# Efficient and Near-optimal Algorithms for Sampling Small Connected Subgraphs

MARCO BRESSAN, Università degli Studi di Milano, Italy

We study the following problem: Given an integer $k \geq 3$ and a simple graph $G$, sample a connected induced $k$-vertex subgraph of $G$ uniformly at random. This is a fundamental graph mining primitive with applications in social network analysis, bioinformatics, and more. Surprisingly, no efficient algorithm is known for uniform sampling; the only somewhat efficient algorithms available yield samples that are only approximately uniform, with running times that are unclear or suboptimal. In this work, we provide: (i) a near-optimal mixing time bound for a well-known random walk technique, (ii) the first efficient algorithm for truly uniform graphlet sampling, and (iii) the first sublinear-time algorithm for $\varepsilon$-uniform graphlet sampling.

CCS Concepts: • **Theory of computation → Streaming, sublinear and near linear time algorithms**; **Random walks and Markov chains**; • **Mathematics of computing → Graph algorithms;**

Additional Key Words and Phrases: Subgraph sampling, random walks, sublinear algorithms

**ACM Reference format:**
Marco Bressan. 2023. Efficient and Near-optimal Algorithms for Sampling Small Connected Subgraphs. *ACM Trans. Algor.* 19, 3, Article 26 (June 2023), 40 pages.
https://doi.org/10.1145/3596495

## 1 INTRODUCTION

A $k$-graphlet of a graph $G$ is a connected and induced $k$-vertex subgraph of $G$. Starting with triangles and wedges, and the discovery of triadic closure in social graphs [19], graphlets have become a central subject of study in social network analysis [9, 39], clustering [28, 37], and bioinformatics [3, 16, 34]; and they have found application in the development of graph kernels [36], graph embeddings [38], and graph neural networks [33]. The underlying idea is that, in many cases, the distribution of $k$-graphlets (the relative number of cliques, stars, paths, and so on) holds fundamental information about the nature of a complex network [31]. Understandably, these findings have sparked research on several basic graphlet mining problems such as finding, counting, listing, and sampling graphlets.

A preliminary version of these results appeared in the Proceedings of the ACM Symposium on Theory of Computing (STOC'21) [10].
Part of this work was done while the author was at the Sapienza University of Rome. The author was partially supported by Google under the Focused Award "Algorithms and Learning for AI" (ALL4AI), by the Bertinoro International Center for Informatics (BICI), by the European Research Council under the Starting Grant DMAP 680153, and by the Department of Computer Science of the Sapienza University of Rome under the grant "Dipartimenti di Eccellenza 2018-2022".
Author's address: M. Bressan, Dipartimento di Informatica, Università degli Studi di Milano, via Celoria 18, 20133, Milan, Italy; email: marco.bressan@unimi.it.

1549-6325/2023/06-ART26 $15.00
https://doi.org/10.1145/3596495

In this work we consider the two following problems. The *uniform graphlet sampling problem* asks, given $G$ and $k$, to return a $k$-graphlet uniformly at random from the set $\mathcal{V}_k$ of all $k$-graphlets of $G$. The *$\varepsilon$-uniform graphlet sampling problem* asks, given $G$, $k$, and $\varepsilon > 0$, to return a $k$-graphlet from any distribution whose total variation distance from the uniform distribution over $\mathcal{V}_k$ is at most $\varepsilon$. Clearly, an efficient algorithm for one of these problems yields an efficient algorithm for estimating the $k$-graphlet distribution. For this reason, uniform and $\varepsilon$-uniform graphlet sampling have been investigated for almost a decade, both in theory and in practice [1, 7, 12–15, 17, 24, 29, 32, 35, 41].

Unfortunately, although sampling a random $k$-vertex subgraph of $G = (V, E)$ uniformly at random is trivial, sampling a graphlet is considerably more challenging, due to the fact that a graphlet is *connected*. Let $n = |V|$ and $m = |E|$. For uniform graphlet sampling, to date, no algorithm is known that runs in less than $\Theta(n + m)$ time per sample. The only somewhat efficient algorithms known are for $\varepsilon$-uniform graphlet sampling, and they can be divided into direct sampling algorithms (that do not have a preprocessing phase) and two-phase sampling algorithms (which have a preprocessing phase and a sampling phase). We now discuss those algorithms briefly. Here and in what follows, we assume that $n + m \gg k$, so a running time of $2^{O(k)}(n + m)$ or $k^{O(k)}(n + m)$ is better than a running time of, say, $O((n+m)^2)$. This reflects the fact that, today, real-world graphs can easily have billions of edges, but $k$ rarely exceeds 5 or 10.

For direct sampling algorithms, the state-of-the-art is the so-called $k$-graphlet walk. To begin, consider the graph $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{E}_k)$ whose vertices are the $k$-graphlets of $G$ and where there is an edge between two graphlets if their intersection is a $(k - 1)$-graphlet. The *$k$-graphlet walk* is the lazy random walk over $\mathcal{G}_k$. It is not hard to show that, if $G$ is connected, then this walk is ergodic and so converges to a stationary distribution. Thus, to obtain $\varepsilon$-uniform graphlets, one can run the walk until it comes $\varepsilon$-close to its stationary distribution and then use rejection sampling. This technique is extensively used, thanks to its simplicity and elegance [1, 7, 17, 24, 29, 35, 41]; the drawback is that its running time depends on $t_\varepsilon(\mathcal{G}_k)$, the $\varepsilon$-mixing time of the walk, which can range anywhere from $\Theta(1)$ to $\Theta(n^{k-1})$ [12, 13]. Indeed, the analysis of $t_\varepsilon(\mathcal{G}_k)$ is nontrivial, and between the best lower and upper bounds there is still a multiplicative gap of $\Delta^{k-1}$ [1], where $\Delta$ is the maximum outdegree of $G$.

For two-phase algorithms, the state-of-the-art is an extension of the color coding technique of [4], proposed in [13]. The preprocessing phase assigns each vertex of $G$ with a uniform random color in $\{1, \ldots, k\}$ and then computes via dynamic programming the number of colorful trees rooted at every vertex of the graph (a subgraph is colorful if its vertices have distinct colors). The sampling phase uses the table counts to sample colorful graphlets uniformly at random. The resulting algorithm uses preprocessing time $2^{O(k)}(n + m)$ and space $2^{O(k)}n$, and expected sampling time $k^{O(k)}\Delta$; by increasing the space to $2^{O(k)}(n + m)$, one can reduce the expected sampling time to $k^{O(k)}$. It is not hard to show that, by increasing the preprocessing time and space to $2^{O(k)}(n+m)\log\frac{1}{\varepsilon}$, one can take $\varepsilon$-uniform graphlet samples in expected time $k^{O(k)}(\log\frac{1}{\varepsilon})^2$ per sample. Unfortunately, all these algorithms do not yield uniform graphlets, unless one runs them from scratch for every sample. Moreover, the dynamic programming part looks like an overkill even for $\varepsilon$-approximate sampling.[1] This leaves open the search for simpler or faster graphlet sampling algorithms.

In conclusion, (i) we do not have tight bounds for the $k$-graphlet walk, (ii) we do not have an efficient algorithm for uniform graphlet sampling, and (iii) we do not know if the existing algorithms for $\varepsilon$-uniform graphlet sampling are optimal. The goal of our work is to fill these gaps.

[1]The dynamic program actually computes, for every $v \in V(G)$, every $h = 1, \ldots, k$, every rooted tree $T$ on $h$ vertices, and every subset $C \subseteq \{1, \ldots, k\}$ of $h$ colors, the number of occurrences of $T$ in $G$ rooted at $v$ whose vertices span precisely the set of colors $C$.

## 2 RESULTS

We give three contributions. First, we settle the mixing time of the $k$-graphlet walk up to multiplicative $k^{O(k)} \log^2 n$ factors. Second, we present the first efficient algorithm for uniform graphlet sampling, with a preprocessing linear in $n + m$ and an expected sampling time $k^{O(k)} \log \Delta$. Third, we give the first $\varepsilon$-uniform graphlet sampling algorithm with sampling time independent of $G$, and preprocessing time $O(n \log n)$, which is sublinear in $m$ as long as $m = \omega(n \log n)$. The rest of this section overviews these results; later sections give the proofs. We adopt two standard graph access models. The first one is the adjacency-list model, where in time $O(1)$ one can learn the degree or the $i$th neighbor of a vertex. The second one is the model of [20, 26], where one can also check for the existence of an edge in time $O(1)$. When we say that our algorithms return a $k$-graphlet of $G$, we mean they return the actual connected subgraph of $G$ and not just its vertex set.

### 2.1 Near-optimal Mixing Time Bounds for the k-graphlet Walk

For a generic graph $\mathcal{G}$ let $t_\varepsilon(\mathcal{G})$ denote the $\varepsilon$-mixing time of the random walk over $\mathcal{G}$ (see Section 4 for a formal definition). Moreover, let $t(\mathcal{G}) = t_{\frac{1}{4}}(\mathcal{G})$; it is well known that $t_\varepsilon(\mathcal{G}) = O(t(\mathcal{G}) \cdot \log \frac{1}{\varepsilon})$, hence bounds on $t(\mathcal{G})$ yield bounds on $t_\varepsilon(\mathcal{G})$ for all $0 < \varepsilon \leq \frac{1}{4}$. Now, let $\rho(G) = \frac{\Delta}{\delta}$ be the ratio between the largest and the smallest degree of $G$, and recall from above the graph $\mathcal{G}_k$ that represents the adjacencies between the $k$-graphlets of $G$. We prove:

Theorem 1. *For all graphs $G$ and all $k \geq 2$,*

$$t(\mathcal{G}_k) \leq t(G) \cdot k^{O(k)} \rho(G)^{k-1} \log n. \tag{1}$$

*Moreover, for any function $\rho(n) \in \Omega(1) \cap O(n)$ there exists a family of arbitrarily large graphs $G$ on $n$ vertices that satisfy $\rho(G) = \Theta(\rho(n))$ and*

$$t(\mathcal{G}_k) \geq t(G) \cdot k^{-O(k)} \rho(G)^{k-1} / \log n. \tag{2}$$

Essentially, Theorem 1 says that the lazy walk on $\mathcal{G}_k$ behaves like the lazy walk on $G$ slowed down by a factor $\rho(G)^{k-1}$. This should be compared with the upper and lower bound of [1], which are, respectively, $t(G)\, \tilde{O}(\rho(G)^{2(k-1)})$ and $t(G)\, \Omega(\rho(G)^{k-1}\delta^{-1})$. Ignoring $k^{O(k)}$ poly $\log n$ factors, we improve those bounds by $\rho(G)^{k-1}$ and $\delta$, respectively.

From Theorem 1, we obtain the best bounds known for $\varepsilon$-uniform graphlet sampling based on random walks:

Theorem 2. *There exists a random-walk based algorithm that in the graph access model of [20, 26] yields the following guarantees: for all $G$, all $k \geq 2$, and all $\varepsilon > 0$, it returns an $\varepsilon$-uniform $k$-graphlet from $G$ in expected time $k^{O(k)} t(G)\, \rho(G)^{k-2} \log \frac{n}{\varepsilon}$. In the adjacency-list graph access model, the algorithm runs in time $k^{O(k)} t(G)\, \rho(G)^{k-2} \Delta \log \frac{n}{\varepsilon}$ or $O(n + m) + k^{O(k)} t(G)\, \rho(G)^{k-2} \log \frac{n}{\varepsilon} \log \Delta$.*

Note that, although $t(\mathcal{G}_k)$ grows with $\rho(G)^{k-1}$, the bound above grows with $\rho(G)^{k-2}$. The reason is that, as noted in [29, 41], sampling $k$-graphlets is equivalent to sampling the edges of $\mathcal{G}_{k-1}$. So, we can run the walk over $\mathcal{G}_{k-1}$ rather than over $\mathcal{G}_k$, which yields a mixing time proportional to $\rho(G)^{k-2}$ rather than $\rho(G)^{k-1}$. As a sanity check, when $k = 2$ our algorithm matches the natural bound $O(t(G))$ achieved by the simple random walk over $G$.

Regarding the techniques, our proofs are very different from those of [1]. There, the authors showed a mapping between the cuts of $\mathcal{G}_k$ and those of $G$; this allowed them to bound the conductance of $\mathcal{G}_k$ by a function of the conductance of $G$ and then bound $t_\varepsilon(\mathcal{G}_k)$ via Cheeger's inequality. However, since Cheeger's inequality can be loose by a quadratic factor, their upper bound on $t_\varepsilon(\mathcal{G}_k)$ grows with $\rho(G)^{2(k-1)}$ instead of $\rho(G)^{k-1}$ (see above). To avoid this blow-up, we establish a connection between the *relaxation times* of $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$, for all $i = 1, \ldots, k-1$, and thus between

$\mathcal{G}_1 = G$ and $\mathcal{G}_k$. To this end, we prove a technical result on the relaxation time of the lazy walk on the *line graph* of $G$ (the graph encoding the adjacencies between the edges of $G$). We state this result here, as it may be of independent interest:

Lemma 3. *Any graph $G$ satisfies $\tau(L(G)) \leq 20\,\rho(G)\,\tau(G)$, where $L(G)$ is the line graph of $G$ and $\tau(\cdot)$ denotes the relaxation time of the lazy random walk.*

### 2.2 Uniform Graphlet Sampling

We describe the first efficient algorithm for *uniform graphlet sampling*:

Theorem 4. *There exists a two-phase graphlet sampling algorithm, Ugs (uniform graphlet sampler), that in the adjacency-list graph access model yields the following guarantees:*

*(1) the preprocessing phase runs in time $O(n\,k^2 \log k + m)$ and space $O(n + m)$*
*(2) the sampling phase returns $k$-graphlets independently and uniformly at random in $k^{O(k)} \log \Delta$ expected time per sample.*

The technique behind Ugs is radically different from random walks and color coding. The key idea is to "regularize" $G$, that is, to sort $G$ so each vertex $v$ has maximum degree in the subgraph $G(v)$ induced by $v$ and all vertices after it (this can be done by just repeatedly removing the maximum-degree vertex from $G$). As we show, this makes each $G(v)$ behave like a regular graph, which makes it efficient to perform rejection sampling of randomly grown spanning trees. It is worth noting that several attempts have been made to sample graphlets uniformly by growing random subsets and applying rejection sampling (see, for instance, [25, 32]). All those algorithms, however, have one crucial limitation: In the worst case, the rejection probability approaches $1 - \Delta^{-k+1}$, in which case roughly $\Delta^{k-1}$ rejection trials are needed to draw a single graphlet. It is somewhat surprising that the fact that just sorting $G$ solves the problem has gone unnoticed until now.

Ugs can also be used as a graphlet *counting* algorithm:

Theorem 5. *Choose any $\varepsilon_0, \varepsilon_1, \delta \in (0, 1)$. There exists an algorithm that in the adjacency-list graph access model runs in time*

$$O(m) + k^{O(k)} \left( \frac{n}{\varepsilon_0^2} \log \frac{n}{\delta} + \frac{1}{\varepsilon_1^2} \log \frac{1}{\delta} \right) \log \Delta, \tag{3}$$

*and, with probability $1 - \delta$, returns for every distinct (up to isomorphism) connected $k$-vertex graph $H$ an additive $(\varepsilon_0 N_H + \varepsilon_1 N_k)$-approximation of $N_H$, where $N_H$ is the number of graphlets of $G$ isomorphic to $H$, and $N_k = |\mathcal{V}_k|$ is the total number of $k$-graphlets in $G$.*

### 2.3 Epsilon-uniform Graphlet Sampling

We present:

Theorem 6. *There exists a two-phase graphlet sampling algorithm, Apx-Ugs, that in the graph access model of [20, 26] has the following guarantees for all $\varepsilon > 0$:*

*(1) the preprocessing phase takes time $O((\frac{1}{\varepsilon})^{\frac{2}{(k-1)}} k^6\, n \log n)$ and space $O(n)$*
*(2) with high probability over the preprocessing phase, the sampling phase returns $k$-graphlets independently and $\varepsilon$-uniformly at random in $k^{O(k)} (\frac{1}{\varepsilon})^{8+\frac{4}{(k-1)}} \log \frac{1}{\varepsilon}$ expected time per sample.*

The remarkable fact about Apx-Ugs is that its preprocessing time grows as $n \log n$, and is therefore independent of the edge set of $G$. This should be contrasted with the color-coding algorithm, whose preprocessing time grows as $n + m$. Moreover, our preprocessing time is polynomial in both $\frac{1}{\varepsilon}$ and $k$, while that of color coding is exponential in $k$. For what concerns the expected sampling time,

Table 1. Our Upper Bounds (Shaded) Compared to Existing Work

| | preprocessing time | preprocessing space | time per sample | output |
|---|---|---|---|---|
| [12] | – | – | $2^{O(k)}(n+m)+k^{O(k)}$ | uniform |
| Ugs | $O(nk^2\log k+m)$ | $O(n+m)$ | $k^{O(k)}\log\Delta$ | uniform |
| [29] | $O(n)$ | $O(n)$ | $k^{O(k)}\left(\Delta\log\frac{n}{\varepsilon}\right)^{k-3}$ | $\varepsilon$-uniform |
| [12] | $2^{O(k)}(n+m)\log\frac{1}{\varepsilon}+k^{O(k)}\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon}$ | $2^{O(k)}n\log\frac{1}{\varepsilon}$ | $k^{O(k)}\Delta\left(\log\frac{1}{\varepsilon}\right)^2$ | $\varepsilon$-uniform |
| [12] | $2^{O(k)}(n+m)\log\frac{1}{\varepsilon}+k^{O(k)}\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon}$ | $2^{O(k)}(n+m)\log\frac{1}{\varepsilon}$ | $k^{O(k)}\left(\log\frac{1}{\varepsilon}\right)^2$ | $\varepsilon$-uniform |
| Apx-Ugs | $O\left(\left(\frac{1}{\varepsilon}\right)^{\frac{2}{(k-1)}}k^6n\log n\right)$ | $O(n)$ | $k^{O(k)}\left(\frac{1}{\varepsilon}\right)^{8+\frac{4}{(k-1)}}\log\frac{1}{\varepsilon}$ | $\varepsilon$-uniform |
| Rwgs | – | – | $k^{O(k)}t(G)\left(\frac{\Delta}{\delta}\right)^{k-2}\log\frac{n}{\varepsilon}$ | $\varepsilon$-uniform |

like the one of color coding, ours is independent of $G$, but it pays an extra poly $\frac{1}{\varepsilon}$ factor. However, we did not make hard attempts to optimize those factors, and they might be improved.

While Ugs is rather simple, Apx-Ugs is considerably more involved. The high-level idea is, unsurprisingly, to "approximate" Ugs in both phases. However, this turns out to be a delicate issue, which requires a careful combination of graph sketching, cut size estimation, and coupling arguments. The reason is that Ugs relies crucially on a particular topological order of $G$, whose exact computation takes time $\Omega(m)$, and which is not clear how to approximate in time $o(m)$. In fact, it is not even clear what definition of "approximate order" is the right one for our purposes; in the end, the definition we use turns out to be nontrivial.

To conclude, we observe that Apx-Ugs is nearly optimal in the graph access model of [26] in the sense that any algorithm must pay a running time close to that of the preprocessing phase of Apx-Ugs even just to return a single $\varepsilon$-uniform graphlet:

Theorem 7. *For any $k\geq 2$ and any $\varepsilon\in[0,1]$, any $\varepsilon$-uniform $k$-graphlet sampling algorithm has worst-case expected running time $\Omega(n/k)$ in the graph access model of [26].*

Proof. Let $G$ contain a $k$-path plus $n-k$ isolated vertices. In the worst case any algorithm must examine $\Omega(n/k)$ vertices in expectation before finding the only $k$-graphlet of $G$. □

Table 1 summarizes our upper bounds and the state-of-the-art.

## 3 RELATED WORK

The $k$-graphlet walk algorithm was introduced by [7] without formal running time bounds. The first bounds on $t_\varepsilon(\mathcal{G}_k)$ were given by [12], while the first bounds tying $t_\varepsilon(\mathcal{G}_k)$ to $t_\varepsilon(G)$ were given by [1]. Recently, [29] developed a graphlet sampling random walk with running time $k^{O(k)}(\Delta\log\frac{n}{\varepsilon})^{k-3}$. Their approach is similar to ours, as they build the $k$-graphlet walk recursively from the $(k-1)$-graphlet walk. However, they assume one can sample edges uniformly at random from $G$ in time $O(1)$, which requires a $O(n)$-time preprocessing, or an additional factor of $t_\varepsilon(G)$ to sample edges via random walks. Moreover, their running bound grows like $\Delta^k$, while ours grows as $(\frac{\Delta}{\delta})^k$.

The color coding extension for estimating graphlet counts was introduced in [12, 13]. This extension does not allow to $\varepsilon$-uniform graphlet sampling directly; however, it can be obtained by making several independent runs, for a total preprocessing time of $2^{O(k)}(n+m)\log\frac{1}{\varepsilon}+k^{O(k)}\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon}$, a preprocessing space of $2^{O(k)}m\log\frac{1}{\varepsilon}$, and an expected sampling time of $k^{O(k)}(\log\frac{1}{\varepsilon})^2$. See Appendix C for a complete proof. As said, one can also obtain uniform samples by running the entire algorithm of [12] from scratch, in $2^{O(k)}O(n+m)$ time per sample.

Rejection sampling is at the heart of several graphlet sampling algorithms, such as path sampling [25] and lifting [32]. These algorithms start by drawing a random vertex from $G$ and then repeatedly selecting random edges in the cut. This technique alone seems destined to fail: In the worst case, the rejection probability must be as large as $\simeq 1-\Delta^{-k+1}$, resulting in a vacuous $O(\Delta^{k-1})$ running time bound. The main idea behind our algorithms is to make such a rejection sampling efficient by sorting $G$ so to virtually "bucket" the graphlets, so within every single bucket the sampling probabilities are roughly balanced.

There is also intense work on sampling and counting copies of *a specific pattern* $H$ in sublinear time, including edges, triangles, cliques, and other patterns [5, 8, 20–23]. However, "sublinear" there is meant in the maximum possible number of copies of $H$, which can be as large as $\Theta(m^{\frac{k}{2}})$. It is also unclear how those techniques can be applied to uniform graphlet sampling.

Several algorithms for counting subgraphs efficiently rely on ordering the input graph, in particular according to the degeneracy (or "smallest-last") ordering [6, 11, 18, 30]. The ordering exploited by our uniform sampling algorithms, which is obtained by a "largest-last" rule, may appear strictly related. However, it is not hard to see that the two orderings are *not* the reverse of each other. Moreover, while the goal of smallest-last orderings is to make the out-degree of every individual vertex as small as possible, the goal of our ordering is to make the degree of each vertex dominate that of those vertices that come afterwards in the ordering.

## 4 PRELIMINARIES AND NOTATION

We assume the word RAM model of computation. For any graph $G = (V, E)$, we assume $V = \{1, \ldots, n\}$. We denote the degree of $v \in V$ by $d_v$. We consider two graph access models. The adjacency-list graph access model supports the following $O(1)$-time queries:

- degree query: given $v \in V$, return $d_v$
- neighbor query: given $v \in V$ and $i \in \mathbb{N}$, return the $i$th neighbor of $v$ in $G$, or $-1$ if $d_v < i$.

