$

Proof. The privacy guarantees follow directly from Corollary 85 and Theorem 80. The three sources of error are the estimation error of the non-private sketch, the noise added to each counter, and the error from removing values below the threshold. The error of the merged sketch is at most $\frac{M}{k+1}$ by Lemma 86. From the first part of the condition of Theorem 80 we have that $1 - \Phi(\frac{\tau}{\sigma})^k \leq \delta$. That is, the k samples of Gaussian noise are all bounded by τ with probability at least $1 - \delta$. By symmetry and a union bound the absolute value of the noise is bounded by τ with probability at least $1 - 2\delta$. Removing noisy counters below the thresholding adds error at most $1 + \tau$. □

Open Problems

In Section 8 we introduced a frequency sketch for a stream of users each with up to m elements. Our sketch has error guarantees similar to the MG sketch and the ℓ_2-sensitivity of the sketch with size k is $\sqrt{k}$. The main open problem that we leave is if there exists a sketch with similar error guarantees and ℓ_2-sensitivity of $\sqrt{m}$ or $O(\sqrt{m})$. This would allow us to add noise to the sketch with magnitude matching the non-streaming setting. The sensitivity of our sketch exceeds $O(\sqrt{m})$ because the number of elements that are decremented when the sketch is full can differ between neighboring sketches. A counter-based sketch with low sensitivity likely must ensure that the number of decremented counters remains stable. We experimented with some variants that decremented a fixed number of elements. Unfortunately, they had higher sensitivity than our sketch as they did not have the property that all counts differ by at most 1.

Bibliography

Get followers/ids — docs — twitter developer platform. URL https://developer.twitter.com/en/docs/twitter-api/v1/accounts-and-users/follow-search-get-users/api-reference/get-followers-ids.

Raghavendra Addanki, Andrew McGregor, and Cameron Musco. Non-adaptive edge counting and sampling via bipartite independent set queries. *arXiv preprint arXiv:2207.02817*, 2022.

Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Trans. Database Syst.*, 38(4), dec 2013. ISSN 0362-5915. doi: 10.1145/2500128. URL https://doi.org/10.1145/2500128.

Maryam Aliakbarpour, Amartya Shankha Biswas, Themis Gouleakis, John Peebles, Ronitt Rubinfeld, and Anak Yodpinyanee. Sublinear-Time Algorithms for Counting Star Subgraphs via Edge Sampling. *Algorithmica*, 80 (2):668–697, feb 2018a. ISSN 14320541. doi: 10.1007/s00453-017-0287-3.

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

Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999. doi: 10.1017/CBO9780511624216.

Apple. Differential privacy overview - apple. https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf. [Online; accessed 12-August-2023].

AC Armenakis, LE Garey, and RD Gupta. An adaptation of a root finding method to searching ordered disk files. *BIT Numerical Mathematics*, 25(4): 561–568, 1985.

Sepehr Assadi. Lecture 3. URL https://people.cs.rutgers.edu/~sa1497/courses/cs514-s20/lec3.pdf.

Sepehr Assadi. CS 514: Advanced Algorithms II-Sublinear Algorithms 1 Sublinear Time Algorithms for Graphs. Technical report, Rutgers University, 2020. URL https://www.cs.rutgers.edu/{~}sa1497/courses/cs514-s20/lec3.pdf.

Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. In *Innovations in Theoretical Computer Science Conference ITCS*, volume 124 of *LIPIcs*, pages 6:1–6:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019a.

Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A Simple Sublinear-Time Algorithm for Counting Arbitrary Subgraphs via Edge Sampling. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, volume 124 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:20, Dagstuhl, Germany, 2019b. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-095-8. doi: 10.4230/LIPIcs.ITCS.2019.6. URL https://drops-dev.dagstuhl.de/entities/document/10.4230/LIPIcs.ITCS.2019.6.

Martin Aumüller, Christian Janos Lebeda, and Rasmus Pagh. Representing sparse vectors with differential privacy, low error, optimal space, and fast access. *Journal of Privacy and Confidentiality*, 12(2), Nov. 2022. doi: 10.29012/jpc.809. URL https://journalprivacyconfidentiality.org/index.php/jpc/article/view/809.

