Sampling an Edge in Sublinear Time Exactly and Optimally

Talya Eden* Shyam Narayanan† Jakub Tětek‡

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.

*Bar Ilan University, talyaa01@gmail.com, supported by the NSF TRIPODS program, award CCF-1740751. This work was partially done while affiliated with MIT and Boston University.

†MIT, shyamsn@mit.edu, supported by the NSF GRFP Fellowship and the NSF TRIPODS Program (award DMS-2022448).

‡BARC, Univ. of Copenhagen, j.tetek@gmail.com, supported by the VILLUM Foundation grant 16582. This work was partially done while visiting MIT.

1 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 [13]. 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 [8] 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. [1]

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 [9], only adding a constant to the failure probability. The algorithm from [9] 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

[1] 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.

knowing m a priori.

1.1 Technical overview The starting point of our algorithm is the algorithm by Eden and Rosenbaum [6], which we now shortly recall.

The algorithm by Eden and Rosenbaum [6]. 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 [6], θ is set to $\sqrt{2m/\varepsilon}$ which implies that for every heavy vertex v, $d_\ell(v) \in [(1-\varepsilon)d(v), d(v)]$.[2] 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 [12].

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 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

[2] 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, $d_\ell(v) > (1-\varepsilon)d(v)$.

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 [12]. 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 [12] with the function $1/(4x)$, thus simulating $Bern(1/(4p/2)) = Bern(1/(2p))$.

1.2 Related work Using uniform edge samples as a basic query in the sublinear time setting was first suggested by Aliakbarpour et al. [2] 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 [3, 10, 5, 13].

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 [6] who gave an algorithm with complexity $O\left(n/\sqrt{\varepsilon m}\right)$. This was later improved by Tětek and Thorup [13] 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. In [8], Eden, Mossel, and Rubinfeld 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 [13]. In [7], Eden, Ron and Rosenbaum 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 [11, 4, 1].

2 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.

3 Sampling an edge

In this section, we give our algorithm for sampling an edge. We start by stating a result of Nacu and Peres [12] about "Bernoulli factories". We recall that a function f from a closed interval I to

Algorithm 1: Sample an edge pointwise $\Theta(1)$-close to uniform

$u \leftarrow$ uniformly random vertex
$j \leftarrow Unif([\theta])$, where $\theta = \lceil\sqrt{6m}\rceil$.
Fail if $d(v) > \theta$ or $d(v) \leq j$
$v \leftarrow j$-th neighbor of u
$B \sim Bern(1/3)$
if $B = 1$ **then**
  return uv
else if $B = 0$ *and* v *is heavy* **then**
  $w \leftarrow$ random neighbor of v
  return vw
end
return *Fail*

$\mathbb{R}$ 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 3.1. *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 3.1 gives us the following corollary.

COROLLARY 3.1. *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.)*

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 3.1 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 [6] but uses the Bernoulli factory of [12] to reduce the sampling error in a significantly more efficient way than in [6]. 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 3.1. *Let e be the edge returned by Algorithm 1 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, $\Pr[e = e'] = \frac{1}{n} \cdot \frac{1}{\lceil\sqrt{6m}\rceil} \cdot \frac{1}{3} = \frac{1}{3n\theta}$.

Algorithm 2: Sample an edge $1 \pm \varepsilon$-pointwise-close to uniform

```
repeat
    vw ← Algorithm 1
    if v is light then
        return vw
    end
    if v is heavy then
        Let w_1, ..., be random neighbors of v
        Y ← use Corollary 3.1 on Bernoulli trials defined as B_i = [d(w_i) ≤ √(6m)]
        if Y = 0 then
            return vw
        end
    end
end
```

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 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 1 returns a heavy edge (which has some bias), we use rejection sampling based on Bernoulli factories to reduce the bias.

THEOREM 3.2. *Algorithm 2 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 3.1, each invocation of Algorithm 1 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 1 returns a heavy edge vw, then for every w_i sampled in Step 7 in Algorithm 2, it holds that the indicator of the event $[d(w_i) \le \lceil\sqrt{6m}\rceil]$ 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 $deg_H(v) \le |H| \le \frac{2m}{\lceil\sqrt{6m}\rceil} \le \sqrt{\frac{2}{3}m} \le \frac{1}{3}d(v)$, 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 3.1, the value Y returned by Algorithm 2 has distribution $Y \sim \text{Bern}(\frac{1}{2p})$. 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 $m \cdot \frac{1}{3n\theta} \ge \frac{\sqrt{m}}{10n}$. 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 1 once, and Corollary 3.1 at most once. Algorithm 1 clearly takes a constant number of queries. If Algorithm 1 returns a heavy edge, then sampling the w_i neighbors in Step 7 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 [12].

References

[1] Raghavendra Addanki, Andrew McGregor, and Cameron Musco. Non-adaptive edge counting and sampling via bipartite independent set queries. *arXiv preprint arXiv:2207.02817*, 2022.

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

[3] 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, 2019.

[4] 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.

[5] 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*, 2021.

[6] 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, 2018. 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`.

[7] 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, 2019. 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`.

[8] 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, 2021.

[9] Uriel Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing*, 35(4):964–984, 2006.

[10] 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, 2020. doi: 10.4230/LIPIcs.ICALP.2020.45. URL https://doi.org/10.4230/LIPIcs.ICALP.2020.45.

[11] 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.

[12] Şerban Nacu and Yuval Peres. Fast simulation of new coins from old. *SIAM Journal on Computing*, 35(4):964–984, 2006.

[13] 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, 2022. Association for Computing Machinery. ISBN 9781450392648. doi: 10.1145/3519935.3520059. URL https://doi.org/10.1145/3519935.3520059.