The graph access model of [20, 26] adds the following $O(1)$-time query:

- pair query (or edge testing query): given $u, v \in V$, tell if $\{u, v\} \in E$.

For any $U \subseteq V$ and $U' \subseteq V \setminus U$, the cut between $U$ and $U'$ is $\mathrm{Cut}(U, U') = E \cap \{\{u, v\} : u \in U, v \in U'\}$. The line graph $L(G) = (V', E')$ of a graph $G = (V, E)$ is defined by $V' = \{v_e : e \in E\}$, and $\{v_e, v_{e'}\} \in E'$ if and only if $|e \cap e'| = 1$. For $u, v \in V(G)$, we write $u \sim v$ for $\{u, v\} \in E(G)$.

A $k$-graphlet $g = (V(g), E(g))$ is a $k$-vertex subgraph of $G$ that is connected and induced. With a slight abuse of notation, we may use $g$ in place of $V(g)$, and $g \cap g'$ in place of $G[V(g) \cap V(g')]$. We denote by $\mathcal{V}_k$ the set of all $k$-graphlets of $G$. The $k$-graphlet graph of $G$ is $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{E}_k)$, where $\{g, g'\} \in \mathcal{E}_k$ if and only if $g \cap g' \in \mathcal{V}_{k-1}$. We note that some works define $g$ and $g'$ to be adjacent if $|V(g) \cap V(g')| = k - 1$, but our proofs do not work in that case (and so the mixing time of those walks may not respect our bounds).

In this article, "$X$ holds with high probability for $Y = \Theta(Z)$" means that for any fixed $a > 0$, we can make $\mathbb{P}(X) > 1 - n^{-a}$ by choosing $Y \in \Theta(Z)$ sufficiently large. Similarly, "X has probability $\mathrm{poly}(x)$" means that for any fixed $a > 0$, we can make $\mathbb{P}(X) < x^a$ by adjusting the constants in our algorithms.

## 5 NEAR-OPTIMAL MIXING TIME BOUNDS FOR THE K-GRAPHLET WALK

In this section, we prove the results of Section 2.1. Towards this end, we need to recall some additional preliminary results on Markov Chains, random walks, and mixing.

### 5.1 Preliminaries

We denote by $X = \{X_t\}_{t\geq 0}$ a generic Markov chain over a finite state space $\mathcal{V}$. We denote by $P$ the transition matrix of the chain and $\pi_t$ the distribution of $X_t$. We always assume that the chain is ergodic and denote by $\pi = \lim_{t\to\infty}\pi_t$ its unique limit distribution. We also let $\pi_{\min} = \min_{x\in\mathcal{V}}\pi(x)$ be the smallest stationary probability of any state. The $\varepsilon$-mixing time of $X$ is $t_\varepsilon(X) = \min\{t_0 : \forall X_0 \in \mathcal{V} : \forall t \geq t_0 : \mathrm{tvd}(\pi_t, \pi) \leq \varepsilon\}$. When we write $t(X)$, we mean $t_{\frac{1}{4}}(X)$. Here, $\mathrm{tvd}(\sigma,\pi) = \max_{A\subseteq\mathcal{V}}\{\sigma(A) - \pi(A)\}$ is the variation distance between the distributions $\sigma$ and $\pi$; if $\mathrm{tvd}(\sigma,\pi) \leq \varepsilon$ and $\pi$ is uniform, then we say $\sigma$ is $\varepsilon$-uniform.

A graph with non-negative edge weights is denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ where $w : \mathcal{E} \to \mathbb{R}_0^+$. For every $u \in \mathcal{V}$, we let $w(u) = \sum_{e\in\mathcal{E}:u\in e} w(e)$. Any such $\mathcal{G}$ induces a *lazy random walk* as follows: Let $P_0$ be the matrix given by $P_0(u,v) = \frac{w(u,v)}{w(u)}$. Now, let $P = \frac{1}{2}(P_0 + I)$ where $I$ is the identity matrix. This can be seen as adding a loop of weight $w(u)$ at each vertex of the graph. Note that $P_0$ and $P$ are both stochastic. The lazy random walk over $\mathcal{G}$ is Markov chain with state space $\mathcal{V}$ and transition matrix $P$. By standard Markov chain theory, if $\mathcal{G}$ is connected, then the lazy random walk is ergodic and converges to the limit distribution $\pi$ given by $\pi(u) = \frac{w(u)}{\sum_{v\in V} w(v)}$. It is well-known that the chain is time-reversible with respect to $\pi$, that is, $\pi(x)P(x,y) = \pi(y)P(y,x)$ for all $x, y \in \mathcal{V}$; and that every time-reversible chain on a finite state space $\mathcal{V}$ can be seen as a random walk over a graph $G = (\mathcal{V}, \mathcal{E}, w)$ where $w(x,y) = \pi(x)P(x,y)$. Thus, we will often write $\mathcal{G}$ in place of $X$, in which case $X$ is understood to be the lazy chain over $\mathcal{G}$. The quantity $Q(x,y) = \pi(x)P(x,y)$ is called transition rate between $x$ and $y$.

The volume of $U \subseteq \mathcal{V}$ is $\mathrm{vol}(U) = \sum_{u\in U} w(u)$. The cut of $U \subseteq \mathcal{V}$ is $\mathrm{Cut}(U) = \{e = \{u,u'\} \in \mathcal{E} : u \in U, u' \in \mathcal{V}\setminus U\}$, and its weight is $c(U) = \sum_{e\in\mathrm{Cut}(U)} w(e)$. The conductance of $U \subseteq \mathcal{V}$ is $\Phi(U) = c(U)/\mathrm{vol}(U)$. The conductance of $\mathcal{G}$ is $\Phi(\mathcal{G}) = \min\{\Phi(U) : U \subset \mathcal{V}, \mathrm{vol}(U) \leq \frac{1}{2}\mathrm{vol}(\mathcal{V})\}$.

#### 5.1.1 *Spectral Gaps and Relaxation Times.*

*Definition 8.* Let $P$ be the transition matrix of $X$, and let $\lambda_* = \max\{|\lambda| : \lambda \text{ is an eigenvalue of } P, \lambda \neq 1\}$. The spectral gap of $X$ is $\gamma = 1 - \lambda_*$. The relaxation time of $X$ is $\tau(X) = \frac{1}{\gamma}$.

Classic mixing time theory (see, e.g., [27]) gives the following relationships:

$$\frac{1}{4\Phi} \leq \tau(X), t_\varepsilon(X) \leq \frac{2}{\Phi^2}\log\frac{1}{\varepsilon\pi_{\min}}, \tag{4}$$

$$(\tau(X)-1)\log\frac{1}{2\varepsilon} \leq t_\varepsilon(X) \leq \tau(X)\log\frac{1}{\varepsilon\pi_{\min}}. \tag{5}$$

One can show that the last inequality implies $\tau(X) \leq c\, t(X)$ for some (small) constant $c \geq 1$.

#### 5.1.2 *Dirichlet Forms.*

For any function $f : \mathcal{V} \to \mathbb{R}$ let $\mathrm{Var}_\pi f = \mathbb{E}_\pi(f - \mathbb{E}_\pi f)^2$.

*Definition 9 (Dirichlet form; see [27], Section 13.2.1).* Let $f : \mathcal{V} \to \mathbb{R}$ be any function. Then the Dirichlet form associated to $P, \pi, f$ is:

$$\mathcal{E}_{P,\pi}(f) = \frac{1}{2}\sum_{x,y\in\mathcal{V}}\big(f(x) - f(y)\big)^2 Q(x,y). \tag{6}$$

The Dirichlet form characterizes the spectral gap as follows:

Lemma 10 (see [27], Lemma 13.12). *The spectral gap satisfies:*

$$\gamma = \min_{\substack{f\in\mathbb{R}^V \\ \mathrm{Var}_\pi(f)\neq 0}} \frac{\mathcal{E}_{P,\pi}(f)}{\mathrm{Var}_\pi(f)}. \tag{7}$$

Next, we recall some results relating the spectral gaps of different chains.

#### 5.1.3 Direct Comparison.

Lemma 11 ([27], Lemma 13.18). *Let $P$ and $\tilde{P}$ be reversible transition matrices with stationary distributions $\pi$ and $\tilde{\pi}$, respectively. If $\mathcal{E}_{\tilde{P},\tilde{\pi}}(f) \leq \alpha\, \mathcal{E}_{P,\pi}(f)$ for all functions $f$, then*

$$\tilde{\gamma} \leq \left(\max_{x \in \mathcal{V}} \frac{\pi(x)}{\tilde{\pi}(x)}\right) \alpha\gamma. \tag{8}$$

Lemma 12 ([2], Lemma 3.29). *Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ possibly with loops. Let $w$ and $w'$ be two weightings of $\mathcal{E}$ and let $\gamma$ and $\gamma'$ be the spectral gaps of the corresponding random walks. Then:*

$$\gamma' \geq \gamma \cdot \frac{\min_{e \in \mathcal{E}}(w(e)/w'(e))}{\max_{v \in \mathcal{V}}(w(v)/w'(v))}. \tag{9}$$

#### 5.1.4 Collapsed Chains. (See [2], Section 2.7.3).

*Definition 13.* Let $A \subset \mathcal{V}$ and let $A^C = \mathcal{V} \setminus A$ (note that $A^C \neq \emptyset$). The collapsed chain $X^*$ has state space $A \cup \{a\}$ where $a$ is a new state representing $A^C$, and transition matrix given by:

$$P^*(u,v) = P(u,v) \qquad u, v \in A \tag{10}$$

$$P^*(u,a) = \sum_{v \in A^C} P(u,v) \qquad u \in A \tag{11}$$

$$P^*(a,v) = \frac{1}{\pi(A^C)} \sum_{u \in A^C} \pi(u)P(u,v) \qquad v \in A \tag{12}$$

$$P^*(a,a) = \frac{1}{\pi(A^C)} \sum_{u \in A^C} \sum_{v \in A^C} \pi(u)P(u,v). \tag{13}$$

Lemma 14 ([2], Corollary 3.27). *The collapsed chain $X^*$ satisfies $\gamma(X^*) \geq \gamma(X)$.*

#### 5.1.5 Induced Chains.

*Definition 15 ([27], Section 13.4).* Let $\emptyset \neq A \subseteq \mathcal{V}$ and $\tau_A^+ = \min\{t \geq 1 : X_t \in A\}$. The induced chain on $A$ is the chain with state space $A$ and transition probabilities:

$$P_A(x,y) = P(X_{\tau_A^+} = y \mid X_0 = x) \qquad \forall x, y \in A. \tag{14}$$

Lemma 16 ([27], Theorem 13.20). *Let $\emptyset \neq A \subseteq \mathcal{V}$, and let $\gamma_A$ be the spectral gap for the chain induced on $A$. Then $\gamma_A \geq \gamma$.*

### 5.2 Proof of the Upper Bound of Theorem 1

This section proves the upper bound of Theorem 1. Consider the random walk over $\mathcal{G}_k$. It is easy to see that $\pi_{\min} \geq k^{-O(k)} n^{-k} = n^{-O(k)}$. Since by Equation (5) we have $t(\mathcal{G}_k) \leq \tau(\mathcal{G}_k) \ln \frac{4}{\pi_{\min}}$, this implies $t(\mathcal{G}_k) \leq O(\tau(\mathcal{G}_k) k \log n)$. Now consider the following inequality:

$$\tau(\mathcal{G}_k) \leq \text{poly}(k)\rho(G)\tau(\mathcal{G}_{k-1}). \tag{15}$$

Applying Equation (15) to $\tau(\mathcal{G}_k), \tau(\mathcal{G}_{k-1}), \ldots, \tau(\mathcal{G}_2)$, and, since $\mathcal{G}_1 = G$ and $\tau(G) = O(t(G))$, we obtain:

$$t(\mathcal{G}_k) \leq k^{O(k)} \rho(G)^{k-1} t(G) \log n, \tag{16}$$

which is precisely the upper bound of Theorem 1. Thus, we only need to prove Equation (15). The main obstacle in proving that inequality is in relating the spectral gaps of two very different walks—one over $G$ and one over $\mathcal{G}_k$. We overcome this obstacle by proving the following result:

Lemma 17. $\tau(\mathcal{G}_k) \leq \text{poly}(k)\tau(L(\mathcal{G}_{k-1}))$.

Together with the following restatement of Lemma 3 (with $\mathcal{G}$ in place of $G$ to avoid ambiguities), this result yields precisely Equation (15).

Lemma 3. *Any graph $\mathcal{G}$ satisfies $\tau(L(\mathcal{G})) \leq 20\,\rho(\mathcal{G})\,\tau(\mathcal{G})$, where $L(\mathcal{G})$ is the line graph of $\mathcal{G}$ and $\tau(\cdot)$ denotes the relaxation time of the lazy random walk.*

Thus, we shall prove Lemma 17 and 3, in this order.

*5.2.1 Proof of Lemma 17.* From $L(\mathcal{G}_{k-1})$, we construct a weighted graph $L_N$ such that $\tau(L_N) \leq \tau(L(\mathcal{G}_{k-1}))$, and then we prove that $\tau(\mathcal{G}_k) \leq \text{poly}(k)\tau(L_N)$. Combining these inequalities gives the claim.

For any $g \in V(\mathcal{G}_k)$ let $H(g) = \{x_{uv} \in V(L(\mathcal{G}_{k-1})) : g = u \cup v\}$. Note that $\{H(g)\}_{g \in V(\mathcal{G}_k)}$ is a partition of $V(L(\mathcal{G}_{k-1}))$ into equivalence classes; each class $H(g)$ contains every vertex of $L(\mathcal{G}_{k-1})$ that represents a pair of $(k-1)$-graphlets whose union yields $g$. Now, let $V(\mathcal{G}_k) = \{g_1, \ldots, g_N\}$, and let $L_0 = L(\mathcal{G}_{k-1})$. For each $i = 1, \ldots, N$, we define $L_i$ by taking $L_{i-1}$ and identifying $H(g_i)$. Formally, we let $L_i = (V(L_i), E(L_i), w_i)$, where $V(L_i) = V(L_{i-1}) \setminus H(g_i) \cup \{a_i\}$ with $a_i$ being a new state representing $H(g_i)$, and:

$$w_i(x, x') = w_{i-1}(x, y) \qquad x \neq a_i, x' \neq a_i \tag{17}$$

$$w_i(x, a_i) = \sum_{x' \in a_i} w_{i-1}(x, x') \qquad x \neq a_i \tag{18}$$

$$w_i(a_i, a_i) = \sum_{x \in a_i} \sum_{x' \in a_i} w_{i-1}(x, x'). \tag{19}$$

Now, we prove two claims from which the thesis immediately follows.

Claim 1. $\tau(\mathcal{G}_k) \leq \text{poly}(k)\tau(L_N)$.

Proof. We show that the walk on $L_N$ is the lazy walk on $\mathcal{G}_k$ up to a reweighting of the edges by multiplicative factors in $[1, \text{poly}(k)]$. By Lemma 12, this implies the thesis. In particular, we show that, if $\mathcal{G}_k$ is taken in its lazy version (with loops accounting for half of the vertex weight), then (1) $V(L_N) = V(\mathcal{G}_k)$, (2) $E(L_N) = E(\mathcal{G}_k)$, (3) $1 \leq \frac{w_N}{w_{\mathcal{G}_k}} \leq \text{poly}(k)$. We denote the generic state $a_i \in L_N$ simply as $g$, meaning that $a_i$ represents $H(g)$.

*(1)* $V(L_N) = V(\mathcal{G}_k)$. Let $g \in V(L_N)$. By construction, $g = u \cup v$ for some $\{u, v\} \in E(\mathcal{G}_{k-1})$. Hence, $g$ has $k$ vertices and is connected, so it is a $k$-graphlet, and $g \in V(\mathcal{G}_k)$. Conversely, let $g \in V(\mathcal{G}_k)$ and let $T$ be a spanning tree of $g$ (which must exist, since $g$ is connected by definition). Let $a, b$ be two distinct leaves of $T$ and let $g' = g \setminus \{a\}$ and $g'' = g \setminus \{b\}$. Then $g', g''$ are connected and have $k-1$ vertices, so they are in $V(\mathcal{G}_{k-1})$. Moreover $|g' \cap g''| = k-2$, so $\{g', g''\} \in E(\mathcal{G}_{k-1})$. Thus, $\{g', g''\} \in L(\mathcal{G}_{k-1})$ and consequently $g \in V(L_N)$. Therefore, $V(L_N) = V(\mathcal{G}_{k-1})$.

*(2)* $E(L_N) = E(\mathcal{G}_k)$. First, both $L_N$ and the lazy version of $\mathcal{G}_k$ have a loop at each vertex ($L_N$ inherits from $L_0$ a positive self-transition probability at each vertex). Now, let $\{g', g''\} \in E(L_N)$ be a non-loop edge. By construction of $L_N$, we have $g' = u \cup v$ and $g'' = u \cup z$, with $\{u, v\}, \{u, z\} \in E(\mathcal{G}_{k-1})$ and $u, v, z \in V(\mathcal{G}_{k-1})$ distinct. This implies $g' \cap g'' = u$ and so $\{g', g''\} \in E(\mathcal{G}_k)$. It follows that $E(L_N) \subseteq E(\mathcal{G}_k)$. Now, let $\{g', g''\} \in E(\mathcal{G}_k)$ be a non-loop edge. Let $u = g' \cap g''$; note that by hypothesis $u$ is connected and $|u| = k-1$, so $u \in V(\mathcal{G}_{k-1})$. Now, let $\{a'\} = g' \setminus g''$ and let $b'$ be any neighbor of $a$ in $u$. Choose any spanning tree $T'$ of $u$ rooted at $b'$, and let $c' \neq b'$ be any leaf of $T'$ (such a leaf exists, since $|g| \geq 3$ and thus $|u| \geq 2$). We define $v = g' \setminus \{c'\}$. Note that by construction (1) $v$ is connected and has size $k-1$, (2) $u \cap v$ is connected and has size $k-2$, and

(3) $u \cup v = g'$. Therefore, $v \in V(\mathcal{G}_{k-1})$ and $\{u, v\} \in E(\mathcal{G}_{k-1})$. A symmetric construction using $g''$ and $u$ yields $z$ such that $z \in V(\mathcal{G}_{k-1})$ and $\{u, z\} \in E(\mathcal{G}_{k-1})$ and $u \cup z = g''$. Now, by construction, $\{u, z\}$ and $\{u, v\}$ give two adjacent states $x_{uv}, x_{uz} \in V(L_0)$. But $u \cup v = g'$ and $u \cup z = g''$, so $x_{uv} \in H(g)$ and $x_{uz} \in H(g'')$. This implies that $\{g', g''\} \in E(L_N)$. So, $E(\mathcal{G}_k) \subseteq E(L_N)$ and we conclude that $E(\mathcal{G}_k) = E(L_N)$.

*(3)* $1 \leq \frac{w_N}{w_{\mathcal{G}_k}} \leq \text{poly}(k)$. First, let us consider non-loop edges. Let $\{a, a'\} \in E(L_N)$ with $a \neq a'$, and let $g, g'$ be the corresponding elements of $\mathcal{G}_k$; note that $g \neq g'$. Observe that $w_N(a, a') = |\operatorname{Cut}(H(g), H(g'))|$, where the cut is taken in $L_0 = L(\mathcal{G}_{k-1})$. Clearly, $|\operatorname{Cut}(H(g), H(g'))| \geq 1$ and $w_{\mathcal{G}_k}(g, g') = 1$, therefore, $1 \leq \frac{w_N(a,a')}{w_{\mathcal{G}_k}(g,g')}$. For the other side, note that there are at most $\binom{k}{2}$ distinct pairs of $(k-1)$-graphlets $u, v \in \mathcal{G}_{k-1}$ such that $u \cup v = g$. Thus, $H(g) \leq \binom{k}{2}$. The same holds for $g'$. Therefore, $|\operatorname{Cut}(H(g), H(g'))| \leq \binom{k}{2}^2$. It follows that $\frac{w_N(a,a')}{w_{\mathcal{G}_k}(g,g')} \leq \binom{k}{2}^2$.

A similar argument holds for the loops. First, recall that $w_{\mathcal{G}_k}(g) = d_g$ by the lazy weighting. Consider then any non-loop edge $\{g, g'\} \in \mathcal{G}_k$. Note that $\{g, g'\}$ determines $u = g \cap g' \in V(\mathcal{G}_{k-1})$ univocally. Moreover, there exist some $v, z \in \mathcal{G}_{k-1}$ such that $u \cup v = g$ and $u \cup z = g'$ and that $\{x_{uv}, x_{uz}\}$ is an edge in $L_0$; and note that there are at most $k$ distinct $v$ and at most $k$ distinct $z$ satisfying these properties. Therefore, every $\{g, g'\}$ can be mapped to a set of between 1 to $k^2$ edges in $L_0$, such that every edge in the set is in the cut between $H(g)$ and $H(g')$. Furthermore, note that different $g'$ are mapped to disjoint sets, since any edge $\{x_{uv}, x_{uz}\}$ identifies univocally $g = u \cup v$ and $g' = u \cup z$. It follows that the cut of $H(g)$ is at least $d_g$ and at most $k^2 d_g$. Since the cut has at least one edge, and $H(g)$ has at most $\binom{k}{2}^2$ internal edges, then the total weight of $H(g)$ is between 1 and $\text{poly}(k)$ times the cut. This is also $w_N(a)$, the weight of the state $a$ representing $g$ in $L_N$. The claim follows by noting that by construction $w_N(a) \leq w_N(a, a) \leq 2w_N(a)$. □

Claim 2. $\tau(L_N) \leq \tau(L(\mathcal{G}_{k-1}))$.

Proof. The walk on $L_i$ is the walk $L_{i-1}$ collapsed respect to $A^C = H(g_i)$ (see Definition 13). Therefore, by Lemma 14 the spectral gaps of the two walks satisfy $\gamma(L_i) \geq \gamma(L_{i-1})$, and the relaxation times satisfy $\tau(L_i) \leq \tau(L_{i-1})$. Thus, $\tau(L_N) \leq \tau(L_0) = \tau(L(\mathcal{G}_{k-1}))$. □

By combining Claims 1 and 2, we obtain $\tau(\mathcal{G}_k) \leq \text{poly}(k)\tau(L_N) \leq \text{poly}(k)\tau(L(\mathcal{G}_{k-1}))$, proving Lemma 5.2.1.

*5.2.2 Proof of Lemma 3.* We build an auxiliary weighted graph $\mathcal{S}'$, as follows: Let $\mathcal{S}$ be the 1-subdivision of $\mathcal{G}$ (the graph obtained by replacing each $\{u, v\} \in E(\mathcal{G})$ with the path $\{u, x_{uv}\}, \{x_{uv}, v\}$ where $x_{uv}$ is a new vertex representing $\{u, v\}$). We make $\mathcal{S}$ lazy by adding loops and assigning the following weights:

$$w_{\mathcal{S}}(u, u) = d_u \qquad u \in V(\mathcal{G}) \tag{20}$$

$$w_{\mathcal{S}}(u, x_{uv}) = 1 \qquad \{u, v\} \in E(\mathcal{G}) \tag{21}$$

$$w_{\mathcal{S}}(x_{uv}, x_{uv}) = 2 \qquad \{u, v\} \in E(\mathcal{G}). \tag{22}$$

The graph $\mathcal{S}'$ is the same as $\mathcal{S}$ but with the following weights:

$$w_{\mathcal{S}'}(u, u) = d_u^2 \qquad u \in V(\mathcal{G}) \tag{23}$$

$$w_{\mathcal{S}'}(u, x_{uv}) = d_u \qquad \{u, v\} \in E(\mathcal{G}) \tag{24}$$

$$w_{\mathcal{S}'}(x_{uv}, x_{uv}) = d_u + d_v \qquad \{u, v\} \in E(\mathcal{G}). \tag{25}$$

The reader may refer to Figure 1 (below).