Victor Balcer and Salil Vadhan. Differential privacy on finite computers. *Journal of Privacy and Confidentiality*, 9(2), Sep. 2019. doi: 10.29012/jpc.679. URL https://journalprivacyconfidentiality.org/index.php/jpc/article/view/679.

Daniel Barth-Jones. The're-identification'of governor william weld's medical information: a critical re-examination of health data identification risks and privacy protections, then and now. *Then and Now (July 2012)*, 2012.

Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Guha Thakurta. Practical locally private heavy hitters. *Advances in Neural Information Processing Systems*, 30, 2017.

Omri Ben-Eliezer, Talya Eden, Joel Oren, and Dimitris Fotakis. Sampling multiple nodes in large networks: Beyond random walks, 2021.

Suman K Bera and C Seshadhri. How to count triangles, without seeing the whole graph. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 306–316, 2020.

Lorenzo Beretta and Jakub Tětek. Better sum estimation via weighted sampling. *ACM Trans. Algorithms*, mar 2024. ISSN 1549-6325. doi: 10.1145/3650030. URL https://doi.org/10.1145/3650030. Just Accepted.

Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Faster counting and sampling algorithms using colorful decision oracle. In *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

Amartya Shankha Biswas, Talya Eden, and Ronitt Rubinfeld. Towards a decomposition-optimal algorithm for counting and sampling arbitrary motifs in sublinear time. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, to appear*, 2021a.

AS Biswas, T Eden, and R Rubinfeld. Towards a decomposition-optimal algorithm for counting and sampling arbitrary motifs in sublinear time. In *Random*, 2021b.

Jeremiah Blocki, Elena Grigorescu, Tamalika Mukherjee, and Samson Zhou. How to make your approximation algorithm private: A black-box differentially-private transformation for tunable approximation algorithms of functions with low sensitivity. *arXiv preprint arXiv:2210.03831*, 2022.

Jonas Böhler and Florian Kerschbaum. Secure multi-party computation of differentially private heavy hitters. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 2361–2377. ACM, 2021. doi: 10.1145/3460120.3484557. URL https://doi.org/10.1145/3460120.3484557.

Prosenjit Bose, Evangelos Kranakis, Pat Morin, and Yihui Tang. Bounds for frequency estimation of packet streams. In Jop F. Sibeyn, editor, *SIROCCO 10: Proceedings of the 10th Internaltional Colloquium on Structural Information Complexity, June 18-20, 2003, Umeå Sweden*, volume 17 of *Proceedings in Informatics*, pages 33–42. Carleton Scientific, 2003.

Mark Bun, Jelani Nelson, and Uri Stemmer. Heavy hitters and the structure of local privacy. *ACM Transactions on Algorithms (TALG)*, 15(4):1–40, 2019.

Clément Canonne and Ronitt Rubinfeld. Testing probability distributions underlying aggregated data. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, pages 283–295, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-43948-7.

Clément L Canonne. *Topics and techniques in distribution testing.* Now Publishers, 2022.

Ricardo Silva Carvalho, Ke Wang, Lovedeep Gondara, and Chunyan Miao. Differentially private top-k selection via stability on unknown domain. In *Conference on Uncertainty in Artificial Intelligence*, pages 1109–1118. PMLR, 2020.

T-H Hubert Chan, Mingfei Li, Elaine Shi, and Wenchang Xu. Differentially private continual monitoring of heavy hitters from distributed streams. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 140–159. Springer, 2012.

Flavio Chiericetti, Anirban Dasgupta, Ravi Kumar, Silvio Lattanzi, and Tamás Sarlós. On sampling nodes in a network. In *Proceedings of the 25th International Conference on World Wide Web*, pages 471–481, 2016.