Now, we prove two claims that, combined, yield the thesis.

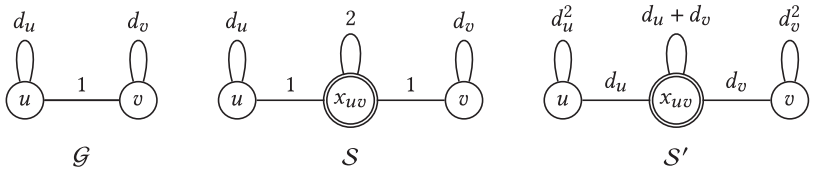


Fig. 1. Left: a pair of $(k-1)$-graphlets $u, v$ forming an edge in $\mathcal{G}_{k-1}$. Middle: how $\{u, v\}$ appears in $\mathcal{S}$, the 1-subdivision of $G$. Right: the reweighting given by $\mathcal{S}'$.

Claim 3. $\tau(\mathcal{S}') \leq 4\rho(\mathcal{G})\tau(\mathcal{G})$.

Proof. Let $\Delta, \delta$ be the maximum and minimum degrees of $\mathcal{G}$. First, note that

$$\min_{\{x,y\}\in E(\mathcal{S})} \frac{w_{\mathcal{S}}(x,y)}{w_{\mathcal{S}'}(x,y)} \geq \frac{1}{\Delta} \quad \text{and} \quad \max_{x\in V(\mathcal{S})} \frac{w_{\mathcal{S}}(x)}{w_{\mathcal{S}'}(x)} \leq \frac{1}{\delta}. \tag{26}$$

By Lemma 12, this implies that $\gamma(\mathcal{S}') \geq \rho(\mathcal{G})^{-1}\gamma(\mathcal{S})$, or equivalently $\tau(\mathcal{S}') \leq \rho(\mathcal{G})\,\tau(\mathcal{S})$. Thus, we need only to show that $\tau(\mathcal{S}) \leq 4\tau(\mathcal{G})$, or equivalently, $\gamma(\mathcal{G}) \leq 4\gamma(\mathcal{S})$. We do so by comparing the numerators and denominators of Equation (7) in Lemma 11 for $\mathcal{S}$ and $\mathcal{G}$.

Consider the walk on $\mathcal{S}$ and let $\pi_{\mathcal{S}}$ be its stationary distribution. Let $f_{\mathcal{S}}$ be the choice of $f$ that attains the minimum in Equation (7) under $\pi = \pi_{\mathcal{S}}$. We will show that there exists $f_{\mathcal{G}} \in \mathbb{R}^{V(\mathcal{G})}$ such that:

$$\frac{\mathcal{E}_{P_{\mathcal{G}},\pi_{\mathcal{G}}}(f_{\mathcal{G}})}{\mathrm{Var}_{\pi_{\mathcal{G}}}(f_{\mathcal{G}})} \leq 4\,\frac{\mathcal{E}_{P_{\mathcal{S}},\pi_{\mathcal{S}}}(f_{\mathcal{S}})}{\mathrm{Var}_{\pi_{\mathcal{S}}}(f_{\mathcal{S}})}. \tag{27}$$

By Lemma 11, this implies our claim, since the left-hand side of Equation (27) bounds $\gamma(\mathcal{G})$ from above and the right-hand side equals $4\gamma(\mathcal{S})$. Now, first, note that $\pi_{\mathcal{S}}(u) = \frac{2}{3}\pi_{\mathcal{G}}(u)$ for all $u \in V(\mathcal{G})$ (the weight of $u$ is the same in $\mathcal{G}$ and $\mathcal{S}$, but the total sum of weights in $\mathcal{S}$ is $\frac{3}{2}$ that of $\mathcal{G}$). Similar calculations show that for all $\{u, v\} \in E(\mathcal{G})$, we have $\pi_{\mathcal{S}}(x_{uv}) = \frac{4}{3d_u}\pi_{\mathcal{G}}(u)$, where $d_u$ is the degree of $u$ in $\mathcal{G}$. Third, observe that, since $f_{\mathcal{S}}$ attains the minimum in Equation (7), then $f_{\mathcal{S}}(x_{uv}) = \frac{f_{\mathcal{S}}(u)+f_{\mathcal{S}}(v)}{2}$ for all $\{u, v\} \in E(\mathcal{G})$. Finally, let $f_{\mathcal{G}}$ be the restriction of $f_{\mathcal{S}}$ to $V(\mathcal{G})$.

First, we compare the numerator of Equation (27) for $\mathcal{S}$ and for $\mathcal{G}$. To begin, note that:

$$\mathcal{E}_{P_{\mathcal{S}},\pi_{\mathcal{S}}}(f_{\mathcal{S}}) = \sum_{\{u,v\}\in E(\mathcal{G})} \Big( (f_{\mathcal{S}}(u) - f_{\mathcal{S}}(x_{uv}))^2\, Q_{\mathcal{S}}(u, x_{uv}) + (f_{\mathcal{S}}(v) - f_{\mathcal{S}}(x_{uv}))^2\, Q_{\mathcal{S}}(u, x_{uv}) \Big). \tag{28}$$

Observe that $Q_{\mathcal{S}}(u, x_{uv}) = Q_{\mathcal{S}}(v, x_{uv}) = \pi_{\mathcal{S}}(u)\frac{1}{2d_u}$, and as noted above, $f_{\mathcal{S}}(x_{uv}) = \frac{f_{\mathcal{S}}(u)+f_{\mathcal{S}}(v)}{2}$, thus $(f_{\mathcal{S}}(u) - f_{\mathcal{S}}(x_{uv})) = (f_{\mathcal{S}}(v) - f_{\mathcal{S}}(x_{uv})) = \frac{1}{2}(f_{\mathcal{S}}(u) - f_{\mathcal{S}}(v))$. Recalling that $\pi_{\mathcal{S}}(u) = \frac{2}{3}\pi_{\mathcal{G}}(u)$,

$$\mathcal{E}_{P_{\mathcal{S}},\pi_{\mathcal{S}}}(f_{\mathcal{S}}) = \frac{1}{2} \sum_{\{u,v\}\in E(\mathcal{G})} (f_{\mathcal{S}}(u) - f_{\mathcal{S}}(v))^2\, \pi_{\mathcal{S}}(u)\frac{1}{2d_u} \tag{29}$$

$$= \frac{1}{3} \sum_{\{u,v\}\in E(\mathcal{G})} (f_{\mathcal{S}}(u) - f_{\mathcal{S}}(v))^2\, \pi_{\mathcal{G}}(u)\frac{1}{2d_u}. \tag{30}$$

However, since by construction $f_{\mathcal{G}}(u) = f_{\mathcal{S}}(u)$ and since $Q_{\mathcal{G}}(u, v) = \pi_{\mathcal{G}}(u)\frac{1}{2d_u}$:

$$\mathcal{E}_{P_{\mathcal{G}},\pi_{\mathcal{G}}}(f_{\mathcal{G}}) = \sum_{\{u,v\}\in E(\mathcal{G})} (f_{\mathcal{G}}(u) - f_{\mathcal{G}}(v))^2\, Q_{\mathcal{G}}(u, v) \tag{31}$$

$$= \sum_{\{u,v\}\in E(\mathcal{G})} (f_{\mathcal{S}}(u) - f_{\mathcal{S}}(v))^2\, \pi_{\mathcal{G}}(u)\frac{1}{2d_u}. \tag{32}$$

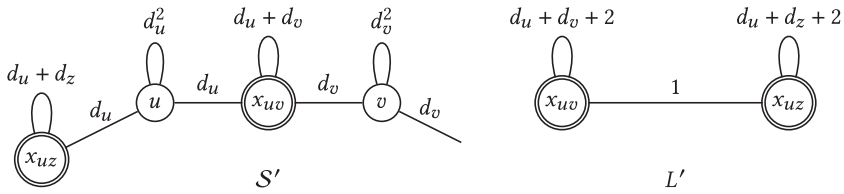


Fig. 2. Left: the graph $\mathcal{S}'$ described above. Right: the reweighted line graph $L'$ obtained by weighting every loop $\{x_{uv}, x_{uv}\}$ of $L(\mathcal{G})$ with $(d_u + d_v + 2)$ instead of $(d_u + d_v - 2)$. The random walk over $L'$ is exactly the random walk over $\mathcal{S}'$ observed only on the set of states $\{x_{uv} : \{u, v\} \in E(\mathcal{G})\}$.

Comparing Equations (30) and (32) shows that $\mathcal{E}_{P_\mathcal{G}, \pi_\mathcal{G}}(f_\mathcal{G}) = 3\,\mathcal{E}_{P_\mathcal{S}, \pi_\mathcal{S}}(f_\mathcal{S})$.

Next, we compare the denominator of Equation (27) for $\mathcal{S}$ and for $\mathcal{G}$. First, we have:

$$\mathrm{Var}_{\pi_\mathcal{S}}(f_\mathcal{S}) = \sum_{u \in V(\mathcal{G})} \pi_\mathcal{S}(u) f_\mathcal{S}(u)^2 + \sum_{\{u,v\} \in E(\mathcal{G})} \pi_\mathcal{S}(x_{uv}) f_\mathcal{S}(x_{uv})^2. \tag{33}$$

Since $\pi_\mathcal{S}(u) = \frac{2}{3}\pi_\mathcal{G}(u)$ and $f_\mathcal{S}(u) = f_\mathcal{G}(u)$, the first term equals $\frac{2}{3}\,\mathrm{Var}_{\pi_\mathcal{G}}(f_\mathcal{G})$. Now, we show that the second term is bounded by $\frac{2}{3}\,\mathrm{Var}_{\pi_\mathcal{G}}(f_\mathcal{G})$. Recalling again that $f_\mathcal{S}(x_{uv}) = \frac{f_\mathcal{S}(u) + f_\mathcal{S}(v)}{2}$:

$$\sum_{\{u,v\} \in E(\mathcal{G})} \pi_\mathcal{S}(x_{uv}) f_\mathcal{S}(x_{uv})^2 = \sum_{\{u,v\} \in E(\mathcal{G})} \pi_\mathcal{S}(x_{uv}) \left( \frac{f_\mathcal{S}(u) + f_\mathcal{S}(v)}{2} \right)^2 \tag{34}$$

$$\leq \sum_{\{u,v\} \in E(\mathcal{G})} \pi_\mathcal{S}(x_{uv}) \frac{1}{2} \left( f_\mathcal{S}(u)^2 + f_\mathcal{S}(v)^2 \right) \tag{35}$$

$$= \sum_{u \in V(\mathcal{G})} \sum_{v : \{u,v\} \in E(\mathcal{G})} \frac{1}{2} \frac{4}{3d_u} \pi_\mathcal{G}(u) f_\mathcal{S}(u)^2 \tag{36}$$

$$= \frac{2}{3} \sum_{u \in V(\mathcal{G})} \pi_\mathcal{G}(u) f_\mathcal{S}(u)^2, \tag{37}$$

where Equation (35) holds by convexity, and Equation (36) holds since every $u \in V(\mathcal{G})$ is charged with $\frac{1}{2}\pi_\mathcal{S}(x_{uv}) f_\mathcal{S}(u)^2$ by every $\{u, v\} \in E(\mathcal{G})$ and since $\pi_\mathcal{S}(x_{uv}) = \frac{4}{3d_u}\pi_\mathcal{G}(u)$. Therefore, $\mathrm{Var}_{\pi_\mathcal{S}}(f_\mathcal{S}) \leq \frac{4}{3}\,\mathrm{Var}_{\pi_\mathcal{G}}(f_\mathcal{G})$, so $\mathrm{Var}_{\pi_\mathcal{G}}(f_\mathcal{G}) \geq \frac{3}{4}\,\mathrm{Var}_{\pi_\mathcal{S}}(f_\mathcal{S})$.

By combining our two bounds, we obtain:

$$\frac{\mathcal{E}_{P_\mathcal{G}, \pi_\mathcal{G}}(f_\mathcal{G})}{\mathrm{Var}_{\pi_\mathcal{G}}(f_\mathcal{G})} \leq \frac{3\,\mathcal{E}_{P_\mathcal{S}, \pi_\mathcal{S}}(f_\mathcal{S})}{\frac{3}{4}\,\mathrm{Var}_{\pi_\mathcal{S}}(f_\mathcal{S})} = 4\,\frac{\mathcal{E}_{P_\mathcal{S}, \pi_\mathcal{S}}(f_\mathcal{S})}{\mathrm{Var}_{\pi_\mathcal{S}}(f_\mathcal{S})}, \tag{38}$$

which shows that $\gamma(\mathcal{G}) \leq 4\gamma(\mathcal{S})$, completing the proof. □

Claim 4. $\tau(L(\mathcal{G})) \leq 5\,\tau(\mathcal{S}')$.

Proof. Let $X = \{X_t\}_{t \geq 0}$ be the walk on $\mathcal{S}'$, and let $Y = X[A]$ be the chain induced by $X$ on the subset of states $A = \{x_{uv} : \{u, v\} \in \mathcal{E}\}$ (Definition 15). Since by Lemma 16 $\tau(Y) \leq \tau(\mathcal{S}')$, we need only to prove that $\tau(L(\mathcal{G})) \leq 5\,\tau(Y)$. To this end, we show that $Y$ is the random walk on the graph $L'$ obtained by weighting $L(\mathcal{G})$ as follows (see Figure 2):

$$w_{L'}(x_{uv}, x_{uz}) = 1 \qquad v \neq z \tag{39}$$

$$w_{L'}(x_{uv}, x_{uv}) = d_u + d_v + 2. \tag{40}$$

To prove the claim, we compute the transition probabilities of $Y$ from $x_{uv}$. First, if $Y_t = x_{uv}$, then we can assume $X_s = x_{uv}$ for some $s = s(t)$. From $x_{uv}$, the possible transitions are $Y_{t+1} = x_{uv}$

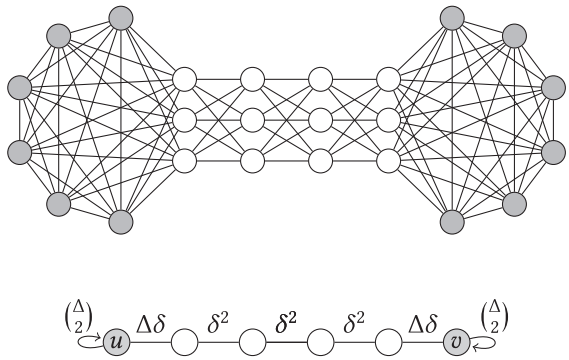


Fig. 3. Above: the graph $G$ for $k = 3$, $\delta = 6$, $\Delta = 3$; clique vertices in gray. Below: $G$ collapsed in a weighted path $P$.

and $Y_{t+1} = x_{uz}$ for some $z \neq v$. The transition $Y_{t+1} = x_{uv}$ happens if and only if one of these three disjoint events occurs:

(1) $X_{s+1} = x_{uv}$,
(2) $X_{s+1} = \ldots = X_{s'-1} = u$ and $X_{s'} = x_{uv}$ for some $s' \geq s + 2$,
(3) the same as (2) but with $v$ in place of $u$.

The probability of event (1) is $\frac{1}{2}$ by construction of the loop weights. The probability of event (2) is the product of $\mathbb{P}(X_{s+1} = u \mid X_s = x_{uv}) = \frac{d_u}{2(d_u+d_v)}$ and $\mathbb{P}(X_{s'} = x_{uv} \mid X_{s'-1} = u) = \frac{1}{d_u}$, since $X$ leaves $u$ with probability 1, in which case it moves to $x_{uv}$ with probability $\frac{1}{d_u}$. Thus, the probability of event (2) is $\frac{1}{2(d_u+d_v)}$, and by symmetry the same is for event (3). Therefore:

$$\mathbb{P}(Y_{t+1} = x_{uv} \mid Y_t = x_{uv}) = \frac{1}{2} + \frac{1}{d_u + d_v} = \frac{d_u + d_v + 2}{2(d_u + d_v)}. \tag{41}$$

The transition $Y_{t+1} = x_{uz}$ is the same as event (2) above, only with $X_{s'} = x_{uz}$ instead of $X_{s'} = x_{uv}$. But conditioned on $X_{s'-1} = u$ the two events have the same probability, therefore:

$$\mathbb{P}(Y_{t+1} = x_{uz} \mid Y_t = x_{uv}) = \frac{1}{2(d_u + d_v)}. \tag{42}$$

Thus, the probabilities are proportional to 1 and $d_u + d_v + 2$, as $w_{L'}$ says.

We can now conclude the proof of the claim. If $d_u + d_v = 2$, then $|E(\mathcal{G})| = 1$, so $L(\mathcal{G})$ is the singleton graph and $\tau(L(\mathcal{G})) = 0$, and $\tau(L(\mathcal{G})) \leq 5\,\tau(\mathcal{S}')$ holds trivially. Suppose instead $d_u + d_v \geq 3$. Then $\frac{d_u+d_v+2}{d_u+d_v-2} \leq \frac{3+2}{3-2} = 5$. Therefore, $w_S \leq w_{L'} \leq 5\,w_{\mathcal{S}'}$, and Lemma 12 yields $\tau(L(\mathcal{G})) \leq 5\,\tau(\mathcal{S}')$. The proof is complete. □

By combining claims 3 and 4, we obtain $\tau(L(\mathcal{G})) \leq 5\,\tau(\mathcal{S}') \leq 20\rho(\mathcal{G})\tau(\mathcal{G})$, proving Lemma 3.

### 5.3 Proof of the Lower Bounds of Theorem 1

We ignore factors depending only on $k$, which are easily proven to be in $k^{O(k)}$. Consider a graph $G$ formed by two disjoint cliques of order $\Delta$, connected by a "fat path" (the Cartesian product of a path and a clique) of length $2(k-1)$ and width $\delta$ (see Figure 3). The total number of vertices is $n = 2\Delta + 2(k-1)\delta) = \Theta(\Delta)$, and we choose $\Delta$ and $\delta$ so $\rho(G) = \frac{\Delta}{\delta} \in \Theta(\rho(n))$.

We start by bounding $t(\mathcal{G}_k)$ from below with a conductance argument. Let $C_u$ be the left clique of $G$, and for $i = 1, \ldots, k-1$, let $L_i$ be the vertices of $G$ at distance $i$ from $C_u$. Let $U$ be the set of all

$k$-graphlets of $G$ containing at least $\frac{k}{2}+1$ vertices from $C_u \cup L_1 \cup \ldots \cup L_{k-1}$, and $\overline{U} = \mathcal{V}_k \setminus U$. Consider the cut between $U$ and $\overline{U}$ in $\mathcal{G}_k$. Observe that $\mathrm{vol}(U) \leq \mathrm{vol}(\overline{U})$, which implies $\Phi(\mathcal{G}_k) \leq \frac{c(U)}{\mathrm{vol}(U)}$. Now, $U$ contains at least $\binom{\Delta}{k} = \Omega(\Delta^k)$ graphlets, each of which has $\Omega(\Delta)$ neighbors in $\mathcal{G}_k$. Hence, $\mathrm{vol}(U) = \Omega(\Delta^{k+1})$. However, consider any $\{g, g'\} \in \mathrm{Cut}(U, \overline{U})$. We claim that $g \cup g'$ is spanned by a tree on $k+1$ vertices that does not intersect the cliques of $G$. Indeed, suppose by contradiction that $g \cap C_u \neq \emptyset$. Since $g \cup g'$ has diameter at most $k$, we deduce that $g' \setminus (C_u \cup L_1 \cup \ldots \cup L_k)$ has size at most 1. This contradicts the fact that $g' \in \overline{U}$, which would require $|g' \setminus (C_u \cup L_1 \cup \ldots \cup L_k)| \geq \frac{k}{2}$, which is strictly larger than 1, since $k \geq 3$. A symmetric argument proves that $g \cup g'$ does not intersect the right clique of $G$. Hence, $g \cup g'$ is spanned by a tree on $k+1$ vertices of the fat path, and the number of such trees is $O(\delta^{k+1})$. Therefore, $c(U) = O(\delta^{k+1})$. By Equation (4), we conclude that:

$$\tau(\mathcal{G}_k) \geq \frac{1}{4\Phi(\mathcal{G}_k)} = \Omega\left(\rho(n)^{k+1}\right). \tag{43}$$

Now, we show that $\rho(n)^2 = \Omega(\tau(G))$. Let $P$ be the weighted path graph obtained from $G$ by identifying the vertices in each clique and the vertices in every layer of the path (see the figure again). Let $X = \{X_i\}_{i \geq 0}$ be the random walk over $G$, and for all $i \geq 0$ let $Y_i$ be the vertex of $V(P)$ corresponding to $X_i$. Note that $Y = \{Y_i\}_{i \geq 0}$ is the random walk over $P$, and that it is coupled to $X$. Now observe that, for any $i \geq 1$, if $Y_i$ is at total variation distance $\varepsilon$ from the stationary distribution $\pi_Y$ of $Y$, then $X_i$ is at total variation distance $\varepsilon$ from the stationary distribution $\pi_X$ of $X$. Therefore, $t(G) = O(t(P))$, which implies $\tau(G) = O(t(P))$. In turn, $P$ is a path of constant length whose edge weights are in the range $[\delta^2, \Delta^2]$. By Lemma 12, this implies that $t(P)$ is within $O(\rho(n)^2)$ times the mixing time of the walk on the unweighted version of $P$, which is constant. Therefore, $\tau(G) = O(\rho(n)^2)$, i.e., $\rho(n)^2 = \Omega(\tau(G))$.

Combining Equation (43) with the fact that $\rho(n)^2 = \Omega(\tau(G))$ and that $t(\mathcal{G}_k) = \Omega(\tau(\mathcal{G}_k))$ yields the lower bound of Theorem 1.

### 5.4 Proof of Theorem 2

First we prove two ancillary facts, and then Theorem 2. Throughout the proofs, we assume the model of [20, 26], so we can check for the existence of any edge in $G$ in time $O(1)$. The last part of Theorem 2 follows easily: One can check for the existence of an edge in time $O(\Delta)$ without any preprocessing or in time $O(\log \Delta)$ after sorting the adjacency lists of $G$ in time $O(n+m)$.

Lemma 18. *For every $g \in V(\mathcal{G}_k)$ let $T(g) = \{\{u, v\} \in E(\mathcal{G}_{k-1}) : u \cup v = g\}$. Then $|T(g)| \leq \binom{k}{2}$, and given $g$, we can compute $|T(g)|$ in time $O(\mathrm{poly}(k))$.*

Proof. Every $\{u, v\} \in T(g)$ satisfies: (i) $u = g \setminus \{x\}$ and $v = g \setminus \{y\}$ for some $x, y \in g$, and (ii) $u$, $v$, and $u \cap v$ are connected. Thus, given $g$, we can just enumerate all $\binom{k}{2}$ pairs of vertices in $g$ and count which ones have $u$, $v$, and $u \cap v$ connected. This gives the bound on $|T(g)|$, too. □

Lemma 19. *Any single step of the lazy walk over $\mathcal{G}_k$ can be simulated in $\mathrm{poly}(k)$ expected time.*

Proof. To decide whether to follow the loop, we just toss a fair coin. Let us now see how to transition to a neighboring graphlet uniformly at random. Let $g \in V(\mathcal{G}_k)$ be the current vertex of the walk and let $N(g)$ be the set of neighbors of $g$ in $\mathcal{G}_k$. For every $y \in V(g)$, consider the following cut in $G$:

$$C(g, y) = \mathrm{Cut}(y, V(G) \setminus V(g)). \tag{44}$$

Clearly, for every edge $\{y, y'\} \in C(g, y)$, the graphlet $g \cup y' \setminus x$ is adjacent to $g$ in $\mathcal{G}_k$, provided that $g \setminus x$ is connected. Moreover, $|C(g, y)|$ can be computed in time $O(k)$, as the difference between $d_y$ and the number of neighbors of $y$ in $G$.

Now, for every $x \in V(g)$ let $c(x) = \sum_{y \in V(g) \setminus x} |C(g, y)|$ if $g \setminus x$ is connected, and $c(x) = 0$ otherwise. Finally, let $c = \sum_{x \in V(g)} c(x)$. We draw $g'$ at random as follows: First, we draw $x \in V(g)$ at random with probability $c(x)/c$. Then, we draw $y \in V(g) \setminus x$ at random with probability $|C(g, y)|/c(x)$. Finally, we select an edge $\{y, y'\}$ uniformly at random in $C(g, y)$. To do this, we just sample $y'$ uniformly at random from the neighbors of $y$ in $G$ until hitting $V(G) \setminus V(g)$. This requires at most $k$ trials in expectation, since $y$ has at most $k - 1$ neighbors in $V(g)$ and has at least one neighbor in $V(G) \setminus V(g)$, otherwise $|C(g, y)|/c(x) = 0$ and we would not have chosen $y$.

Now consider any $g' \sim g$. Note that $g'$ is identified by the pair $(x, y')$ where $\{x\} = V(g) \setminus V(g')$ and $\{y'\} = V(g') \setminus V(g)$. The probability that the random process above generates $g'$ is:

$$\frac{c(x)}{c} \cdot \sum_{\substack{y \in V(g) \setminus x \\ y' \sim y}} \frac{|C(g, y)|}{c(x)} \cdot \frac{1}{|C(g, y)|} = \frac{1}{c} \left| \{y \in V(g) \setminus x : y' \sim y\} \right|, \tag{45}$$

that is, equal for all $g'$ up to a multiplicative factor between 1 and $k$. However, once we have drawn $(x, y')$, we can compute $|\{y \in V(g) \setminus x : y' \sim y\}|$ in time poly$(k)$ and apply rejection sampling to make the output distribution uniform. The expected number of rejection trials is in poly$(k)$ as well and so is the expected running time of the entire process. □

We can now prove Theorem 2. Consider $\mathcal{G}_{k-1}$. By construction, $\{u, v\} \in E(\mathcal{G}_{k-1})$ if and only if $g = u \cup v \in V(\mathcal{G}_{k-1})$. Recall from Lemma 18 the set $T(g)$ and that $|T(g)| \leq k^2$. Hence, if we draw from a $O(\frac{\varepsilon}{k^2})$-uniform distribution over $E(\mathcal{G}_{k-1})$ and accept the sampled edge $\{u, v\}$ with probability $\frac{1}{T(g)}$ where $g = u \cup v$, then the distribution of accepted graphlets will be $\varepsilon$-uniform. Let then $X = \{X_t\}_{t \geq 0}$ be the lazy random walk over $\mathcal{G}_{k-1}$, and for all $t \geq 0$ let $Y_t = X_t \cup X_{t+1}$. Then, $Y_t$ is $O(\frac{\varepsilon}{k^2})$-uniform over $E(\mathcal{G}_{k-1})$ if $X_t$ is $O(\frac{\varepsilon}{k^2})$-uniform distribution over $V(\mathcal{G}_{k-1})$. This holds since the distributions $\pi_t$ of $X_t$ and $\sigma_t$ of $Y_t$ satisfy $\sigma_t = M\pi_t$, where $M$ is a stochastic matrix. Since for stochastic matrices $\|M\|_1 \leq 1$, we have $\|\sigma_t - \sigma\|_1 \leq \|M(\pi_t - \pi)\|_1 \leq \|M\|_1 \|\pi_t - \pi\|_1$ where $\pi$ and $\sigma$ are the stationary distributions of $Y_t$ and $X_t$. Thus, we just need to run the walk over $\mathcal{G}_{k-1}$ for $t_{\varepsilon_k}(\mathcal{G}_{k-1})$ steps where $\varepsilon_k = \Theta(\frac{\varepsilon}{k^2})$. From the proof of Theorem 1, one can immediately see that $t_{\varepsilon_k}(\mathcal{G}_{k-1}) = k^{O(k)} O(t_\varepsilon(G)\, \rho(G)^{k-2} \log \frac{n}{\varepsilon})$. (The $\frac{1}{k^2}$ factor in $\varepsilon_k$ is absorbed by $k^{O(k)}$). Finally, by Lemma 19, each step takes $O(\text{poly}(k))$ time in expectation. This completes the proof.

## 6 UNIFORM GRAPHLET SAMPLING

This section presents our uniform graphlet sampling algorithm, Ugs. The key idea of the algorithm is to make rejection sampling efficient. To understand how, let us first describe why rejection sampling is usually *not* efficient. Suppose we have a generic random process that draws graphlets from $\mathcal{V}_k$. For each graphlet $g \in \mathcal{V}_k$ let $p(g)$ be the probability that the process yields $g$, and let $p^* = \min_{g \in \mathcal{V}_k} p(g)$. In rejection sampling, when we draw $g$, we randomly *accept* it with probability $p^* p(g)^{-1}$. In this way, the probability that $g$ is returned, which equals the probability that $g$ is both sampled *and* accepted, is $p(g)\, p^* p(g)^{-1} = p^*$, which is independent of $g$. This makes the distribution of returned graphlets uniform regardless of $p$. The key problem is that $p^*$ may be very small—which happens, for instance, if the random process samples graphlets by growing a random spanning tree around a high-degree vertex of $G$. In this case, we can have $p^* = O(\Delta^{-(k-1)})$, so we may need $\Omega(\Delta^{k-1})$ trials before accepting a graphlet. Unfortunately, all known graphlet sampling algorithms based on rejection sampling suffer from this "curse of rejection," and indeed they may need time $\Omega(\Delta^{k-1})$ for sampling just one uniform graphlet in the worst case.

The main idea of UGS is to circumvent the obstacle by sorting $G$. By doing this, we will implicitly partition $\mathcal{V}_k$ into $n$ buckets $B(1), \ldots, B(n)$, one for each vertex of $G$, in such a way that for each $v \in V(G)$ we will know $|B(v)|$ with good accuracy. This will constitute our preprocessing phase. In the sampling phase, we will pick $v$ with probability proportional to our estimate of $|B(v)|$, and we will sample almost-uniformly from $B(v)$. To this end, we note that sampling from $B(v)$ amounts to sampling a $k$-graphlet from the subgraph $G(v)$ of $G$ induced by $v$ and all vertices after $v$ in the order. This can be done efficiently, since, as we will see, for our purposes $G(v)$ behaves like a regular graph. Moreover, we will be able to compute efficiently all the probabilities involved in this process. This will allow us to reject the sampled subgraph efficiently and with the correct probability, guaranteeing a truly uniform distribution.

### 6.1 A Toy Example: Regular Graphs

Let us build the intuition with a toy example. Suppose that $G$ is $d$-regular. For simplicity, suppose that $G$ is connected, too. To begin, we let $p(v) = \frac{1}{n}$ for all $v \in V(G)$ and choose $v$ according to $p$, i.e., uniformly at random. Note that $p(v)$ is roughly proportional to the number of $k$-graphlets containing $v$, which is easily seen to be between $k^{-O(k)}d^{k-1}$ and $k^{O(k)}d^{k-1}$ for all $v$. Once we have chosen $v$, we sample a graphlet containing $v$ by running the following random growing process: Set $S_1 = \{v\}$, and for $i = 2, \ldots, k$, build $S_i$ from $S_{i-1}$ by choosing a random edge in the cut between $S_{i-1}$ and the rest of $G$ and adding to $S_{i-1}$ the other endpoint of the edge. Denote by $p_v(g)$ the probability that $g$ is obtained when the random growing process starts at $v$, and by $p(g) = \sum_{v \in g} p(v)p_v(g)$ the probability that $g$ is obtained. It is easy to show that for any $g \in \mathcal{V}_k$ we have:

$$\frac{1}{n}k^{-O(k)}d^{-(k-1)} \leq p(g) \leq \frac{1}{n}k^{O(k)}d^{-(k-1)}. \tag{46}$$

Now, we design the rejection step. First, observe that by setting $p^* = \frac{1}{n}k^{-Ck}d^{-(k-1)}$ with $C$ large enough, for all $g \in \mathcal{V}_k$, we will have $p(g) \geq p^*$ and therefore $p^*p(g)^{-1} \leq 1$. Moreover, in time $k^{O(k)}$ we can easily compute $p(g)$ for any given $g$ (this is shown below). In summary, once we have sampled $g$ we can efficiently compute $p_{acc}(g) = p^*p(g)^{-1} \leq 1$. Then, we accept $g$ with probability $p_{acc}(g)$. The probability that $g$ is sampled *and* accepted is $p(g)p_{acc}(g) = p^*$, which is independent of $g$. Therefore, the distribution of the returned $k$-graphlets is uniform over $\mathcal{V}_k$. Moreover, by the inequalities above we have $p_{acc}(g) \geq k^{-O(k)}$, hence we will terminate after $k^{O(k)}$ rejection trials in expectation. Thus, when $G$ is $d$-regular we have an efficient uniform graphlet sampling algorithm.

### 6.2 The Preprocessing Phase

Let $G$ be an arbitrary graph. Our goal is to "regularize" $G$, in a certain sense, so we can apply the scheme of the toy example above. Let us start by introducing some notation. Given an order $\prec$ over $V$, we denote by $G(v) = G[\{u \succeq v\}]$ the subgraph of $G$ induced by $v$ and all vertices after it in the order, and for all $u \in G(v)$, we denote by $d(u|G(v))$ the degree of $u$ in $G(v)$. Before moving to the algorithm, we introduce a definition that is central to the rest of the work.

*Definition 20.* $\prec$ is an **$\alpha$-degree-dominating order ($\alpha$-DD order)** of $G$ if for all $v$ and all $u \succ v$ we have $d(v|G(v)) \geq \alpha\, d(u|G(v))$.

Our algorithm starts by computing a 1-dominating order for $G$, which guarantees that $v$ has the largest degree in $G(v)$. Such an order can be easily computed in time $O(n + m)$ by repeatedly removing from $G$ the vertex of maximum degree [30]. (Later on, we will need to compute approximate $\alpha$-DD orders for $\alpha < 1$ in time roughly $O(n \log n)$, which is not as easy.) After computing our 1-DD order $\prec$, in time $O(n + m)$ we also sort the adjacency lists of $G$ accordingly, via bucket sort. This will be used to find efficiently the edges of $G(v)$ via binary search.

Next, we (virtually) partition graphlets into *buckets*.

*Definition 21.* The bucket $B(v)$ is the set of graphlets whose smallest vertex according to $\prec$ is $v$.

Clearly, the buckets $B(v)$ form a partition of $\mathcal{V}_k$. Similarly to $d^{k-1}$ in the toy example above, here $d(v|G(v))^{k-1}$ gives a rough estimate of the number of graphlets in $B(v)$. Indeed, if $B(v) \neq \emptyset$, then we can easily show that:

$$k^{-O(k)}|B(v)| \leq d(v|G(v))^{k-1} \leq k^{O(k)}|B(v)|. \tag{47}$$

It is easy to see that the $d(v|G(v))$ are known after computing $\prec$. Hence, we will use $d(v|G(v))^{k-1}$ as a proxy for $|B(v)|$. In time $O(n)$, we compute:

$$Z = \sum_{v \in V : B(v) \neq \emptyset} d(v|G(v))^{k-1} \tag{48}$$

$$p(v) = \mathbb{I}\{B(v) \neq \emptyset\} \cdot \frac{d(v|G(v))^{k-1}}{Z}, \quad \forall v \in V(G). \tag{49}$$

This defines a distribution $p$ over the buckets that we will use in the sampling phase. Note that, to compute $Z$ and $p$, we must detect whether $B(v) = \emptyset$ for each $v$. To this end, we use a BFS from $v$ that explores $G(v)$ and stops as soon as $k$ vertices are found. We can show that this takes time $O(k^2 \log k)$ by listing edges from the end of the adjacency lists. This makes our overall preprocessing time grow to $O(nk^2 \log k + m)$, as claimed in Theorem 4. This concludes our preprocessing phase. See Algorithm 1 for the pseudocode.

**ALGORITHM 1:** DD($G$)

```
compute ≺ using a bucketing algorithm                    ▷ O(n + m)
sort the adjacency lists of G according to ≺             ▷ O(n + m)
compute d(v|G(v)) for all v ∈ V(G)                       ▷ O(n + m)
for each v ∈ V do
    check if B(v) ≠ ∅ with a BFS                         ▷ O(k^2 log k)
    let b_v = I{B(v) ≠ ∅} · d(v|G(v))^{k-1}
return ≺ and {b_v}_{v∈V}
```

Lemma 22. *DD($G$) can be implemented to run in time $O(nk^2 \log k + m)$. The output order $\prec$ is a 1-DD order for $G$ and thus satisfies $d(v|G(v)) \geq d(u|G(v))$ for all $u \succ v$. The output estimates $b_v > 0$ satisfy $\frac{b_v}{|B(v)|} \in [k^{-O(k)}, k^{O(k)}]$.*

Proof. Consider the first two lines of DD($G$). Computing $\prec$ takes time $O(n + m)$ by a standard bucketing technique [30]. Sorting the adjacency lists of $G$ according to $\prec$ takes time $O(n + m)$ via bucket sort. With one final $O(n + m)$-time pass, we compute, for each $v$, the position $i_v$ of $v$ in its own sorted adjacency list, from which we compute $d(v|G(v)) = d_v - i_v$ in constant time for each $v$.

Now consider the main loop. Clearly, $B(v) \neq \emptyset$ if and only if $k$ vertices are reachable from $v$ in $G(v)$. Thus, we perform a BFS in $G(v)$, starting from $v$, and stopping as soon as $k$ pushes have been made on the queue (counting $v$ as well). To keep track of which vertices have been pushed, we can use a dictionary; as we need to hold at most $k$ entries, every insertion and lookup will take time $O(\log k)$. After popping a generic vertex $u$ from the queue, we proceed as follows: We take every neighbor $z$ of $u$ in reverse order (that is, according to $\succ$). If $z \succ v$ and $z$ has not been pushed, then we push it. As soon as we encounter $z \prec u$, we stop and pop a new vertex from the queue. Suppose that, after popping $u$, we examine $\ell$ of its neighbors. Then, at least $\ell + 1$ vertices must

have been pushed so far, since $u$ itself was pushed, and every neighbor examined was certainly pushed (before or when examined). Thus, for every vertex $u$, we examine at most $k-1$ neighbors (since we stop the whole algorithm as soon as $k$ vertices are pushed). Since we push at most $k$ vertices in total, we also pop at most $k$ vertices in total. Therefore, we examine a total of $O(k^2)$ vertices. Thus, we spend a total time $O(k^2 \log k)$. Summarizing, we obtain a total time bound of $O(n+m) + O(nk^2 \log k) = O(nk^2 \log k + m)$.

The claim on $b_v$ follows by Lemma 45, since in $G(v)$ vertex $v$ has maximum degree $d(v|G(v))$. □

### 6.3 The Sampling Phase

The sampling phase starts by drawing a vertex $v$ from the distribution $p$. Using the alias method [40], each such random draw takes time $O(1)$ after a $O(n)$-time-and-space preprocessing (which we do in the preprocessing phase). Once we have drawn $v$, we draw a graphlet from $B(v)$ using what we call the *random growing process* at $v$. This is the same process used in the toy example above, but restricted to the subgraph $G(v)$.

*Definition 23.* The *random growing process* at $v$ is defined as follows: $S_1 = \{v\}$, and for each $i = 1, \ldots, k-1$, $S_{i+1} = S_i \cup \{u_i, u_i'\}$, where $\{u_i, u_i'\}$ is uniform random over $\mathrm{Cut}(S_i, G(v) \setminus S_i)$.

Now, we make two key observations. First, the random growing process at $v$ returns a roughly uniform random graphlet of $B(v)$ and can be implemented efficiently, thanks to the sorted adjacency lists of $G$. Second, the probability that the random growing process returns a specific graphlet $g$ can be computed efficiently, thanks again to the sorted adjacency lists. These two facts are proven below; before, however, we need a technical result about the size of the cuts in $G(v)$.

Lemma 24. *Let $V(G)$ be sorted according to a 1-DD order. Consider any sequence of sets $S_1, \ldots, S_{k-1}$ such that $S_1 = \{v\}$, that $G(v)[S_i]$ is connected for all $i$, and that $S_i = S_{i-1} \cup \{s_i\}$ for some $s_i \in G(v)$. Then for all $i = 1, \ldots, k-1$:*

$$i^{-1} \leq \frac{|\mathrm{Cut}(S_i, G(v) \setminus S_i)|}{d(v|G(v))} \leq i. \tag{50}$$

Proof. Let $c_i = |\mathrm{Cut}(S_i, G(v) \setminus S_i)|$ for short. For the lower bound, note that $c_i \geq 1$ for all $i = 1, \ldots, k-1$, since $G[S_k]$ is connected. Moreover $c_1 = d(v|G(v))$, since $S_1 = \{v\}$. Now, if $c_1 \leq i$, then $\frac{c_1}{i} \leq 1$ and therefore $c_i \geq \frac{c_1}{i}$. If instead $c_1 \geq i$, since the degree of $v$ in $S_i$ is at most $i-1$, then the cut of $S_i$ still contains at least $c_1 - |S_i| + 1 \geq c_1 - (i-1)$ edges. Therefore:

$$c_i \geq c_1 - (i-1) \geq c_1 - (i-1)\frac{c_1}{i} = \frac{c_1}{i} = \frac{1}{i} \cdot d(v|G(v).) \tag{51}$$

For the upper bound, note that:

$$c_i \leq \sum_{u \in S_i} d(u|G(v)) \leq \sum_{u \in S_i} d(v|G(v)) = i \cdot d(v|G(v),) \tag{52}$$

where we used the fact that $v$ is the maximum-degree vertex of $G(v)$. □

Algorithms Rand-Grow$(G, v)$ and Prob$(G, S)$ give the pseudocode of the random growing process and of the algorithm for computing the probability that the process returns a particular graphlet. Lemma 25 shows that Rand-Grow$(G, v)$ can be implemented efficiently and that it returns a graphlet that is roughly uniform. Lemma 26 shows that Prob$(G, S)$ is correct and efficient.

Lemma 25. *Suppose $G$ is sorted according to a 1-DD order and choose any $v$ such that $B(v) \neq \emptyset$. Then Rand-Grow$(G, v)$ runs in time $O(k^3 \log \Delta)$. Moreover, for any $g = G[S] \in B(v)$, the probability $p(S)$ that Rand-Grow$(G, v)$ returns $S$ is between $\frac{1}{(k-1)!} d(v|G(v))^{-(k-1)}$ and $(k-1)!^3 d(v|G(v))^{-(k-1)}$.*

**ALGORITHM 2:** RAND-GROW$(G, v)$

$S_1 = \{v\}$
**for** $i = 1, \ldots, k-1$ **do**
  **for** $u \in S_i$ **do**
    $c_i(u) = d(u|G(v))$−(degree of $u$ in $G[S_i]$)
  draw $u$ with probability $\frac{c_i(u)}{\sum_{z \in S_i} c_i(z)}$
  draw $u'$ u.a.r. from the neighbors of $u$ in $G(v) \setminus S_i$
  $S_{i+1} = S_i \cup \{u'\}$
**return** $S_k$

**ALGORITHM 3:** PROB$(G, S = \{v, u_2, \ldots, u_k\})$

$p = 0$
**for** each permutation $\sigma = (\sigma_2, \ldots, \sigma_k)$ of $u_2, \ldots, u_k$ **do**
  $p_\sigma = 1$
  **for** each $i = 1, \ldots, k-1$ **do**
    $S_i = \{v, \sigma_2, \ldots, \sigma_i\}$
    $n_i$ = number of neighbors of $\sigma_{i+1}$ in $S_i$
    $c_i(u) = d(u|G(v))$−(degree of $u$ in $G[S_i]$)
    $p_\sigma = p_\sigma \cdot \frac{n_i}{c_i}$
  $p = p + p_\sigma$
**return** $p$

PROOF. *Running time.* Consider one iteration of the main loop. For every $u \in S_i$, computing $c_i(u)$ takes time $O(k \log \Delta)$. Indeed, using neighbor queries and binary search, in time $O(\log \Delta)$, we locate the position of $v$ in the adjacency list of $u$, which subtracted from $d_u$ yields $d(u|G(v))$. Similarly, in time $O(k \log \Delta)$, we can compute the number of neighbors of $u$ in $S_i$. Summed over all $u \in S_i$ this consumes time $O(k^2 \log \Delta)$ in total. Drawing $u$ takes time $O(k)$. Finally, drawing $u'$ takes $O(k)$ as well. To see this, note that if $u$ had no neighbors in $S_i$, then we could just draw a vertex uniformly at random from the last $d(u|G(v))$ elements of the adjacency list of $u$. However, $u$ has neighbors in $S_i$. But, we still know the (at most $k$) disjoint sublists of the adjacency lists containing the neighbors in the cut. Thus, we can draw a uniform integer $j \in [c_i(u)]$ and select the $j$th neighbor of $u$ in $G(v) \setminus S_i$ in time $O(k)$. This proves that one iteration of the main loop of RAND-GROW takes time $O(k^2 \log \Delta)$. Thus, RAND-GROW runs in time $O(k^3 \log \Delta)$.

*Probability.* Consider any $S$ such that $g = G[S] \in B(v)$. Thus, $S$ is a $k$-vertex subset such that $v \in S$ and that $G[S]$ is connected. We compute an upper bound and a lower bound on the probability $p(S)$ that the algorithm returns $S$.

Clearly, there are at most $(k-1)!$ sequences of vertices that RAND-GROW$(G, v)$ can choose to produce $S$. Fix any such sequence, $v, u_2, \ldots, u_k$, and let $S_i = \{v, \ldots, u_i\}$. Let $c_i(u) = |\operatorname{Cut}(u, G(v) \setminus S_i)|$, and let $c_i = \sum_{u \in S_i} c_i(u) = |\operatorname{Cut}(S_i, G(v) \setminus S_i|$. By construction, $S_{i+1}$ is obtained by adding to $S_i$ the endpoint $u_{i+1}$ of an edge chosen uniformly at random in $\operatorname{Cut}(S_i, G(v) \setminus S_i)$. Thus, for any $u' \in G(v) \setminus S_i$, we have:

$$\mathbb{P}(u_{i+1} = u') = \frac{d(u'|S_i \cup u')}{c_i} \leq \frac{i^2}{d(v|G(v))}, \tag{53}$$

where in the inequality we used the facts that $d(u'|S_i \cup u') \leq i$ is the number of neighbors of $u'$ in $S_i$, and that $c_i \geq \frac{d(v|G(v))}{i}$ by Lemma 24. Thus, the probability that RAND-GROW$(G, v)$ draws the

particular sequence $v, u_2, \ldots, u_k$ is at most $\prod_{i=1}^{k-1} \frac{i^2}{d(v|G(v))} = (k-1)!^2 d(v|G(v))^{-(k-1)}$. Since there are at most $(k-1)!$ sequences, $p(S) \leq (k-1)!^3 d(v|G(v))^{-(k-1)}$.