Flavio Chierichetti and Shahrzad Haddadan. On the complexity of sampling vertices uniformly from a graph. In *Leibniz International Proceedings in Informatics, LIPIcs*, volume 107. Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, jul 2018. ISBN 9783959770767. doi: 10.4230/LIPIcs.ICALP.2018.149.

Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, and Thanh T. L. Tran. Differentially private summaries for sparse data. In *ICDT*, pages 299–311. ACM, 2012. doi: 10.1145/2274576.2274608.

Anirban Dasgupta, Ravi Kumar, and Tamas Sarlos. On estimating the average degree. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, page 795–806, New York, NY, USA, 2014a. Association for Computing Machinery. ISBN 9781450327442. doi: 10.1145/2566486.2568019. URL https://doi.org/10.1145/2566486.2568019.

Anirban Dasgupta, Ravi Kumar, and Tamas Sarlos. On estimating the average degree. In *Proceedings of the 23rd international conference on World wide web*, pages 795–806, 2014b.

Persi Diaconis and Susan Holmes. Lecture Notes – Monograph Series. Institute of Mathematical Statistics, 2004. ISBN 0-940600-62-5.

David Durfee and Ryan M Rogers. Practical differentially private top-k selection with pay-what-you-get composition. *Advances in Neural Information Processing Systems*, 32, 2019.

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4): 211–407, 2014. doi: 10.1561/0400000042.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, pages 265–284. Springer, 2006a.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, pages 265–284. Springer, 2006b.

Talya Eden and Will Rosenbaum. On sampling edges almost uniformly. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2018a.

Talya Eden and Will Rosenbaum. Lower Bounds for Approximating Graph Parameters via Communication Complexity. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*, volume 116 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:18, Dagstuhl, Germany, 2018b. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-085-9. doi: 10.4230/LIPIcs.APPROX-RANDOM.2018.11. URL http://drops.dagstuhl.de/opus/volltexte/2018/9415.

Talya Eden and Will Rosenbaum. On sampling edges almost uniformly. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2018c.

Talya Eden and Will Rosenbaum. On Sampling Edges Almost Uniformly. In Raimund Seidel, editor, *1st Symposium on Simplicity in Algorithms (SOSA 2018)*, volume 61 of *OpenAccess Series in Informatics (OASIcs)*, pages 7:1–7:9, Dagstuhl, Germany, 2018d. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-064-4. doi: 10.4230/OASIcs.SOSA.2018.7. URL http://drops.dagstuhl.de/opus/volltexte/2018/8300.

Talya Eden, Amit Levi, Dana Ron, and C. Seshadhri. Approximately Counting Triangles in Sublinear Time. *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2015-Decem:614–633, apr 2015. doi: 10.1109/FOCS.2015.44. URL http://arxiv.org/abs/1504.00954http://dx.doi.org/10.1109/FOCS.2015.44.

Talya Eden, Dana Ron, and C. Seshadhri. Sublinear time estimation of degree distribution moments: The degeneracy connection. In *Leibniz International Proceedings in Informatics, LIPIcs*, volume 80. Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, jul 2017a. ISBN 9783959770415. doi: 10.4230/LIPIcs.ICALP.2017.7.

Talya Eden, Dana Ron, and C. Seshadhri. Sublinear Time Estimation of Degree Distribution Moments: The Degeneracy Connection. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:13, Dagstuhl, Germany, 2017b. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-041-5. doi: 10.4230/LIPIcs.ICALP.2017.7. URL http://drops.dagstuhl.de/opus/volltexte/2017/7374.

Talya Eden, Dana Ron, and Will Rosenbaum. The Arboricity Captures the Complexity of Sampling Edges. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 52:1–52:14, Dagstuhl, Germany, 2019a. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-109-2. doi: 10.4230/LIPIcs.ICALP.2019.52. URL http://drops.dagstuhl.de/opus/volltexte/2019/10628.

Talya Eden, Dana Ron, and Will Rosenbaum. The Arboricity Captures the Complexity of Sampling Edges. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 52:1–52:14, Dagstuhl, Germany, 2019b. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-109-2. doi: 10.4230/LIPIcs.ICALP.2019.52. URL http://drops.dagstuhl.de/opus/volltexte/2019/10628.