However, since $G[S]$ is connected, then there is at least one sequence $v, u_2, \ldots, u_k$ such that $c_i \geq 1$ for all $i = 1, \ldots, k-1$, which therefore satisfies:

$$\mathbb{P}(u_{i+1} = u') = \frac{d(u'|S_i \cup u')}{c_i} \geq \frac{1}{i\, d(v|G(v))}, \tag{54}$$

where we used the facts that $d(u'|S_i \cup u') \geq 1$, since $u'$ is a neighbor of some $u \in S_i$, and that $c_i \leq i \cdot d(v|G(v))$, by Lemma 24. So, the probability that Rand-Grow$(G, v)$ draws this particular sequence is at least $\prod_{i=1}^{k-1} \frac{1}{i d(v|G(v))} = \frac{1}{(k-1)!} d(v|G(v))^{-(k-1)}$, which is a lower bound on $p(S)$. □

Lemma 26. *Prob$(G, S = \{v, u_2, \ldots, u_k\})$ runs in time* poly$(k)O(k! \log \Delta)$ *and outputs the probability $p(S)$ that Rand-Grow$(G, v)$ returns $S$.*

Proof. The proof is essentially the same of Lemma 25. □

We can now complete the sampling phase by performing a rejection step. After drawing $v$ with probability $p(v)$, we draw a random graphlet $g$ from $B(v)$ by invoking Rand-Grow$(G, v)$, and we compute $p_v(g)$ by invoking Prob$(G, g)$. By construction, the overall probability that we have drawn $g$ is $p(g) = p(v) \cdot p_v(g)$. By the definition of $p(v)$ and by Lemma 25:

$$k^{-O(k)} \frac{1}{Z} \leq p(v) \cdot p_v(g) \leq k^{O(k)} \frac{1}{Z}. \tag{55}$$

We therefore set the acceptance probability to:

$$p_{acc}(g) = \frac{k^{-Ck}}{p(v) \cdot p_v(g) \cdot Z}. \tag{56}$$

This makes the probability that $g$ is sampled *and* accepted equal to:

$$p(g) \cdot p_{acc}(g) = p(v) \cdot p_v(g) \cdot \frac{k^{-Ck}}{p(v) \cdot p_v(g) \cdot Z} = \frac{k^{-Ck}}{Z}, \tag{57}$$

which is independent of $g$ and thus constant over $\mathcal{V}_k$. For $C$ large enough, Equations (55) and (56) imply $p_{acc} \in [k^{-O(k)}, 1]$. Therefore, $p_{acc}(g)$ is a valid probability, and moreover, we will accept a graphlet after $k^{O(k)}$ trials in expectation. As by Lemmas 25 and 26 the running time of a single trial is poly$(k) \log \Delta$, the total expected time per sample is $k^{O(k)} \log \Delta$, as claimed in Theorem 4.

To wrap up, Algorithm 4 gives the main body of Ugs.

**ALGORITHM 4:** Ugs$(G)$

```
(≺, {b_v}_{v∈V}) = DD(G)
let Z = Σ_{v∈V} b_v
let p(v) = b_v / Z for each v ∈ V
let β_k(G) = 1 / (k! Z)

function SAMPLE()
    while true do
        draw v from the distribution p
        S = Rand-Grow(G, v)
        p(S) = Prob(G, S)
        with probability β_k(G) / (p(v) p(S)) return S
```

## 7 EPSILON-UNIFORM GRAPHLET SAMPLING

This section describes our $\varepsilon$-uniform graphlet sampling algorithm, Apx-Ugs. At a high level, Apx-Ugs is an adaptation of Ugs. To begin, we observe that Ugs relies on the following key ingredients. First, the vertices of $G$ are sorted according to a 1-DD order $\prec$, which ensures that each subgraph $G(v)$ behaves like a regular graph for what concerns sampling (Lemma 25). Second, the edges of $G$ are sorted according to $\prec$ as well, which makes it possible to compute the size of the cuts $|\operatorname{Cut}(u, G(v) \setminus S_i)|$ in time proportional to $\log \Delta$ (Lemmas 25 and 26). Unfortunately, both ingredients require a $\Theta(m)$-time preprocessing. To reduce the preprocessing time to $O(n \log n)$, we introduce:

(1) A preprocessing routine that computes w.h.p. an approximate $\alpha$-DD order, together with good bucket size estimates. By "approximate," we also mean that some buckets might be erroneously deemed empty, but we guarantee that those buckets contain a fraction $\leq \varepsilon$ of all graphlets.
(2) A sampling routine that emulates the one of Ugs, but replaces the exact cut sizes with additive approximations. These approximations are good enough that, with good probability, Apx-Ugs behaves as Ugs, including the rejection step.

Achieving these guarantees is not just a matter of sampling and concentration bounds. For instance, to obtain an $\alpha$-DD order, we cannot just sub-sample the edges of $G$ and compute the 1-DD order on the resulting subgraph: The sorting process would introduce correlations, destroying concentration. Similarly, we cannot just compute a multiplicative estimate of $|\operatorname{Cut}(u, G(v) \setminus S_i)|$: Without sorted lists, this would require $\Omega(\Delta)$ queries, as we might have $d_u = \Delta$ and $|\operatorname{Cut}(u, G(v) \setminus S_i)| = 1$. Similar obstacles arise in estimating $p_v(g)$.

### 7.1 Approximating a Degree-dominating Order

We introduce our notion of approximate degree-dominating order. In what follows, $\boldsymbol{b} = (b_v)_{v \in V}$ is a vector of bucket size estimates.

*Definition 27.* A pair $(\prec, \boldsymbol{b})$ where $\boldsymbol{b} = (b_v)_{v \in V}$ is an $(\alpha, \beta)$-DD order for $G$ if:

(1) $\sum_{v : b_v > 0} |B(v)| \geq (1 - \beta) \sum_v |B(v)|$,
(2) $b_v > 0 \implies k^{-O(k)} \beta \leq \frac{b_v}{|B(v)|} \leq k^{O(k)} \frac{1}{\beta}$,
(3) $b_v > 0 \implies d(v|G(v)) \geq \alpha\, d_v \geq \alpha\, d(u|G(v))$ for all $u \succ v$,
(4) $v \prec u \implies d_v \geq 3k\alpha\, d_u$ .

Let us elaborate on this. The first property says that the buckets that are deemed nonempty hold a fraction $1-\beta$ of all graphlets. The second property says that every bucket that is deemed nonempty comes with a good estimate of its size. The third property says that $\prec$ is an $\alpha$-DD order if restricted to the buckets that are deemed nonempty and gives an additional guarantee on $d_v$. The fourth property will be used later on. The idea is that, if we look only at buckets that are deemed nonempty, then we will have guarantees similar to a 1-DD order; but bear in mind that here the edges of $G$ will not be sorted, and this will complicate things significantly.

The algorithm below, Apx-DD$(G, \beta)$, computes efficiently an $(\alpha, \beta)$-DD order with $\alpha = \beta^{\frac{1}{k-1}} \frac{1}{6k^3}$. This will be enough for our purposes. In the remainder, we prove Lemmas 28 and 29, which give the guarantees of Apx-DD$(G, \beta)$. For technical reasons, instead of $\alpha$ the proofs and the algorithm use $\eta = \alpha k = \varepsilon^{\frac{1}{k-1}} \frac{1}{6k^2}$. The intuition of the algorithm is the following: We start at round $t = 0$ with $\prec_0$ being the order of $V(G)$ by nonincreasing degree; this corresponds to the optimistic guess that $d(v|G(v)) = d_v$ for all $v$. Then, we take every $v \in V(G)$ in the order of $\prec_0$, and we check if $d(v|G(v))$ is indeed close of $d_v$. To this end, we sample $\frac{1}{\eta^2} \log n$ random neighbors of $v$ for some

appropriate $\eta$ and check how many are after $v$ in $\prec_t$. If that fraction is at least $\eta$, then we let $b_v = (d_v)^{k-1}$ and set $\prec_{t+1}=\prec_t$. Otherwise, we let $b_v = 0$ and update $\prec_{t+1}$ from $\prec_t$ by pushing $v$ to its "correct" position. This is enough for vertices of sufficiently high degree, but not for those of small degree. Indeed, for those vertices $B(v)$ might be empty even though $d(v|G(v))$ is close to $d_v$, just because $d(v|G(v))$ is small in an absolute sense. Hence, for vertices of small degree, we check whether $B(v) \neq \emptyset$ explicitly.

A crucial point of the algorithm is that it does *not* maintain $\prec_t$ explicitly; this would cost too much. Instead, the algorithm maintains a set of counters $\{s_u : u \in V(G)\}$ and then computes and returns an ordering $\prec$ such that $u \prec v$ if and only if $(s_u > s_v) \vee ((s_u = s_v) \wedge (u > v))$ for all distinct $u, v \in V(G)$. The analysis, however, considers $\prec$ as evolving with $t$, as described intuitively above.

**ALGORITHM 5:** Apx-DD$(G, \beta)$

let $\eta = \beta^{\frac{1}{k-1}} \frac{1}{6k^2}$ and $h = \Theta(\eta^{-2} \log n)$
init $s_v = d_v$ for all $v \in V$
**for** each $v$ in $V$ in nonincreasing order of degree **do**
    sample $h$ neighbors $x_1, \ldots, x_h$ of $v$ u.a.r.
    let $X = \sum_{j=1}^{h} \mathbb{I}\{x_j \succ v\}$
    **if** $X \geq 2\eta h$ **then**
        let $b_v = (d_v)^{k-1}$
    **else**
        let $b_v = 0$ and $s_v = 3\eta\, d_v$
compute $\prec$ such that $u \prec v \iff (s_u > s_v) \vee ((s_u = s_v) \wedge (u > v))$
**for** each $v : d_v \leq k/\eta$ **do**
    **if** $B(v) \neq \emptyset$ **then**
        compute $d(v|G(v))$ and let $b_v = d(v|G(v))^{k-1}$
    **else**
        let $b_v = 0$
**return** $\prec$ and $\{b_v\}_{v \in V}$

Lemma 28. *With high probability Apx-DD$(G, \beta)$ returns an $(\alpha, \beta)$-DD order for $G$ with $\alpha = \beta^{\frac{1}{k-1}} \frac{1}{6k^3}$.*

Lemma 29. *Apx-DD$(G, \beta)$ can be implemented to run in time $O(\beta^{-\frac{2}{k-1}} k^6\, n \log n)$.*

To carry out the proofs, we need some notation and a few observations about Apx-DD$(G, \beta)$. We denote by:

- $t = 1, \ldots, n$ the generic round of the first loop
- $t_v$ the round where $v$ is processed
- $s_t(v)$ the value of $s_v$ at the beginning of round $t$; note that $s_t(v) \geq s_{t+1}(v)$, that $s_{t_v}(v) = d_v$, and that $s_{n+1}(v) = s_{t_v+1}(v) \in \{d_v, 3\eta d_v\}$
- $\prec_t$ the order defined by $u \succ_t v$ if and only if $(s_t(u) < s_t(v)) \vee ((s_t(u) = s_t(v)) \wedge (u < v))$
- $G_t(v) = G[\{z : z \succeq_t v\}]$ the subgraph induced by $v$ and the vertices after it at time $t$
- $d(u|G_t(v))$ the degree of $u$ in $G_t(v)$; obviously, $d(u|G_t(v)) \leq d_u$ for all $t$
- $X_j = \mathbb{I}\{x_j \succ v\}$ and $X = \sum_{j=1}^{h} X_j$, in a generic round (hence, $\prec = \prec_t$ for some $t$)

We denote the returned order by $\prec_n$ (formally it would be $\prec_{n+1}$, but clearly this equals $\prec_n$), and by $G_n(\cdot)$ the subgraphs induced in $G$ under such an order. By $b_v$, we always mean the value of $b_v$ at return time, unless otherwise specified.

Observation 30. *For any $t$, if $u \succ_t v$ then $s_t(u) \le s_t(v)$.*

Proof. By definition $u \succ_t v$ if and only if $(s_t(u) < s_t(v)) \vee ((s_t(u) = s_t(v)) \wedge (u < v))$. Therefore, in particular, $s_t(u) \le s_t(v)$. □

Observation 31. *If $u \prec_{t_u} v$, then $G_n(v) \subseteq G_{t_u}(u)$ and $d(u|G_n(v)) \le d(u|G_{t_u}(u))$.*

Proof. By definition, $G_n(v) \subseteq G_{t_u}(u)$ means $\{z : z \succeq_n v\} \subseteq \{z : z \succeq_{t_u} u\}$, which is equivalent to $\{z : z \prec_{t_u} u\} \subseteq \{z : z \prec_n v\}$. Consider then any $z : z \prec_{t_u} u$. This implies $z \prec_{t_u} v$ (since $u \prec_{t_u} v$) and $t_z < t_u$ (since $t_z > t_u$ would imply $z \succ_{t_u} u$). But $z$ cannot be moved past $v$ in any round $t' > t$. Thus, $z \prec_n v$. Therefore, $\{z : z \prec_{t_u} u\} \subseteq \{z : z \prec_n v\}$, as desired. The second claim follows by the monotonicity of $d(u|\cdot)$. □

Observation 32. *For all $v$ and all $t \ge t_v$, we have $d(v|G_{t_v}(v)) \ge d(v|G_t(v))$, with equality if $b_v > 0$.*

Proof. Consider any $z : z \prec_{t_v} v$. Note that $t_z < t_v$, hence $z \notin G_{t_v}(v)$ by definition of $G_{t_v}(v)$. Moreover, $z$ will never be moved past $v$ in any round $t \ge t_v$, so $z \notin G_t(v)$ as well. Therefore, $G_{t_v}(v) \supseteq G_t(v)$ for all $t \ge t_v$. Now the claim follows by monotonicity of $d(v|\cdot)$, and by noting that if $b_v > 0$, then $v$ is not moved at round $t_v$ and thus $G_t(v) = G_{t_v}(v)$ for all $t \ge t_v$. □

Observation 33. *In any round, conditioned on past events, w.h.p. $\left|X - \mathbb{E}X\right| \le \eta h$.*

Proof. Consider round $t_v$. Conditioned on past events, the $X_j$ are independent binary random variables. Therefore, by Hoeffding's inequality:

$$\mathbb{P}(|X - \mathbb{E}X| > h\eta) < 2e^{-2h\eta^2} = e^{-\Theta(\log n)} = n^{-\Theta(1)}, \tag{58}$$

where $h = \Theta(\eta^{-2} \log n)$ and thus the $\Theta(1)$ at the exponent can be chosen arbitrarily large. □

Observation 34. *With high probability, $d(v|G_t(v)) \le s_{n+1}(v)$ for every $v$ anytime in any round $t$.*

Proof. If $s_{n+1}(v) = d_v$, then clearly $s_{n+1}(v) \ge d(v|G_t(v))$. Suppose instead that $s_{n+1}(v) = 3\eta d_v$. By Observation 32, $d(v|G_t(v)) \le d(v|G_{t_v}(v))$. So, we only need to show that with high probability $d(v|G_{t_v}(v)) \le 3\eta d_v$. Consider the random variable $X = \sum_{j=1}^{h} X_j$ at round $t_v$, and note that $\mathbb{E}X_j = \frac{d(v|G_{t_v}(v))}{d_v}$ for all $j$. Therefore, if $d(v|G_{t_v}(v)) > 3\eta d_v$, then $\mathbb{E}X > 3\eta h$. Now, the algorithm updates $s_v$ only if $X < 2\eta h$. This implies the event $X < \mathbb{E}X - \eta h$, which by Observation 33 fails with high probability. Thus, with high probability $d(v|G_{t_v}(v)) \le 3\eta d_v$. □

Observation 35. *If round $t_v$ of the first loop sets $b_v > 0$, then w.h.p. $d(v|G_{t_v}(v)) \ge \eta d_v$, else w.h.p. $d(v|G_{t_v}(v)) \le 3\eta d_v$. If the second loop sets $b_v > 0$, then $d(v|G_{t_v}(v)) \ge \frac{\eta}{k} d_v$ deterministically.*

Proof. The first claim has the same proof of Observation 34: If $d(v|G_{t_v}(v)) < \eta d_v$, then $\mathbb{E}X < \eta h$, so $b_v > 0$ implies $X \ge 2\eta h$ and thus $X > \mathbb{E}X + \eta h$. Similarly, if $d(v|G_{t_v}(v)) > 3\eta d_v$, then $\mathbb{E}X > 3\eta h$, so letting $b_v = 0$ implies $X < 2\eta h$, which means $X < \mathbb{E}X - \eta h$. Both events fail with high probability by Observation 33. For the second claim, note that the second loop sets $b_v > 0$ only if $d_v \le \frac{k}{\eta}$, which implies $\frac{\eta}{k} d_v \le 1$, and if $B(v) \ne \emptyset$, which implies $d(v|G_n(v)) \ge 1$. Thus, $d(v|G_n(v)) \ge \frac{\eta}{k} d_v$. Observation 32 gives $d(v|G_{t_v}(v)) \ge d(v|G_n(v))$, concluding the proof. □

Observation 36. *With high probability, for all $v$, for all $u \succ_n v$, we have $d(u|G_n(v)) \le s_{n+1}(v)$.*

Proof. Consider the beginning of round $t_u$. Suppose that $v \prec_{t_u} u$, which implies $t_v < t_u$. Then:

$$\begin{aligned} s_{n+1}(v) &= s_{t_v+1}(v) && && (59)\\ &= s_{t_u}(v) && \text{since } t_v < t_u && (60)\\ &\geq s_{t_u}(u) && \text{Observation 30, using } u \succ_{t_u} v && (61)\\ &= d_u && \text{by construction} && (62)\\ &\geq d(u|G_n(v).) && && (63) \end{aligned}$$

Suppose instead $u \prec_{t_u} v$. Then, with high probability:

$$\begin{aligned} s_{n+1}(v) &\geq s_{n+1}(u) && \text{Observation 30, using } u \succ_n v && (64)\\ &\geq d(u|G_{t_u}(u)) && \text{Observation 34, with } t = t_u && (65)\\ &\geq d(u|G_n(v)) && \text{Observation 31.} && (66) \end{aligned}$$

In any case, $d(u|G_n(v)) \leq s_{n+1}(v)$. □

Proof of Lemma 28. For technical reasons, we prove the four properties of $(\prec, \boldsymbol{b})$ (see Definition 27) in a different order. Moreover, we substitute $\alpha = \frac{\eta}{k}$. This yields the four properties:

(1) if $v \prec u$, then $d_v \geq 3\eta\, d_u$
(2) if $b_v > 0$, then $d(v|G(v)) \geq \frac{\eta}{k} d_v \geq \frac{\eta}{k} \cdot d(u|G(v))$ for all $u \succ v$
(3) if $b_v > 0$, then $k^{-O(k)}\beta \leq \frac{b_v}{|B(v)|} \leq \frac{k^{O(k)}}{\beta}$
(1) $\sum_{v:b_v>0} |B(v)| \geq (1-\beta) \sum_{v \in V} |B(v)|$.

*Proof of (1).* Simply note that $d_v \geq s_{n+1}(v) \geq s_{n+1}(u) \geq 3\eta d_u$, where the middle inequality holds by Observation 30, since $u \succ_n v$.

*Proof of (2).* Consider any $u \succ_n v$ with $b_v > 0$. Then, with high probability:

$$\begin{aligned} d(v|G_n(v)) &= d(v|G_{t_v}(v)) && \text{Observation 32, using } b_v > 0 \text{ and } t = n && (67)\\ &\geq \frac{\eta}{k} d_v && \text{Observation 35, using } b_v > 0 && (68)\\ &= \frac{\eta}{k} s_{n+1}(v) && \text{by the algorithm, since } b_v > 0 && (69)\\ &\geq \frac{\eta}{k} d(u|G_n(v)) && \text{Observation 36, since } u \succ_n v. && (70) \end{aligned}$$

*Proof of (3).* First, we show that if $b_v > 0$, then w.h.p. $|B(v)| \geq 1$. If $v$ is processed by the second loop, then $b_v > 0$ if and only if $|B(v)| \geq 1$. Otherwise, we know $d_v > \frac{k}{\eta}$, and w.h.p.:

$$\begin{aligned} d(v|G_n(v)) &= d(v|G_{t_v}(v)) && \text{Observation 32, using } b_v > 0 && (71)\\ &\geq \eta d_v && \text{Observation 35, using } b_v > 0 && (72)\\ &> k && \text{as } d_v > \frac{k}{\eta}. && (73) \end{aligned}$$

So, w.h.p. $d(v|G_n(v)) > k$, in which case $G_n(v)$ contains a $k$-star centered in $v$, implying $|B(v)| \geq 1$.

Thus, we continue under the assumption $|B(v)| \geq 1$. To ease the notation, define $d^*_v = d(v|G_n(v))$ and $\Delta^*_v = \max_{u \in G(v)} d(u|G_n(v))$. Lemma 45 applied to $G_n(v)$ yields:

$$k^{-O(k)} \left(d^*_v\right)^{k-1} \leq |B(v)| \leq k^{O(k)} \left(\Delta^*_v\right)^{k-1}. \tag{74}$$

We now show that w.h.p.:

$$\beta k^{-O(k)}(\Delta_v^*)^{k-1} \le b_v \le \frac{1}{\beta}k^{O(k)}(d_v^*)^{k-1}, \tag{75}$$

which implies our claim. For the upper bound, note that by construction $b_v \le (d_v)^{k-1}$ and that, by point (2) of this lemma, w.h.p. $d_v \le \frac{k}{\eta}d_v^*$. Substituting $\eta$, we obtain:

$$b_v \le (d_v)^{k-1} \le (d_v^*)^{k-1}\left(\frac{k}{\eta}\right)^{k-1} = \frac{1}{\beta}k^{O(k)}(d_v^*)^{k-1}. \tag{76}$$

For the lower bound, note that, since $b_v > 0$, then $b_v \ge (d_v^*)^{k-1}$. Indeed, if $b_v > 0$, then either $b_v = (d_v)^{k-1} \ge (d_v^*)^{k-1}$ from the first loop, or $b_v = (d_v^*)^{k-1}$ from the second loop (since the value $d(v|G(v))$ in the second loop equals $d(v|G_n(v))$, that is, $d_v^*$). Now, point (2) of this lemma gives $d_v^* \ge \frac{\eta}{k}\cdot d(u|G_n(v))$ for all $u \succ_n v$. Thus, $\Delta_v^* = \max_{u\in G(v)} d(u|G_n(v)) \le \frac{k}{\eta}d_v^*$. Therefore:

$$(\Delta_v^*)^{k-1} \le \left(\frac{k}{\eta}d_v^*\right)^{k-1} = \beta k^{O(k)}(d_v^*)^{k-1} \le \beta k^{O(k)} b_v. \tag{77}$$

*Proof of (4).* We prove the equivalent claim:

$$\sum_{v:b_v=0}|B(v)| \le \beta\sum_{v\in V}|B(v)|. \tag{78}$$

Consider any $v$ with $b_v = 0$ and $|B(v)| > 0$. These are the only vertices contributing to the left-hand summation. First, we note that $d_v > \frac{k}{\eta}$. Indeed, if $|B(v)| > 0$ and $d_v \le \frac{k}{\eta}$, then the second loop of Apx-DD processes $v$ and sets $b_v = (d(v|G_n(v)))^{k-1}$, which is positive, since $|B(v)| > 0$ implies $d(v|G(v)) > 0$. Thus, we can assume that $b_v = 0$, $|B(v)| \ge 1$, $d_v > \frac{k}{\eta}$, and $v$ is not processed in the second loop. Since $d_v > \frac{k}{\eta} > k-1$, then $G$ contains at least $\binom{d_v}{k-1} \ge 1$ stars centered in $v$. Each such star contributes 1 to $\sum_{v\in V}|B(v)|$. Since $\frac{(d_v)^{k-1}}{(k-1)^{k-1}} \le \binom{d_v}{k-1}$, whenever $\binom{d_v}{k-1} \ge 1$, we obtain:

$$\sum_{v:b_v=0}\frac{(d_v)^{k-1}}{(k-1)^{k-1}} \le \sum_{v:b_v=0}\binom{d_v}{k-1} \le k\sum_{v:b_v=0}|B(v)| \le k\sum_{v}|B(v)|, \tag{79}$$

where the factor $k$ arises from each star being counted up to $k$ times by the left-hand side (once for each vertex in the star).

However, by Observation 34 and Observation 36, w.h.p. $d(v|G_n(v)) \le s_{n+1}(v)$ and $d(u|G_n(v)) \le s_{n+1}(v)$ for all $u \succ_n v$. Hence, the maximum degree of $G_v$ is w.h.p. at most $s_{n+1}(v)$. But $s_{n+1}(v) = 3\eta d_v$, since $b_v = 0$ is set in the first loop. Thus, the maximum degree of $G_v$ is at most $3\eta(d_v)$. By Lemma 45, then,

$$\sum_{v:b_v=0}|B(v)| \le \sum_{v:b_v=0}(k-1)!(3\eta\, d_v)^{k-1}. \tag{80}$$

By coupling Equations (79) and (80) and substituting $\eta = \beta^{\frac{1}{k-1}}\frac{1}{6k^2}$, we obtain:

$$\frac{\sum_{v:b_v=0}|B(v)|}{\sum_v|B(v)|} \le \frac{\sum_{v:b_v=0}(k-1)!(3\eta\, d_v)^{k-1}}{\frac{1}{k}\sum_{v:b_v=0}\frac{(d_v)^{k-1}}{(k-1)^{k-1}}} \qquad \text{by (79) and (80)} \tag{81}$$

$$= \frac{(k-1)!(3\eta)^{k-1}\sum_{v:b_v=0}(d_v)^{k-1}}{\frac{1}{k(k-1)^{k-1}}\sum_{v:b_v=0}(d_v)^{k-1}} \tag{82}$$

$$< (3\eta)^{k-1} k(k-1)^{2(k-1)} \tag{83}$$

$$= \left(3\beta^{\frac{1}{k-1}} \frac{1}{6k^2}\right)^{k-1} k(k-1)^{2(k-1)} \tag{84}$$

$$= \beta \frac{k(k-1)^{2(k-1)}}{2^{k-1} k^{2(k-1)}}, \tag{85}$$

which for all $k \geq 2$ is bounded from above by $\beta$. The proof is complete. □

Proof of Lemma 29. The initialization of Apx-DD is dominated by sorting $V$ in order of degree, which takes time $O(n)$ via bucket sort. In the first loop, at each iteration, we draw $O(h) = O(\eta^{-2} \log n)$ samples, each of which takes time $O(1)$ via neighbor queries. Evaluating $x_j \succ v$ takes time $O(1)$, and computing $X$ takes time $O(h) = O(\eta^{-2} \log n)$. Updating $s_v$ takes time $O(1)$. Thus, each iteration of the first loop takes time $O(\eta^{-2} \log n)$. Computing $\prec$ takes $O(n \log n)$ as every comparison takes time $O(1)$.

Consider now the second loop; we claim that each iteration takes time $O(\eta^{-2} k^2 \log k)$. To see this, let us describe the BFS in more detail. We start by pushing $v$ in the queue, and we maintain the invariant that the queue holds only vertices of $G(v)$. To this end, when we pop a generic vertex $u$, we examine every edge $\{u, z\} \in E(G)$ and push $z$ only if $z \succ v$ and $z$ was not pushed before. Note that checking whether $z \succ v$ takes time $O(1)$. Now, we bound the number of neighbors $z$ of $u$ that are examined. First, this number is obviously at most $d_u$. Recall that $3\eta d_u \leq s_{n+1}(u)$ by construction of the algorithm. Moreover, since $u \in G_n(v)$, then $u \succ_{n+1} v$, which by Observation 30 implies $s_{n+1}(u) \leq s_{n+1}(v) \leq d_v \leq \frac{k}{\eta}$. Therefore, $d_u \leq \frac{k}{3\eta^2}$. Hence, the number of neighbors $z$ of $u$ that are examined is at most $\frac{k}{3\eta^2}$. Since we push at most $k$ vertices before stopping, the total number of vertices/edges examined by each BFS is in $O(\eta^{-2} k^2)$. To store the set of pushed vertices, we use a dictionary with logarithmic insertion and lookup time. Hence, each BFS will take time $O(\eta^{-2} k^2 \log k)$. Finally, computing $d(v|G(v))$ also takes time $d_v \leq \frac{k}{3\eta^2}$. Thus, each iteration of the second loop runs in time $O(\eta^{-2} k^2 \log k)$.

As each loop makes at most $n$ iterations, the total running time of Apx-DD$(G, \beta)$ is:

$$O(n \log n) + O(n\eta^{-2} \log n) + O(n\eta^{-2} k^2 \log k) = O(\eta^{-2} k^2 n \log n). \tag{86}$$

Replacing $\eta = O(\beta^{\frac{1}{k-1}} k^{-2})$ shows that the running time is in $O(\beta^{-\frac{2}{k-1}} k^6 n \log n)$, as claimed. □

We can conclude the preprocessing phase of Apx-Ugs. We set $\beta = \frac{\varepsilon}{2}$, and run $(\prec, \boldsymbol{b}) =$ Apx-DD $(G, \beta)$. Then, for all $v$, we let $p(v) = \frac{b_v}{\sum_u b_u}$; we also set a few other variables. The running time is dominated by Apx-DD$(G, \beta)$, which by Lemma 29 takes time $O(\varepsilon^{-\frac{2}{k-1}} k^6 n \log n)$. This proves the preprocessing time bound of Theorem 6 and completes the description of the preprocessing phase.

### 7.2 The Sampling Phase: A Coupling of Algorithms

Recall that, by Lemma 28, with high probability the preprocessing phase yields an $(\alpha, \frac{\varepsilon}{2})$-DD order $(\prec, \boldsymbol{b})$ for $G$, with $\alpha = \Theta(\varepsilon^{\frac{1}{k-1}} k^{-3})$. From now on, we assume this holds. Then, by Definition 27, $\cup_{v:b_v>0} B(v)$ contains a fraction $1 - \frac{\varepsilon}{2}$ of all graphlets. Hence, our goal becomes sampling $\frac{\varepsilon}{2}$-uniformly from $\cup_{v:b_v>0} B(v)$. By the triangle inequality, this will give an $\varepsilon$-uniform distribution over $\mathcal{V}_k$. To achieve $\frac{\varepsilon}{2}$-uniformity over $\cup_{v:b_v>0} B(v)$, we modify Ugs step-by-step. To begin, we consider what would happen if we sorted $G$ according to $\prec$ and ran the sampling phase of Ugs using the bucket size estimates $\boldsymbol{b}$. We show that, by mildly reducing the acceptance probability, we could make the output graphlet distribution uniform over $\cup_{v:b_v>0} B(v)$. The resulting algorithm,

UGS-COMPARE$(G, \varepsilon)$, is given below. Note that UGS-COMPARE$(G, \varepsilon)$ is just for analysis purposes; we use it as a comparison term to establish the $\varepsilon$-uniformity of our algorithm.

**ALGORITHM 6:** UGS-COMPARE$(G, \varepsilon)$

```
1: let C_1 = a large enough universal constant
2: let β = ε/2
3: (≺, b) = APX-DD(G, α, β)
4: let Z = Σ_{v∈V} b_v, and for each v ∈ V let p(v) = b_v/Z
5: sort the adjacency lists of G according to ≺
6:
7: function SAMPLE( )
8:     while true do
9:         draw v from the distribution p
10:        S = RAND-GROW(G, v)
11:        p_v(S) = PROB(G, S)
12:        with probability 1/(p(v) p_v(S)) · β/Z · k^{-C_1 k} return S
```

LEMMA 37. *In UGS-COMPARE$(G, \varepsilon)$, suppose APX-DD$(G, \alpha, \beta)$ succeeds (Lemma 28), and let* $p_{acc}(v, S) = \frac{1}{p(v)\,p(S)} \frac{\beta}{Z} k^{-C_1 k}$ *be the expression computed by SAMPLE( ) at line 12. Then* $p_{acc}(v, S) \in [\varepsilon^2 k^{-O(k)}, 1]$, *and moreover, the distribution of the graphlets returned by SAMPLE( ) is uniform over* $\cup_{v: b_v > 0} B(v)$.

PROOF. Rewrite:

$$p_{\text{acc}}(v, S) = \frac{1}{p(v)\,p(S)} \frac{\beta}{Z} k^{-C_1 k} = \frac{1}{\frac{b_v}{Z} p(S)} \frac{\beta}{Z} k^{-C_1 k} = \frac{\beta k^{-C_1 k}}{b_v\, p(S)}. \tag{87}$$

If APX-DD$(G, \alpha, \beta)$ succeeds, then $(\prec, \boldsymbol{b})$ is an $(\alpha, \beta)$-order with $\alpha = \beta \frac{1}{k-1} \frac{1}{6k^3}$; we will show that, if this is the case, then the last expression in Equation (87) is in $[\varepsilon^2 k^{-O(k)}, 1]$. This implies that $p_{\text{acc}}(v, S)$ is a well-defined probability; the uniformity of the returned graphlets then follows immediately from the fact that the sampling routine is the one of UGS.

*Upper bound.* We bound $b_v\, p(S)$ from below. First, since $v$ was chosen, then $p_v > 0$ and thus $b_v > 0$, in which case by construction APX-DD$(G, \alpha, \beta)$ sets:

$$b_v = \min\left(d_v, d(v|G(v))\right)^{k-1} \geq d(v|G(v))^{k-1}. \tag{88}$$

Now, we adapt the lower bound on $p(S)$ of Lemma 25 by modifying Equation (54). By Definition 27, $b_v > 0$ implies $|B(v)| > 0$, so the hypotheses of Lemma 25 are satisfied. Since $G[S]$ is connected, then at least one sequence $v, u_2, \ldots, u_k$ exists such that $c_i \geq 1$ for all $i = 1, \ldots, k-1$, and $p(S)$ is at least the probability that RAND-GROW follows that sequence. By Lemma 28, all $u > v$ satisfy $d(u|G(v)) \leq \frac{1}{\alpha} \cdot d(v|G(v))$; this holds for $u = v$ as well, since $\alpha \leq 1$. It follows that $c_i \leq \frac{i}{\alpha}\, d(v|G(v))$ for all $i = 1, \ldots, k-1$. Hence, for all $i = 1, \ldots, k-1$, Equation (54) becomes:

$$\mathbb{P}(u_{i+1} = u') \geq \frac{\alpha}{i\, d(v|G(v))}. \tag{89}$$

Thus, the probability that the algorithm follows $v, u_2, \ldots, u_k$ is at least

$$p(S) \geq \prod_{i=1}^{k-1} \frac{\alpha}{i\, d(v|G(v))} = \frac{\alpha^{k-1}}{(k-1)!\, d(v|G(v))^{k-1}}. \tag{90}$$

Combining Equations (88) and (90), we conclude that for some absolute constant $C_2$:

$$b_v \cdot p(S) \geq d(v|G(v))^{k-1} \cdot \frac{\alpha^{k-1}}{(k-1)!\, d(v|G(v))^{k-1}} \geq \frac{\alpha^{k-1}}{(k-1)!}, \tag{91}$$

and therefore

$$\frac{\beta k^{-C_1 k}}{b_v\, p(S)} \leq \frac{\beta k^{-C_1 k}}{\frac{\alpha^{k-1}}{(k-1)!}} \leq \frac{\beta k^{-(C_1-C_2)k}}{\alpha^{k-1}}. \tag{92}$$

Since $\alpha = \beta^{\frac{1}{k-1}} \frac{1}{6k^3}$, we have:

$$\frac{\beta k^{-(C_1-C_2)k}}{\alpha^{k-1}} = \frac{\beta k^{-(C_1-C_2)k} (6k^3)^{k-1}}{\beta} \leq k^{-(C_1-C_3)k} \tag{93}$$

for some constant $C_3$. Choosing $C_1 \geq C_3$, the acceptance probability is in $[0, 1]$.

*Lower bound.* We bound $b_v\, p(S)$ from above. On the one hand, note that the upper bound on $p(S)$ of Lemma 25 applies even for an $(\alpha, \beta)$-order. Indeed, that bound is based on the lower bound of Lemma 24 whose proof uses only $d(v|G(v))$ but not $d(u|G(v))$ for any $u \succ v$. Thus,

$$p(S) \leq \frac{k^{C_4 k}}{d(v|G(v))^{k-1}} \tag{94}$$

for some constant $C_4$. On the other hand, $b_v \leq d_v^{k-1}$ by construction of Apx-DD. Moreover, since $b_v > 0$, by Definition 27, we have $d_v \leq \frac{1}{\alpha} d(v|G(v))$. Therefore,

$$b_v \leq \frac{1}{\alpha^{k-1}} d(v|G(v))^{k-1}. \tag{95}$$

We conclude that:

$$b_v \cdot p(S) \leq \frac{1}{\alpha^{k-1}} d(v|G(v))^{k-1} \frac{k^{C_4 k}}{d(v|G(v))^{k-1}} = \frac{k^{C_4 k}}{\alpha^{k-1}}. \tag{96}$$

Since $\alpha = \beta^{\frac{1}{k-1}} \frac{1}{6k^3}$, we have:

$$b_v \cdot p(S) \leq \frac{k^{C_4 k} (6k^3)^{k-1}}{\beta} \leq \frac{k^{C_5 k}}{\beta} \tag{97}$$

for some constant $C_5$. Hence,

$$\frac{\beta k^{-C_1 k}}{b_v\, p(S)} \geq \beta k^{-C_1 k} \frac{\beta}{k^{C_5 k}} = \beta^2 c^{-(C_5+C_1)k}. \tag{98}$$

Replacing $\beta = \frac{\varepsilon}{2}$ shows that the acceptance probability is at least $\varepsilon^2 k^{-O(k)}$, as claimed. □

Ugs-Compare is now our baseline. Our goal is building an algorithm whose output distribution is $\frac{\varepsilon}{2}$-close to that of Ugs-Compare, *without using the sorted adjacency lists*. To this end, we will carefully re-design the routines of Ugs and use several coupling arguments. In what follows, we assume that $V(G)$ is sorted by $\prec$, and we fix some $v \in V(G)$ with $b_v > 0$.

**ALGORITHM 7:** EstimateCuts$(G, v, U, \alpha, \beta, \delta)$

let $\ell = \frac{1}{k\delta\alpha^2}$, $h = \Theta(\ell^2 \log \frac{k}{\beta})$
let $\widehat{c}(U) = 0$
**for** each $u \in U$ **do**
    sample $h$ neighbors $x_1, \ldots, x_h$ of $u$ i.i.d. u.a.r.
    let $X = \sum_{j=1}^{h} \mathbb{I}\{x_j \succ v \wedge x_j \notin U\}$
    **if** $X \geq \ell$ **then** let $\widehat{c}(u) = \frac{d_u}{h} X$ **else** let $\widehat{c}(u) = 0$
**return** $\{\widehat{c}(u)\}_{u \in S}$

*7.2.1 Approximating the Cuts.* First, we show how to estimate efficiently the size of the cuts encountered by the random growing process. We will use these estimates to approximate the random growing process itself, as well as the computation of $p_v(g)$. The quality of our estimates and the cost of computing them are both based on the properties of $(\alpha, \beta)$-DD orders.

Lemma 38. *EstimateCuts$(G, v, U, \alpha, \beta)$ runs in time $O(|U|^2 \frac{1}{k\delta^2\alpha^4} \log \frac{1}{\beta})$.*

Proof. At each iteration EstimateCuts$(G, v, U, \alpha, \beta)$ draws $h = O(\frac{1}{k^2\delta^2\alpha^4}) \log \frac{k}{\beta}$ samples, which is in $O(\frac{1}{k\delta^2\alpha^4} \log \frac{1}{\beta})$ as $\log \frac{k}{\beta} = O(k \log \frac{1}{\beta})$. For each sample, computing $\mathbb{I}\{x_j \succ v \wedge x_j \notin U\}$ takes time $O(|U|)$ via edge queries. Summing over all iterations gives a bound of $O(|U|) \cdot O(\frac{1}{k\delta^2\alpha^4} \log \frac{1}{\beta}) \cdot O(|U|) = O(|U|^2 \frac{1}{k\delta^2\alpha^4} \log \frac{1}{\beta})$. □

Lemma 39. *Let $G[U]$ be a connected subgraph of $G(v)$ on $i < k$ vertices containing $v$. With probability $1 - \text{poly}\,\frac{\beta}{k}$, the output of EstimateCuts$(G, v, U, \alpha, \beta, \delta)$ satisfies:*

$$|\widehat{c}(u) - c(u)| \leq \delta\, d(v|G(v)) \quad \forall u \in U, \tag{99}$$

*where $c(u) = |\operatorname{Cut}(u, G(v) \setminus U)|$. In this case, then $|\widehat{c}(U) - c(U)| \leq |U|\delta k c(U)$ too, where $\widehat{c}(U) = \sum_{u \in U} \widehat{c}(u)$ and $c(U) = \sum_{u \in U} c(u) = |\operatorname{Cut}(U, G(v) \setminus U)|$.*

Proof. Fix any $u \in U$ and consider the iteration where the edges of $u$ are sampled. For each $j \in [h]$ let $X_{u,j} = \mathbb{I}\left\{x_j \succ v \wedge x_j \notin U\right\}$. Clearly, $\mathbb{E}[X_{u,j}] = \frac{hc(u)}{d_u}$. Let $X_u = \sum_{j=1}^{h} X_{u,j}$; this is the value of $X$ tested by the algorithm at $u$'s round. To begin, we note that:

$$\mathbb{E}\left[\frac{d_u}{h} X_u\right] = \frac{d_u}{h} \mathbb{E}\left[\sum_{j=1}^{h} X_{u,j}\right] = c(u). \tag{100}$$

Define $\gamma(u) = \frac{d_u}{h} X_u$. Clearly,

$$\mathbb{P}(|\gamma(u) - c(u)| > \delta d(v|G(v))) = \mathbb{P}\left(|X_u - \mathbb{E}X_u| > h\,\delta\, \frac{d(v|G(v))}{d_u}\right). \tag{101}$$

Note that the algorithm can set $\widehat{c}(u) = \gamma(u)$ or $\widehat{c}(u) = 0$. First, we show that $\gamma(u)$ is concentrated around $c(u)$. Then, we deal with the value of $\widehat{c}(u)$ set by the algorithm.

Clearly, $X$ is the sum of $h$ i.i.d. indicator random variables. By Hoeffding's inequality, for any $\delta > 0$, we have $\mathbb{P}(|X_u - \mathbb{E}X_u| > t) < 2e^{-2\frac{t^2}{h}}$. With $t = \frac{\delta\, d(v|G(v))\, h}{d_u}$, we obtain:

$$\mathbb{P}\left(|X_u - \mathbb{E}X_u| > h\,\delta\, \frac{d(v|G(v))}{d_u}\right) < 2\exp\left(-2h\,\delta^2 \left(\frac{d(v|G(v))}{d_u}\right)^2\right). \tag{102}$$

Now, as $b_v > 0$ and $u > v$, by Definition 27, we have $d(v|G(v)) \geq \alpha d_v$ and $d_v \geq 3k\alpha d_u$. Hence, $d_u < \frac{1}{3k\alpha^2} d(v|G(v))$, so $\frac{d(v|G(v))}{d_u} > 3k\alpha^2$. Therefore:

$$h\,\delta^2 \left(\frac{d(v|G(v))}{d_u}\right)^2 \geq 9h\,\delta^2 k^2 \alpha^4. \tag{103}$$

However, note that $h = \Theta(\frac{1}{\delta^2 k^2 \alpha^4} \log \frac{k}{\beta})$. Thus, $\mathbb{P}\Big(|\gamma(u) - c(u)| > \delta\, d(v|G(v))\Big) \leq \text{poly}\,\frac{\beta}{k}$.

Now, the algorithm fails if it either sets $\widehat{c}(u) = \gamma(u)$ and $|\gamma(u) - c(u)| > \delta\, d(v|G(v))$ or if it sets $\widehat{c}(u) = 0$ and $|0 - c(u)| > \delta\, d(v|G(v))$. The probability of the first event is at most the probability that $|\gamma(u) - c(u)| > \delta\, d(v|G(v))$, which is poly $\frac{\beta}{k}$ as shown above. So, we must bound the probability that $\widehat{c}(u) = 0$ and $|0 - c(u)| > \delta d(v|G(v))$; this second condition is just $c(u) > \delta\, d(v|G(v))$. Recalling that $\mathbb{E}X_u = h\frac{c(u)}{d_u}$ and that $\frac{d(v|G(v))}{d_u} \geq 3k\alpha^2$ and $\delta = \frac{\beta}{4k^2}$, we obtain:

$$\mathbb{E}X_u = h\frac{c(u)}{d_u} > h\frac{\delta\, d(v|G(v))}{d_u} \geq 3kh\alpha^2\delta. \tag{104}$$

Note that $h\alpha^2\delta = \frac{h}{\ell} \in \Omega(\sqrt{h \log \frac{k}{\beta}})$. Also note that $\ell = o(\frac{h}{\ell})$. This implies:

$$\mathbb{E}X_u - \ell = \Omega\left(\sqrt{h \log \frac{k}{\beta}}\right). \tag{105}$$

Now, $\widehat{c}(u) = 0$ by construction of the algorithm implies $X_u < \ell$, which can be rewritten as $X_u < \mathbb{E}X_u - t$ with $t = \mathbb{E}X_u - \ell = \Omega(\sqrt{h \log \frac{k}{\beta}})$. Since $X_u$ is the sum of $h$ i.i.d. indicator random variables, Hoeffding's inequality gives:

$$\mathbb{P}\left(X_u - \mathbb{E}X_u < t\right) < e^{-\frac{2t^2}{h}} = \exp\left(-\frac{\Omega\left(\sqrt{h \log \frac{k}{\beta}}\right)^2}{h}\right) = \exp\left(-\Omega\left(\frac{k}{\beta}\right)\right). \tag{106}$$

Hence, this event has probability at most poly $\frac{\beta}{k}$, too. We conclude that $|\widehat{c}(u) - c(u)| \leq \delta\, d(v|G(v))$ with probability at least $1 - \text{poly}\,\frac{\beta}{k}$. By a union bound over $u \in S$, this proves the claim for the $\widehat{c}(u)$.