Talya Eden, Saleet Mossel, and Ronitt Rubinfeld. Amortized Edge Sampling. aug 2020. URL http://arxiv.org/abs/2008.08032.

Talya Eden, Saleet Mossel, and Ronitt Rubinfeld. Sampling Multiple Edges Efficiently. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021)*, volume 207 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 51:1–51:15, Dagstuhl, Germany, 2021a. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-207-5. doi: 10.4230/LIPIcs.APPROX/RANDOM.2021.51. URL https://drops.dagstuhl.de/opus/volltexte/2021/14744.

Talya Eden, Saleet Mossel, and Ronitt Rubinfeld. Sampling multiple edges efficiently. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021b.

Talya Eden, Shyam Narayanan, and Jakub Tětek. *Sampling an Edge in Sublinear Time Exactly and Optimally*, pages 253–260. 2023. doi: 10.1137/1.9781611977585.ch23. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611977585.ch23.

Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.

Uriel Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM J. Comput.*, 35: 964–984, 01 2006a. doi: 10.1137/S0097539704447304.

Uriel Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing*, 35(4):964–984, 2006b.

Hendrik Fichtenberger, Mingze Gao, and Pan Peng. Sampling Arbitrary Subgraphs Exactly Uniformly in Sublinear Time. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 45:1–45:13, Dagstuhl, Germany, 2020a. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISBN 978-3-95977-138-2. doi: 10.4230/LIPIcs.ICALP.2020.45. URL https://drops.dagstuhl.de/opus/volltexte/2020/12452.

Hendrik Fichtenberger, Mingze Gao, and Pan Peng. Sampling arbitrary subgraphs exactly uniformly in sublinear time. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 45:1–45:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020b. doi: 10.4230/LIPIcs.ICALP.2020.45. URL https://doi.org/10.4230/LIPIcs.ICALP.2020.45.

Quan Geng, Peter Kairouz, Sewoong Oh, and Pramod Viswanath. The staircase mechanism in differential privacy. *IEEE Journal of Selected Topics in Signal Processing*, 9(7):1176–1184, 2015. doi: 10.1109/JSTSP.2015.2425831.

Badih Ghazi, Noah Golowich, Ravi Kumar, Rasmus Pagh, and Ameya Velingker. On the power of multiple anonymous messages. *IACR Cryptol. ePrint Arch.*, page 1382, 2019. URL https://eprint.iacr.org/2019/1382.

Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM Journal on Computing*, 41 (6):1673–1693, 2012.

Oded Goldreich. *Introduction to property testing*. Cambridge University Press, Cambridge, United Kingdom ; New York, NY, USA, 2018. ISBN 9781107194052.

Oded Goldreich and Dana Ron. Approximating average parameters of graphs. In Josep Díaz, Klaus Jansen, José D. P. Rolim, and Uri Zwick, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 363–374, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-38045-0.

Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random Structures and Algorithms*, 32(4):473–493, jul 2008. ISSN 10429832. doi: 10.1002/rsa.20203. URL http://doi.wiley.com/10.1002/rsa.20203.

Google Anonymization Team. Delta for thresholding. https://github.com/google/differential-privacy/blob/main/common_docs/Delta_For_Thresholding.pdf. [Online; accessed 15-April-2024].

Samuel Haney, Damien Desfontaines, Luke Hartman, Ruchit Shrestha, and Michael Hay. Precision-based attacks and interval refining: how to break, then fix, differential privacy on finite computers. *CoRR*, abs/2207.13793, 2022. doi: 10.48550/arXiv.2207.13793. URL https://doi.org/10.48550/arXiv.2207.13793.

D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685, 1952a. ISSN 01621459. URL http://www.jstor.org/stable/2280784.

D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685, 1952b. ISSN 01621459. URL http://www.jstor.org/stable/2280784.

Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, page 1–10, New York, NY, USA, 1977. Association for Computing Machinery. ISBN 9781450374095. doi: 10.1145/800105.803390. URL https://doi.org/10.1145/800105.803390.