For $\widehat{c}(U)$, note that:

$$|\widehat{c}(U) - c(U)| = \left|\sum_{u\in U} \widehat{c}(u) - \sum_{u\in U} c(u)\right| \leq \sum_{u\in U} |\widehat{c}(u) - c(u)| \leq |U|\,\delta\, d(v|G(v)). \tag{107}$$

By Lemma 24 $d(v|G(v)) \leq kc(U)$, concluding the proof. □

*7.2.2 Approximating the Random Growing Process.* Using EstimateCuts we now run an approximate random growing process as follows: Start with $S_1 = \{v\}$, and at each step $i = 1, \ldots, k-1$, run EstimateCuts with $U = S_i$. This gives estimates of $|\,\text{Cut}(u, G(v) \setminus S_i)|$ for all $u \in S_i$. Using these estimates, sample a random edge near-uniformly from $\text{Cut}(S_i, G(v) \setminus S_i)$. The result is the following routine whose output distribution is close to Rand-Grow:

Lemma 40. *Let $p$ be the output distribution of Rand-Grow$(G, v)$ and $q$ the output distribution of Apx-Rand-Grow$(G, v, \alpha, \beta, \gamma)$. Then, $tvd(p, q) \leq \gamma + \text{poly}\,\frac{\beta}{k}$.*

Proof. We establish a coupling between the two algorithms. For $i = 1$, both algorithms set $S_1 = \{v\}$. Now suppose that both algorithms agree on $S_1, \ldots, S_i$ and they are about to choose $S_{i+1}$. We show that with probability $1 - \frac{\gamma}{Ck}$ they agree on the next edge drawn, and therefore on $S_{i+1}$.

**ALGORITHM 8:** Apx-Rand-Grow$(G, v, \alpha, \beta, \gamma)$

1: $S_1 = \{v\}$
2: **for** $i = 1, \ldots, k-1$ **do**
3: $(\widehat{c_i}(u))_{u \in S_i} =$ EstimateCuts$(G, v, S_i, \alpha, \beta, O(\gamma k^{-4}))$
4: $\widehat{c_i} = \sum_{u \in S_i} \widehat{c_i}(u)$
5: draw $u$ with probability $\frac{\widehat{c_i}(u)}{\widehat{c_i}}$ (if all $\widehat{c_i}(u) = 0$ then fail)
6: **repeat**
7: draw $u'$ u.a.r. from the adjacency list of $u$
8: **until** $u' \in G(v) \setminus S_i$
9: let $S_{i+1} = S_i \cup \{u'\}$
10: **return** $S_k$

Here, $C$ is a constant that we can make arbitrarily large by appropriately choosing the constants used along the algorithm and in EstimateCuts.

For $u \in S_i$ let $c_i(u) = |\operatorname{Cut}(u, G(v) \setminus S_i)|$, and let $c_i = \sum_{u \in S_i} c_i(u)$. For each $u \in S_i$, let $p_i(u) = \frac{c_i(u)}{c_i}$ and $q_i(u) = \frac{\widehat{c_i}(u)}{\widehat{c}(S_i)}$. So, $p_i(u)$ is the probability that Rand-Grow draws $u$ at line 5, and $q_i(u)$ the probability that Apx-Rand-Grow draws $u$ at line 5.

Now, if Apx-Rand-Grow and Rand-Grow both choose $u$, then we can couple them so they choose the same edge. This holds since both algorithms draw $u'$ uniformly from all neighbors of $u$ in $G(v) \setminus S_i$. So, the probability that the two algorithms choose a different edge is at most the probability that they choose $u$ differently, that is, by $\operatorname{tvd}(q_i, p_i) = \frac{1}{2}\|q_i - p_i\|_1$. Therefore:

$$\operatorname{tvd}(q_i, p_i) = \frac{1}{2} \sum_{u \in S_i} |q_i(u) - p_i(u)| = \sum_{u \in T} (q_i(u) - p_i(u)) = \sum_{u \in T} \left( \frac{\widehat{c_i}(u)}{\widehat{c}(S_i)} - \frac{c_i(u)}{c_i} \right), \tag{108}$$

where $T = \{u \in S_i : q_i(u) > p_i(u)\}$. Now, by Lemma 39, with probability $1 - \text{poly}\,\frac{\gamma}{k}$, we have $|\widehat{c}(u_i) - c(u_i)| \le \delta\, d(c|G(v))$ for all $u \in S_i$, and $|\widehat{c}(S_i) - c_i| \le k^2 \delta c_i$. In this case,

$$\frac{\widehat{c_i}(u)}{\widehat{c}(S_i)} - \frac{c_i(u)}{c_i} \le \frac{c_i(u) + \delta\, d(c|G(v))}{c_i(1 - k^2\delta)} - \frac{c_i(u)}{c_i} \tag{109}$$

$$= \frac{c_i(u) + \delta\, d(c|G(v)) - c_i(u)(1 - k^2\delta)}{c_i(1 - k^2\delta)} \tag{110}$$

$$= \frac{\delta\, d(c|G(v)) + k^2 \delta c_i(u)}{c_i(1 - k^2\delta)}. \tag{111}$$

Clearly, $c_i(u) \le c_i$, and by Lemma 24, $d(c|G(v)) \le k c_i$. Therefore:

$$\frac{\delta\, d(c|G(v)) + k^2 \delta c_i(u)}{c_i(1 - k^2\delta)} \le \frac{\delta\, k\, c_i + k^2 \delta c_i}{c_i(1 - k^2\delta)} \tag{112}$$

$$= \frac{\delta\, k\ + k^2\delta}{1 - k^2\delta} \tag{113}$$

$$\le k \cdot \frac{2\delta\, k^2}{1 - k^2\delta} \tag{114}$$

$$= \frac{2\delta\, k^3}{1 - k^2\delta}. \tag{115}$$

For any $\delta = O(\frac{\gamma}{k^4})$, this is in $O(\frac{\gamma}{k^2})$. Taking the sum over $u \in T$, we obtain $\operatorname{tvd}(q_i, p_i) = O(\frac{\gamma}{k})$.

Thus, the two algorithms will disagree on $S_{i+1}$ with probability at most poly $\frac{\beta}{k} + O(\frac{\gamma}{k})$. By a union bound on all $i$, the algorithms disagree on $S_k$ with probability poly $\frac{\beta}{k} + O(\gamma)$. The $O(\gamma)$ part can made smaller than $\gamma$ by choosing $\delta = O(\frac{\gamma}{k^4})$ small enough. □

Lemma 41. *Apx-Rand-Grow$(G, v, \alpha, \beta, \gamma)$ has expected running time $O(\frac{k^9}{\gamma^2\alpha^4} \log \frac{1}{\beta})$.*

Proof. At each iteration, since $|S_i| < k$, by Lemma 38, obtaining the cut estimates takes time $O(\frac{k}{\delta^2\alpha^4} \log \frac{1}{\beta})$. As $\delta = O(\frac{\gamma}{k^4})$, this gives a bound of $O(\frac{kk^8}{\gamma^2\alpha^4} \log \frac{1}{\beta}) = O(\frac{k^9}{\gamma^2\alpha^4} \log \frac{1}{\beta})$. We show that this dominates the expected time of the trials at lines 6–8 as well.

Let $T$ be the random variable giving the number of times lines 6–8 are executed. Let $\mathcal{E}_u$ be the event that $u \in S_i$ is chosen at line 5. Clearly, $\mathcal{E}_u$ implies $\widehat{c_i}(u) > 0$. Thus, $\mathbb{P}(\mathcal{E}_u) \le \mathbb{P}(\widehat{c_i}(u) > 0)$. Moreover, conditioned on $\mathcal{E}_u$, the algorithm returns after $\frac{d_u}{c_i(u)}$ trials in expectation. Therefore:

$$\mathbb{E}T = \sum_{u \in S_i} \mathbb{P}(\mathcal{E}_u)\, \mathbb{E}[T \mid \mathcal{E}_u] \le \sum_{u \in S_i} \mathbb{P}(\widehat{c_i}(u) > 0) \frac{d_u}{c_i(u)}. \tag{116}$$

Now recall EstimateCuts. By construction, $\widehat{c_i}(u) > 0$ implies $X_u \ge \ell$, where $\mathbb{E}X_u = \frac{c_i(u)}{d_u} h$. By Markov's inequality:

$$\mathbb{P}(\widehat{c_i}(u) > 0) = \mathbb{P}(X \ge \ell) \le \frac{\mathbb{E}X}{\ell} = \frac{c_i(u)}{d_u} \frac{h}{\ell} = \frac{c_i(u)}{d_u} \ell \log \frac{k}{\beta}. \tag{117}$$

Therefore:

$$\mathbb{E}T \le \sum_{u \in S_i} \frac{c_i(u)}{d_u} \ell \log \frac{k}{\beta} \cdot \frac{d_u}{c_i(u)} \le k\ell \log \frac{k}{\beta} = \frac{1}{\delta\alpha^2} \log \frac{k}{\beta} = \frac{k}{\delta\alpha^2} \log \frac{1}{\beta}. \tag{118}$$

Finally, note that each single trial takes time $O(k)$ via edge queries. The resulting time bound is in $O(\frac{k^2}{\delta\alpha^2} \log \frac{1}{\beta})$, which is dominated by the sampling running time (see above). □

*7.2.3 Approximating the Acceptance Probability.* Next, we compute an acceptance probability. For any $g \in B(v)$, let $q_v(g)$ be the probability that Apx-Rand-Grow returns $g$. If we could compute $q_v(g)$, then we would be done. However, computing $q_v(g)$ requires computing the exact sizes of the cuts, which takes time $\Omega(\Delta)$ in the worst case. Fortunately, we can show that a good approximation of $p_v(g)$, the probability that Rand-Grow returns $g$, is sufficient. By the properties of $(\alpha, \beta)$-DD orders, we can compute such an approximation efficiently.

**ALGORITHM 9:** Apx-Prob$(G, S = \{v, u_2, \ldots, u_k\}, \alpha, \beta, \rho)$

```
1:  p̂ = 0
2:  for each permutation σ = (σ_2, ..., σ_k) of u_2, ..., u_k do
3:      p̂_σ = 1
4:      for each i = 1, ..., k − 1 do
5:          S_i = {v, σ_2, ..., σ_i}
6:          n_i = |Cut(S_i, σ_{i+1})|
7:          (ĉ_i(u))_{u∈S_i} = EstimateCuts(G, v, S_i, α, β/k^{O(k)}, O(ρ/k^2))
8:          ĉ_i = Σ_{u∈S_i} ĉ_i(u)
9:          p̂_σ = p̂_σ · n_i/ĉ_i
10:     p̂ = p̂ + p̂_σ
11: return p̂
```

Lemma 42. *For any $g = G[S] \in B(v)$ and $\rho > 0$, Apx-Prob$(G, S, \alpha, \beta, \rho)$ runs in time $k^{O(k)} \frac{1}{\rho^2 \alpha^4} \log \frac{1}{\beta}$ and with probability $1 - \text{poly}\,\frac{\beta}{k}$ returns a multiplicative $(1 \pm \rho)$-approximation $\widehat{p}_v(g)$ of $p_v(g)$.*

Proof sketch. The running time analysis is straightforward. For the correctness, let $\Sigma$ be the set of all permutations $\sigma = (\sigma_1, \ldots, \sigma_k)$ of $v, u_2, \ldots, u_k$ such that $\sigma_1 = v$. For each $\sigma \in \Sigma$, let $S_i^\sigma$ be the first $i$ vertices in $S$ as given by $\sigma$. Note that Apx-Prob returns:

$$\widehat{p} = \sum_{\sigma \in \Sigma} \widehat{p}_\sigma = \sum_{\sigma \in \Sigma} \prod_{i=1}^{k-1} \frac{n(S_i^\sigma)}{\widehat{c}(S_i^\sigma)}, \tag{119}$$

where $n(S_i^\sigma)$ is the size of the cut between $S_i^\sigma$ and $\sigma_{i+1}$, and $\widehat{c}(S_i^\sigma)$ is the value of $\widehat{c_i}$ used by Apx-Prob. Instead, Prob$(G, S)$ returns:

$$p = \sum_{\sigma \in \Sigma} p_\sigma = \sum_{\sigma \in \Sigma} \prod_{i=1}^{k-1} \frac{n(S_i^\sigma)}{c(S_i^\sigma)}. \tag{120}$$

Therefore,

$$\frac{\widehat{p}}{p} = \frac{\sum_{\sigma \in \Sigma} \prod_{i=1}^{k-1} c(S_i^\sigma)}{\sum_{\sigma \in \Sigma} \prod_{i=1}^{k-1} \widehat{c}(S_i^\sigma)}. \tag{121}$$

Look at a single term $\sigma$. Note that $\widehat{c}(S_i^\sigma)$ is estimated as in EstimateCuts$(G, v, U, \alpha, \beta, \delta)$, but with $k$ times as many samples. Therefore, the guarantees of Lemma 39 apply, but the deviation probability shrinks by a factor $2^{-k}$. Since there are at most $2^k$ different subsets $S_i^\sigma$, by a union bound, with probability $1 - \text{poly}\,\frac{\beta}{k}$, we have $\widehat{c}(S_i^\sigma) \in c(S_i^\sigma) \cdot (1 \pm \delta\, k^2) c(S_i^\sigma)$ for all $S_i^\sigma$ simultaneously, where we used $|S| \le k$. Thus,

$$\frac{\prod_{i=1}^{k-1} c(S_i^\sigma)}{\prod_{i=1}^{k-1} \widehat{c}(S_i^\sigma)} = \prod_{i=1}^{k-1} \frac{c(S_i^\sigma)}{\widehat{c}(S_i^\sigma)} \in \left( \frac{1}{(1 \pm \delta\, k^2)} \right)^{k-1}. \tag{122}$$

For $\delta = O(\frac{\rho}{k^3})$ small enough, the right-hand side is in $(1 \pm \rho)$. This gives $\frac{\widehat{p}}{p} \in (1 \pm \rho)$, as claimed. □

*7.2.4 Coupling the Algorithms.* We conclude the sampling phase of Apx-Ugs. After drawing $v$, we invoke Apx-Rand-Grow$(G, v, \alpha, \beta, \gamma)$ and Apx-Prob$(G, S, \alpha, \beta, \rho)$ with $\gamma = \rho = \varepsilon^3 k^{-C_2 k}$, where $S$ is the set of vertices returned by Apx-Rand-Grow. Hence, we have a random graphlet $g = G[S]$ together with a probability estimate $\widehat{p}_v(g)$. We then accept $g$ with probability inversely proportional to $\widehat{p}_v(g)$. For reference, see the code below.

The next two lemmas show that Apx-Ugs satisfies the claims of Theorem 6.

Lemma 43. *Suppose that the preprocessing of Apx-Ugs$(G, \varepsilon)$ succeeds (Lemma 28). Then, each invocation of Sample( ) returns a graphlet independently and $\varepsilon$-uniformly at random from $G$.*

Proof. By Lemma 28, Apx-DD$(G, \alpha, \beta)$ with high probability returns an $(\alpha, \beta)$-DD order for $G$. The rest of the proof is conditioned on this event. We will couple the sampling phases of Apx-Ugs$(G, \varepsilon)$ and Ugs-Compare$(G, \varepsilon)$. Note that the the preprocessing phases of the two algorithms are identical (save for the fact that Ugs-Compare also sorts the adjacency lists). In particular, they use the same order $\prec$ over $V(G)$, which induces the same bucketing $\{B(v)\}_{v \in V}$, as well as the same bucket size estimates $\{b_v\}_{v \in V}$, and therefore also the same distribution $p$ over $V$.

Let $H = \{v \in V : b_v > 0\}$. Let $\mathcal{U}_V$ and $\mathcal{U}_H$ be the uniform distributions, respectively, over $\cup_{v \in V} B(v)$ and $\cup_{v \in H} B(v)$, and let $q$ be the output distribution of Apx-Ugs::Sample. Our claim is that $\text{tvd}(q, \mathcal{U}_V) \le \varepsilon$. By the triangle inequality, $\text{tvd}(q, \mathcal{U}_V) \le \text{tvd}(\mathcal{U}_H, \mathcal{U}_V) + \text{tvd}(q, \mathcal{U}_H)$, and by

Definition 27, the buckets indexed by $H$ hold a fraction at least $1 - \beta = 1 - \frac{\varepsilon}{2}$ of all graphlets, hence, $\text{tvd}(\mathcal{U}_H, \mathcal{U}_V) \leq \frac{\varepsilon}{2}$. Therefore, to prove that $\text{tvd}(q, \mathcal{U}_V) \leq \varepsilon$, we need only to prove that $\text{tvd}(q, \mathcal{U}_H) \leq \frac{\varepsilon}{2}$, which we do in the remainder.

**ALGORITHM 10:** Apx-Ugs$(G, \varepsilon)$

1: let $C_1, C_2$ = large enough universal constants
2: let $\beta = \frac{\varepsilon}{2}$ and let $(\prec, \boldsymbol{b})$ = Apx-DD$(G, \alpha, \beta)$
3: let $Z = \sum_{v \in V} b_v$, and for each $v \in V$ let $p(v) = \frac{b_v}{Z}$
4: sort $V(G)$ according to $\prec$
5:
6: **function** Sample( )
7: **while** true **do**
8: draw $v$ from the distribution $p$
9: $S$ = Apx-Rand-Grow$(G, v, \alpha, \beta, \varepsilon^3 k^{-C_2 k})$
10: $\widehat{p}(S)$ = Apx-Prob$(G, S, \alpha, \beta, \varepsilon^3 k^{-C_2 k})$
11: with probability min $(1, \frac{1}{p(v)\widehat{p}(S)} \frac{\beta}{Z} k^{-C_1 k})$ **return** $S$

First, by Lemma 37, $\mathcal{U}_H$ is precisely the output distribution of Ugs-Compare::Sample. Thus, we will couple Ugs-Compare::Sample and Apx-Ugs::Sample; under this coupling they will return the same graphlet with probability at least $1 - \frac{\varepsilon}{2}$, establishing that $\text{tvd}(q, \mathcal{U}_H) \leq \frac{\varepsilon}{2}$.

To begin, since the Ugs-Compare::Sample and Apx-Ugs::Sample use the same distribution $p$ over the buckets, we can couple them so they choose the same bucket $B(v)$. Now, let $S_P$ denote the random set of vertices drawn by Ugs-Compare::Sample at line 10, and by $S_Q$ the one drawn by Apx-Ugs::Sample at line 9. As the two algorithms invoke, respectively, Rand-Grow$(G, v)$ and Apx-Rand-Grow$(G, v, \alpha, \frac{\varepsilon}{2}, \varepsilon^3 k^{-C_2 k})$, Lemma 40 yields:

$$\text{tvd}(S_P, S_Q) \leq \varepsilon^3 k^{-C_2 k} + \text{poly}\, \frac{\varepsilon}{k}. \tag{123}$$

Hence, we can couple the two algorithms so $\mathbb{P}(S_Q \neq S_P) \leq \varepsilon^3 k^{-C_2 k}$.

Now, let $X_P$ be the indicator random variable of the event that Ugs-Compare::Sample accepts $S_P$ (line 12 of Ugs-Compare), and $X_Q$ the indicator random variable of the event that Apx-Ugs::Sample accepts $S_Q$ (line 11 of Apx-Ugs). The outcome of Ugs-Compare::Sample is the pair $(S_P, X_P)$, and that of Apx-Ugs::Sample is the pair $(S_Q, X_Q)$. Let $\mathcal{D}_P$ and $\mathcal{D}_Q$ be the distributions of, respectively, $(S_P, X_P)$ and $(S_Q, X_Q)$. Note that $\mathcal{D}_P(\cdot | X_P = 1)$ and $\mathcal{D}_Q(\cdot | X_Q = 1)$ are the distributions of the graphlets returned by, respectively, Ugs-Compare::Sample and Apx-Ugs::Sample. Thus, our goal is to show:

$$\text{tvd}(\mathcal{D}_P(\cdot | X_P = 1), \mathcal{D}_Q(\cdot | X_Q = 1)) \leq \frac{\varepsilon}{2}. \tag{124}$$

Let $X_\vee = \max(X_P, X_Q)$ be the indicator random variable of the event that at least one algorithm accepts its graphlet. Clearly, $\mathbb{P}(X_\vee = 1) \geq \mathbb{P}(X_P = 1)$, and by Lemma 37, $P(X_P = 1) \geq \varepsilon^2 k^{-O(k)}$. By the triangle inequality:

$$\begin{aligned} \text{tvd}(\mathcal{D}_P(\cdot | X_P = 1), \mathcal{D}_Q(\cdot | X_Q = 1)) \leq\ & \text{tvd}(\mathcal{D}_P(\cdot | X_P = 1), \mathcal{D}_P(\cdot | X_\vee = 1)) \\ & + \text{tvd}(\mathcal{D}_P(\cdot | X_\vee = 1), \mathcal{D}_Q(\cdot | X_\vee = 1)) \\ & + \text{tvd}(\mathcal{D}_Q(\cdot | X_Q = 1), \mathcal{D}_Q(\cdot | X_\vee = 1).) \end{aligned} \tag{125}$$

Let us start by bounding the middle term. We have:

$$\begin{aligned}
\mathrm{tvd}(\mathcal{D}_P(\cdot|X_\vee = 1), \mathcal{D}_Q(\cdot|X_\vee = 1)) &\le \mathbb{P}(S_Q \neq S_P \mid X_\vee = 1) && \text{by the coupling} && (126)\\
&\le \frac{\mathbb{P}(S_Q \neq S_P)}{\mathbb{P}(X_\vee = 1)} && && (127)\\
&\le \frac{\mathbb{P}(S_Q \neq S_P)}{\mathbb{P}(X_P = 1)} && X_\vee = \max(X_P, X_Q) && (128)\\
&\le \frac{\mathbb{P}(S_Q \neq S_P)}{\varepsilon^2 k^{-O(k)}} && \text{Lemma 37} && (129)\\
&\le \frac{\varepsilon^3 k^{-C_2 k}}{\varepsilon^2 k^{-O(k)}} && \text{see above} && (130)\\
&= \varepsilon k^{-(C_2+C_3)k} && && (131)
\end{aligned}$$

for some $C_3$ independent of $C_2$.