Christian Janos Lebeda and Jakub Tetek. Better differentially private approximate histograms and heavy hitters using the misra-gries sketch. *SIGMOD Rec.*, 53(1):7–14, may 2024. ISSN 0163-5808. doi: 10.1145/3665252.3665255. URL https://doi.org/10.1145/3665252.3665255.

John Kallaugher, Andrew McGregor, Eric Price, and Sofya Vorotnikova. The complexity of counting cycles in the adjacency list streaming model. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '19, page 119–133, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362276. doi: 10.1145/3294052.3319706. URL https://doi.org/10.1145/3294052.3319706.

Liran Katzir, Edo Liberty, and Oren Somekh. Estimating sizes of social networks via biased sampling. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, page 597–606, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306324. doi: 10.1145/1963405.1963489. URL https://doi.org/10.1145/1963405.1963489.

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

Mihail N. Kolountzakis, Gary L. Miller, Richard Peng, and Charalampos E. Tsourakakis. 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 https://doi.org/10.1080/15427951.2012.625260.

Aleksandra Korolova, Krishnaram Kenthapadi, Nina Mishra, and Alexandros Ntoulas. Releasing search queries and clicks privately. In *WWW*, pages 171–180. ACM, 2009. doi: 10.1145/1526709.1526733.

John A Lapinskas, Holger Dell, and Kitty Meeks. Approximately counting and sampling small witnesses using a colourful decision oracle. In *ACM-SIAM Symposium on Discrete Algorithms (SODA20)*, 2019.

Richard J. Lipton, Jeffrey F. Naughton, Donovan A. Schneider, and S. Seshadri. Efficient sampling strategies for relational database operations. *Theoretical Computer Science*, 116(1):195–226, 1993. ISSN 0304-3975. doi: https://doi.org/10.1016/0304-3975(93)90224-H. URL https://www.sciencedirect.com/science/article/pii/030439759390224H.

Ryan MacCarthy. The average twitter user now has 707 followers, Jun 2016. URL https://kickfactory.com/blog/average-twitter-followers-updated-2016/.

Darakhshan J. Mir, S. Muthukrishnan, Aleksandar Nikolov, and Rebecca N. Wright. Pan-private algorithms via statistics on sketches. In Maurizio Lenzerini and Thomas Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 37–48. ACM, 2011. doi: 10.1145/1989284.1989290. URL https://doi.org/10.1145/1989284.1989290.

J. Misra and David Gries. Finding repeated elements. *Science of Computer Programming*, 2(2):143–152, 1982. ISSN 0167-6423. doi: https://doi.org/10.1016/0167-6423(82)90012-0. URL https://www.sciencedirect.com/science/article/pii/0167642382900120.

Rajeev Motwani, Rina Panigrahy, and Ying Xu. Estimating sum by weighted sampling. In *Proceedings of the 34th International Conference on Automata, Languages and Programming*, ICALP'07, page 53–64, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3540734198.

Şerban Nacu and Yuval Peres. Fast simulation of new coins from old. *SIAM Journal on Computing*, 35(4):964–984, 2006.

Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the netflix prize dataset. *arXiv preprint cs/0610105*, 2006.

Joseph P Near and Chiké Abuah. Programming differential privacy. *URL: https://uvm*, 2021.

Krzysztof Onak and Xiaorui Sun. Probability-revealing samples. In *AISTATS*, 2018.

Rasmus Pagh and Mikkel Thorup. Improved utility analysis of private countsketch. In *Advances in Neural Information Processing Systems*, volume 35, pages 25631–25643, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/a47f5cdff1469751597d78e803fc590f-Paper-Conference.pdf.

W Wesley Peterson. Addressing for random-access storage. *IBM journal of Research and Development*, 1(2):130–146, 1957.

Gang Qiao, Weijie Su, and Li Zhang. Oneshot differentially private top-k selection. In *International Conference on Machine Learning*, pages 8672–8681. PMLR, 2021.

Zhan Qin, Yin Yang, Ting Yu, Issa Khalil, Xiaokui Xiao, and Kui Ren. Heavy hitter estimation over set-valued data with local differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 192–203, 2016.

Seagate. Seagate® desktop HDD, ST4000DM000, ST3000DM003, 2014. URL https://www.seagate.com/www-content/product-content/desktop-hdd-fam/en-us/docs/100710254f.pdf.

C Seshadhri. A simpler sublinear algorithm for approximating the triangle count. *arXiv preprint arXiv:1505.01927*, 2015.

Jakub Tětek. Additive noise mechanisms for making randomized approximation algorithms differentially private. *arXiv preprint arXiv:2211.03695*, 2022.

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

Toshiba. Enterprise hard drives MG series, 2019. URL https://www.toshiba-storage.com/wp-content/uploads/2019/09/TOSH_DS_MG_Series_print.pdf.

Jakub Tětek. Approximate Triangle Counting via Sampling and Fast Matrix Multiplication. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 107:1–107:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-235-8. doi: 10.4230/LIPIcs.ICALP.2022.107. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2022.107.

Jakub Tětek and Mikkel Thorup. Edge sampling and graph parameter estimation via vertex neighborhood accesses. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 1116–1129, New York, NY, USA, 2022a. Association for Computing Machinery. ISBN 9781450392648. doi: 10.1145/3519935.3520059. URL https://doi.org/10.1145/3519935.3520059.

Jakub Tětek and Mikkel Thorup. Edge sampling and graph parameter estimation via vertex neighborhood accesses. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 1116–1129, New York, NY, USA, 2022b. Association for Computing Machinery. ISBN 9781450392648. doi: 10.1145/3519935.3520059. URL https://doi.org/10.1145/3519935.3520059.

Jakub Tětek. Approximate Triangle Counting via Sampling and Fast Matrix Multiplication. *arXiv:2104.08501 [cs]*, May 2021. URL http://arxiv.org/abs/2104.08501. arXiv: 2104.08501.

Tianhao Wang, Ninghui Li, and Somesh Jha. Locally differentially private heavy hitter identification. *IEEE Transactions on Dependable and Secure Computing*, 18(2):982–993, 2019.

Osamu Watanabe. Sequential sampling techniques for algorithmic learning theory. *Theoretical Computer Science*, 348(1):3–14, 2005. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2005.09.003. URL https://www.sciencedirect.com/science/article/pii/S0304397505005219. Algorithmic Learning Theory (ALT 2000).

Arjun Wilkins, Daniel Kifer, Danfeng Zhang, and Brian Karrer. Exact privacy analysis of the gaussian sparse histogram mechanism. *Journal of Privacy and Confidentiality*, 14(1), Feb. 2024. doi: 10.29012/jpc.823. URL https://journalprivacyconfidentiality.org/index.php/jpc/article/view/823.

Hao Wu and Anthony Wirth. Asymptotically optimal locally private heavy hitters via parameterized sketches. In *International Conference on Artificial Intelligence and Statistics*, pages 7766–7798. PMLR, 2022.

Dan Zhao, Suyun Zhao, Hong Chen, Ruixuan Liu, Cuiping Li, and Wenjuan Liang. Efficient protocols for heavy hitter identification with local differential privacy. *Frontiers of Computer Science*, 16(5):1–11, 2022a.

Fuheng Zhao, Dan Qiao, Rachel Redberg, Divyakant Agrawal, Amr El Abbadi, and Yu-Xiang Wang. Differentially private linear sketches: Efficient implementations and applications. In *Advances in Neural Information Processing Systems*, volume 35, pages 12691–12704, 2022b. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/525338e0d98401a62950bc7c454eb83d-Paper-Conference.pdf.

Wennan Zhu, Peter Kairouz, Brendan McMahan, Haicheng Sun, and Wei Li. Federated heavy hitters discovery with differential privacy. In *International Conference on Artificial Intelligence and Statistics*, pages 3837–3847. PMLR, 2020.