We bound similarly the sum of the other two terms. For the first term note that:

$$\mathrm{tvd}(\mathcal{D}_P(\cdot|X_P = 1), \mathcal{D}_P(\cdot|X_\vee = 1)) \le \mathbb{P}(X_P = 0 \mid X_\vee = 1). \tag{132}$$

This is true since $\mathcal{D}_P(\cdot|X_P = 1)$ is just $\mathcal{D}_P(\cdot|X_\vee = 1)$ conditioned on $X_P = 1$, an event that has probability $1 - \mathbb{P}(X_P = 0 \mid X_\vee = 1)$. Symmetrically, for the last term

$$\mathrm{tvd}(\mathcal{D}_Q(\cdot|X_Q = 1), \mathcal{D}_Q(\cdot|X_\vee = 1)) \le \mathbb{P}(X_Q = 0 \mid X_\vee = 1). \tag{133}$$

Thus:

$$\begin{aligned}
&\mathrm{tvd}(\mathcal{D}_P(\cdot|X_P = 1), \mathcal{D}_P(\cdot|X_\vee = 1)) + \mathrm{tvd}(\mathcal{D}_Q(\cdot|X_Q = 1), \mathcal{D}_Q(\cdot|X_\vee = 1)) && (134)\\
&\quad \le \mathbb{P}(X_P = 0 \mid X_\vee = 1) + \mathbb{P}(X_Q = 0 \mid X_\vee = 1). && (135)
\end{aligned}$$

Now,

$$\begin{aligned}
\mathbb{P}(X_P = 0 \mid X_\vee = 1) + \mathbb{P}(X_Q = 0 \mid X_\vee = 1) &= \mathbb{P}(X_P \neq X_Q \mid X_\vee = 1) && && (136)\\
&\le \frac{\mathbb{P}(X_P \neq X_Q)}{\mathbb{P}(X_\vee = 1)} && && (137)\\
&\le \frac{\mathbb{P}(X_P \neq X_Q)}{\varepsilon^2 k^{-O(k)}} && \text{see above.} && (138)
\end{aligned}$$

For the numerator,

$$\begin{aligned}
\mathbb{P}(X_Q \neq X_P) &\le \mathbb{P}(S_Q \neq S_P) + \mathbb{P}(X_Q \neq X_P \mid S_Q = S_P) && && (139)\\
&\le \varepsilon^3 k^{-C_2 k} + \mathbb{P}(X_Q \neq X_P \mid S_Q = S_P) && \text{see above.} && (140)
\end{aligned}$$

As said, $\mathbb{P}(S_Q \neq S_P) \le \varepsilon_1$. As $X_Q$ and $X_P$ are binary, our coupling yields:

$$\begin{aligned}
\mathbb{P}(X_Q \neq X_P \mid S_Q = S_P) &= \left|\mathbb{P}(X_P = 1 \mid S_Q = S_P) - \mathbb{P}(X_Q = 1 \mid S_Q = S_P)\right| && && (141)\\
&\le \left|\frac{\mathbb{P}(X_P = 1 \mid S_Q = S_P)}{\mathbb{P}(X_Q = 1 \mid S_Q = S_P)} - 1\right|. && && (142)
\end{aligned}$$

Now, let $S$ be any realization of $S_Q$ and $S_P$. By construction of the algorithms, $\mathbb{P}(X_P = 1 \mid S_P = S) = \nu$, and $\mathbb{P}(X_Q = 1 \mid S_Q = S) = \min(1, \widehat{\nu})$, where:

$$\nu = \frac{1}{p(v)\,p(S)}\frac{\beta}{Z}k^{-C_1 k}, \qquad \widehat{\nu} = \frac{1}{p(v)\,\widehat{p}(S)}\frac{\beta}{Z}k^{-C_1 k}. \tag{143}$$

Therefore:

$$\mathbb{P}(X_Q \neq X_P \mid S_Q = S_P) \leq \left|\frac{\nu}{\min(1,\widehat{\nu})} - 1\right| \leq \left|\frac{\nu}{\widehat{\nu}} - 1\right| = \left|\frac{\widehat{p}(S)}{p(S)} - 1\right|, \tag{144}$$

where the second inequality holds since, if $\widehat{\nu} > 1$, then

$$\left|\frac{\nu}{\min(1,\widehat{\nu})} - 1\right| = \left|\frac{\nu}{1} - 1\right| < \left|\frac{\nu}{\widehat{\nu}} - 1\right|. \tag{145}$$

Now, by Lemma 42, with probability $1 - \text{poly}\,\frac{\varepsilon}{k}$, we have $\widehat{p}(S) \in (1 \pm \varepsilon^3 k^{-C_2 k})p(S)$. So, if this event holds, then we have $\mathbb{P}(X_Q \neq X_P \mid S_Q = S_P) \leq \varepsilon^3 k^{-C_2 k}$. If it fails, then we still have the trivial bound $\mathbb{P}(X_Q \neq X_P \mid S_Q = S_P) \leq 1$. By the law of total probability,

$$\mathbb{P}(X_Q \neq X_P \mid S_Q = S_P) \leq \left(1 - \text{poly}\,\frac{\varepsilon}{k}\right)\varepsilon^3 k^{-C_2 k} + \text{poly}\,\frac{\varepsilon}{k} = O(\varepsilon^3 k^{-C_2 k}). \tag{146}$$

Applying these two bounds to the right-hand side of Equation (139), we obtain:

$$\mathbb{P}(X_Q \neq X_P) = O(\varepsilon^3 k^{-C_2 k}). \tag{147}$$

Going back to Equation (138), we obtain:

$$\mathbb{P}(X_P = 0 \mid X_\vee = 1) + \mathbb{P}(X_Q = 0 \mid X_\vee = 1) = O\left(\frac{\varepsilon^3 k^{-C_2 k}}{\varepsilon^2 k^{-O(k)}}\right) = O(\varepsilon). \tag{148}$$

By taking this bound together with Equation (131), we conclude that:

$$\text{tvd}(\mathcal{D}_P(\cdot|X_P = 1), \mathcal{D}_Q(\cdot|X_Q = 1)) = O(\varepsilon), \tag{149}$$

which we can bring below $\frac{\varepsilon}{2}$ by adjusting the constants. This concludes the proof. □

Lemma 44. *Suppose that the preprocessing of Apx-Ugs$(G, \varepsilon)$ succeeds (Lemma 28). Then, each invocation of Sample( ) has expected running time $k^{O(k)}\varepsilon^{-8-\frac{4}{k-1}} \log \frac{1}{\varepsilon}$.*

Proof. First, we bound the expected number of rounds of Apx-Ugs::Apx-Rand-Grow. Recall $X_P$ and $X_Q$ from the proof of Lemma 43. Note that $\mathbb{P}(X_Q = 1) \geq \mathbb{P}(X_P = 1) - \mathbb{P}(X_Q \neq X_P)$. Moreover, the proof of Lemma 43 showed $\mathbb{P}(X_Q \neq X_P) = O(\varepsilon^3 k^{-C_2 k})$. Therefore, $\mathbb{P}(X_Q = 1) \geq \mathbb{P}(X_P = 1) - O(\varepsilon^3 k^{-C_2 k})$. However, by Lemma 37, $\mathbb{P}(X_P = 1) \geq k^{-C_3}\varepsilon^2$ for some constant $C_3$. Therefore, $\mathbb{P}(X_Q = 1) \geq \varepsilon^2 k^{-C_3 k} - \varepsilon^3 k^{-C_2 k} = k^{-O(k)}\varepsilon^2$. So, the expected number of rounds performed by Apx-Ugs::Apx-Rand-Grow is bounded by $k^{O(k)}\varepsilon^{-2}$.

Now, we bound the expected time spent in each round. By Lemma 41, and as $\gamma = \varepsilon^3 k^{-C_2 k}$ and $\alpha = \beta^{\frac{1}{k-1}}\frac{1}{6k^3}$ and $\beta = \frac{\varepsilon}{2}$, Apx-Rand-Grow$(G, v, \alpha, \beta, \gamma)$ has expected running time at most:

$$O\left(\frac{k^9}{\gamma^2\alpha^4}\log\frac{1}{\gamma}\right) = k^{O(k)}\varepsilon^{-6-\frac{4}{k-1}}\log\frac{1}{\varepsilon}. \tag{150}$$

Note that the bound holds at each round, regardless of past events. Using Lemma 42, one can show the same bound holds for the running time Apx-Prob$(G, S, \alpha, \beta, \rho)$, where $\rho = \varepsilon^3 k^{-C_2 k}$.

Therefore, the total expected running time satisfies:

$$\mathbb{E}[T] \leq k^{O(k)}\varepsilon^{-2} \cdot k^{O(k)}\varepsilon^{-6-\frac{4}{k-1}}\log\frac{1}{\varepsilon} = k^{O(k)}\varepsilon^{-8-\frac{4}{k-1}}\log\frac{1}{\varepsilon}, \tag{151}$$

which concludes the proof. □

## 8 CONCLUSIONS

We have shown that, starting from just sorting a graph in linear time, one can overcome the usual inefficiency of rejection sampling of graphlets. This idea yields the first efficient uniform and $\varepsilon$-uniform graphlet sampling algorithms, with preprocessing times $O(|G|)$ and $O(|V(G)|\log|V(G)|)$. These are the first algorithms with strong theoretical guarantees for these problems in a long line of research that spans the past decade. Due to their simplicity, we believe that our algorithms are amenable to being ported in parallel, distributed, or dynamic settings; these are all directions for future research. We also leave open the problem of determining whether $\Omega(|G|)$ operations are necessary for uniform graphlet sampling when $|G| = \omega(|V(G)|)$; a positive answer would imply the optimality of our uniform sampling algorithm.

## APPENDICES

## A ANCILLARY RESULTS

Lemma 45. *Let $G = (V, E)$ be any graph, and for any $v \in V$ let $N_v$ be the number of $k$-graphlets of $G$ containing $v$. If $N_v > 0$, then:*

$$N_v \geq \frac{(d_v)^{k-1}}{(k-1)^{k-1}} = (d_v)^{k-1}k^{-O(k)}. \tag{152}$$

*Moreover, if $d_u \leq \Delta$ for all $u \in G$, then:*

$$N_v \leq (k-1)!(\Delta)^{k-1} = (\Delta)^{k-1}k^{O(k)}. \tag{153}$$

Proof. For the lower bound, if $d_v \leq k-1$, then $\frac{(d_v)^{k-1}}{(k-1)^{k-1}} \leq 1$, so if $N_v \geq 1$, then $N_v \geq \frac{(d_v)^{k-1}}{(k-1)^{k-1}}$. If instead $d_v > k-1$, then $N_v \geq \binom{d_v}{k-1}$, since any set of vertices formed by $v$ and $k-1$ of its neighbors is connected. However $N_v \geq \binom{d_v}{k-1} \geq \frac{(d_v)^{k-1}}{(k-1)^{k-1}}$, since $\binom{a}{b} \geq \frac{a^b}{b^b}$ for all $a \geq 1$ and all $b \in [a]$.

For the upper bound, note that we can construct a connected subgraph on $k$ vertices containing $v$ by starting with $S_1 = \{v\}$ and at every step $i = 1, \ldots, k-1$ choosing a neighbor of $S_i$ in $G \setminus S_i$. Since each $u \in G$ has degree at most $\Delta$, then $S_i$ has at most $i\Delta$ neighbors. Thus, the total number of choices is at most $\prod_{i=1}^{k-1} i\Delta = (k-1)!(\Delta)^{k-1}$. □

## B PROOF OF THEOREM 5

We start by running the preprocessing phase of Ugs. Let $N_k = \sum_{v\in V}|B(v)|$ be the total number of $k$-graphlet occurrences in $G$. We compute an estimate $\widehat{N}_k$ of $N_k$ such that $|\widehat{N} - N_k| \leq \varepsilon_0 N$ with probability at least $1 - \frac{\delta}{2}$. To this end, for each $v \in V$ such that $B(v) \neq \emptyset$, we estimate $|B(v)|$ up to a multiplicative error $(1 \pm \varepsilon_0)$ with probability $1 - \frac{\delta}{2n}$, as detailed below. By a union bound, setting $\widehat{N}_k$ to the sum of all those estimates will satisfy the bound above.

To estimate $|B(v)|$, we run the sampling routine of Ugs over bucket $B(v)$. However, after $S$ is sampled, instead of rejecting it randomly, we return the probability $p(S)$ computed by Prob(G,S). By Lemma 26, $p(S)$ is exactly the probability that $S$ is sampled. Thus, if $X$ is the random variable denoting the output value of this modified routine, then we have:

$$\mathbb{E}[X] = \sum_{S\in B(v)} p(S)\cdot\frac{1}{p(S)} = |B(v)|. \tag{154}$$

It remains to apply concentration bounds. To this end, note that $X \leq k^{O(k)}\mathbb{E}[X]$ by Lemma 25. Thus, $X \in [0, k^{O(k)}\mathbb{E}[X]]$. Therefore, by averaging over $\ell$ independent samples of $X$, we obtain:

$$\mathbb{P}\left(\left|\frac{1}{\ell}\sum_{i=1}^{\ell} X_i - \mathbb{E}[X]\right| > \varepsilon_0 \mathbb{E}[X]\right) < 2\exp\left(-\frac{(\varepsilon_0\mathbb{E}[X])^2\ell}{(k^{O(k)}\mathbb{E}[X])^2}\right) = 2\exp\left(-\frac{\varepsilon_0^2\ell}{k^{O(k)}}\right). \tag{155}$$

Therefore, our guarantees are achieved by setting $\ell = k^{O(k)}\varepsilon_0^{-2}\ln\frac{2n}{\delta}$. Since we have at most $n$ nonempty buckets, to estimate $\widehat{N}_k$ we use a total of $k^{O(k)}n\varepsilon_0^{-2}\ln\frac{2n}{\delta}$ samples.

Next, we estimate the graphlet frequencies via the sampling routine of UGS. For every distinct (up to isomorphism) $k$-vertex simple connected graph $H$, let $N_H$ be the number of distinct $k$-graphlet occurrences of $H$ in $G$. Clearly, $\sum_H N_H = N_k$. Let $f_H = \frac{N_H}{N_k}$ be the relative frequency of $H$. Now, we take $\text{poly}(k)\,\frac{4}{\varepsilon_1^2}\ln\frac{1}{\delta}$ independent uniform samples. By standard concentration bounds, we obtain an estimate $\widehat{f}_H$ of $f_H$ such that $|\widehat{f}_H - f_H| \leq \frac{\varepsilon_1}{2}$ with probability at least $1 - \delta_1^{-\text{poly}(k)}$. Since there are $2^{\text{poly}(k)}$ distinct $k$-vertex (connected) graphs, by a union bound, we obtain such an estimate $\widehat{f}_H$ for all $H$ simultaneously with probability $1 - \frac{\delta}{2}$.

Now, for all $H$, we set $\widehat{N}_H = \widehat{N_k}\widehat{f}_H$. By a union bound, with probability at least $1-\delta$, we have simultaneously for all $H$:

$$\begin{aligned}
\widehat{N}_H - N_H &= \widehat{N_k}\widehat{f}_H - N_H && (156)\\
&\leq N_k(1+\varepsilon_0)\left(f_H + \frac{\varepsilon_1}{2}\right) - N_H && (157)\\
&= (1+\varepsilon_0)N_H + N_k(1+\varepsilon_0)\frac{\varepsilon_1}{2} - N_H && (158)\\
&\leq \varepsilon_0 N_H + \varepsilon_1 N_k, && (159)
\end{aligned}$$

on the one hand, and similarly, $\widehat{N}_H - N_H \geq -\varepsilon_0 N_H - \varepsilon_1 N_k$ on the other hand. Therefore, $|\widehat{N}_H - N_k| \leq \varepsilon_0 N_H + \varepsilon_1 N_k$ with probability at least $1-\delta$ for all $H$ simultaneously, as desired.

The running time is given by (i) the preprocessing phase, which takes time $O(k^2 + m)$; (ii) $k^{O(k)}n\varepsilon_0^{-2}\ln\frac{2n}{\delta} + \text{poly}(k)\,\frac{4}{\varepsilon_1^2}\ln\frac{1}{\delta}$ samples, each one taking time $k^{O(k)}\log\Delta$ as per Theorem 4. This gives a total running time of:

$$O(k^2 n + m) + \left(k^{O(k)}n\varepsilon_0^{-2}\ln\frac{2n}{\delta} + \text{poly}(k)\,\frac{4}{\varepsilon_1^2}\ln\frac{1}{\delta}\right)k^{O(k)}\log\Delta, \tag{160}$$

which is in $O(m) + k^{O(k)}(n\varepsilon_0^{-2}\ln\frac{n}{\delta} + \varepsilon_1^{-2}\ln\frac{1}{\delta})\log\Delta$. The proof is complete.

## C EPSILON-UNIFORM SAMPLING VIA COLOR CODING

We show how to use the color coding algorithm of [12] in a black-box fashion to perform $\varepsilon$-uniform sampling from $G$. The overhead in the running time and space is $2^{O(k)}O(\log\frac{1}{\varepsilon})$, and the overhead in the sampling time is $2^{O(k)}O((\log\frac{1}{\varepsilon})^2)$.

First, we perform $\ell = O(e^k\log\frac{1}{\varepsilon})$ independent runs of the preprocessing phase of the algorithm of [12], storing all their output count tables. This gives a time-and-space $O(e^k\log\frac{1}{\varepsilon})$ overhead with respect to [12]. In each run, any graphlet $g$ has probability $\frac{k^k}{k!} \geq e^{-k}$ of becoming colorful. Thus, with $O(e^k\log\frac{1}{\varepsilon})$ independent runs, $g$ is colorful with probability $1 - \text{poly}\,\varepsilon$ in at least one run and appears in the respective count table. As shown in [12], for each run $i = 1, \ldots, \ell$ one can estimate, within a multiplicative $(1\pm\varepsilon)$ factor, the number of colorful graphlets $N_i$, using $O(\frac{k^{O(k)}}{\varepsilon^2})$ samples. In time $O(\frac{k^{O(k)}}{\varepsilon^2}\log\frac{1}{\varepsilon})$, we can do so for all runs with probability $1 - \text{poly}\,\varepsilon$. This concludes the preprocessing phase.

For sampling, we choose a random run $i \in [\ell]$ with probability proportional to the estimate of $N_i$. Then, we draw a graphlet from that run uniformly at random using the sampling phase of

[12]. This yields a graphlet uniformly at random from the union of all colorful graphlets in all runs. Thus, the probability that a specific graphlet $g$ is sampled is proportional to the number of runs $\ell(g)$ where $g$ is colorful, which we can compute by looking at the colors assigned to $g$ by every run in time $\ell = O(e^k \log \frac{1}{\varepsilon})$. Then, we accept $g$ with probability $\frac{1}{\ell(g)} \geq \frac{1}{\ell}$. Therefore, we need at most $\ell = O(e^k \log \frac{1}{\varepsilon})$ trials in expectation before a graphlet is accepted. This gives an overhead of $O(e^k \log \frac{1}{\varepsilon})^2$ in the sampling phase. This construction can be derandomized using an $(n, k)$-family of perfect hash functions of size $\ell = 2^{O(k)} \log n$ (see [4]). This derandomization would increase the time and space of the preprocessing by a factor $\log n$, but we would still need to estimate the number of graphlets in each run, so the final distribution would still be non-uniform.

## REFERENCES


[1] Matteo Agostini, Marco Bressan, and Shahrzad Haddadan. 2019. Mixing time bounds for graphlet random walks. *Inform. Process. Lett.* 152 (2019), 105851. DOI: https://doi.org/10.1016/j.ipl.2019.105851

[2] David Aldous and James Fill. 1995. *Reversible Markov Chains and Random Walks on Graphs*. Retrieved from https://www.stat.berkeley.edu/ aldous/RWG/book.pdf.

[3] N. Alon, P. Dao, I. Hajirasouliha, F. Hormozdiari, and S. C. Sahinalp. 2008. Biomolecular network motif counting and discovery by color coding. *Bioinformatics* 24, 13 (July 2008), i241–249. DOI: https://doi.org/10.1093/bioinformatics/btn163

[4] Noga Alon, Raphael Yuster, and Uri Zwick. 1995. Color-coding. *J. ACM* 42, 4 (1995), 844–856. DOI: https://doi.org/10.1145/210332.210337

[5] Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. 2018. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. In *Proceedings of the ITCS*. 6:1–6:20. DOI: https://doi.org/10.4230/LIPIcs.ITCS.2019.6

[6] Suman K. Bera, Lior Gishboliner, Yevgeny Levanzov, C. Seshadhri, and Asaf Shapira. 2022. Counting subgraphs in degenerate graphs. *J. ACM* 69, 3 (2022), 23:1–23:21. DOI: https://doi.org/10.1145/3520240

[7] Mansurul A. Bhuiyan, Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. 2012. GUISE: Uniform sampling of graphlets for large graph analysis. In *Proceedings of the IEEE ICDM*. 91–100. DOI: https://doi.org/10.1109/ICDM.2012.87

[8] Amartya Shankha Biswas, Talya Eden, and Ronitt Rubinfeld. 2021. Towards a decomposition-optimal algorithm for counting and sampling arbitrary motifs in sublinear time. In *Proceedings of the APPROX/RANDOM*. 55:1–55:19. DOI: https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2021.55

[9] Anthony Bonato, David F. Gleich, Myunghwan Kim, Dieter Mitsche, Paweł Prałat, Yanhua Tian, and Stephen J. Young. 2014. Dimensionality of social networks using motifs and eigenvalues. *PloS One* 9, 9 (2014), e106052. DOI: https://doi.org/10.1371/journal.pone.0106052

[10] Marco Bressan. 2021. Efficient and near-optimal algorithms for sampling connected subgraphs. In *Proceedings of the ACM STOC*. 1132–1143. DOI: https://doi.org/10.1145/3406325.3451042

[11] Marco Bressan. 2021. Faster algorithms for counting subgraphs in sparse graphs. *Algorithmica* (2021). DOI: https://doi.org/10.1007/s00453-021-00811-0

[12] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2017. Counting graphlets: Space vs time. In *Proceedings of the ACM WSDM*. 557–566. DOI: https://doi.org/10.1145/3018661.3018732

[13] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2018. Motif counting beyond five nodes. *ACM Trans. Knowl. Discov. Data* 12, 4 (Apr. 2018). DOI: https://doi.org/10.1145/3186586

[14] Marco Bressan, Stefano Leucci, and Alessandro Panconesi. 2019. Motivo: Fast motif counting via succinct color coding and adaptive sampling. *Proc. VLDB Endow.* 12, 11 (July 2019), 1651–1663. DOI: https://doi.org/10.14778/3342263.3342640

[15] Marco Bressan, Stefano Leucci, and Alessandro Panconesi. 2021. Faster motif counting via succinct color coding and adaptive sampling. *ACM Trans. Knowl. Discov. Data* 15, 6 (May 2021). DOI: https://doi.org/10.1145/3447397

[16] Jin Chen, Wynne Hsu, Mong Li Lee, and See-Kiong Ng. 2006. NeMoFinder: Dissecting genome-wide protein-protein interactions with meso-scale network motifs. In *Proceedings of the ACM KDD*. 106–115. DOI: https://doi.org/10.1145/1150402.1150418

[17] Xiaowei Chen, Yongkun Li, Pinghui Wang, and John C. S. Lui. 2016. A general framework for estimating graphlet statistics via random walk. *Proc. VLDB Endow.* 10, 3 (Nov. 2016), 253–264. DOI: https://doi.org/10.14778/3021924.3021940

[18] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and subgraph listing algorithms. *SIAM J. Comput.* 14, 1 (1985), 210–223. DOI: https://doi.org/10.1137/0214017

[19] David Easley and Jon Kleinberg. 2010. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press. DOI: https://doi.org/10.1017/CBO9780511761942
[20] Talya Eden, Amit Levi, Dana Ron, and C. Seshadhri. 2017. Approximately counting triangles in sublinear time. *SIAM J. Comput.* 46, 5 (2017), 1603–1646. DOI: https://doi.org/10.1137/15M1054389
[21] Talya Eden, Saleet Mossel, and Ronitt Rubinfeld. 2021. Sampling multiple edges efficiently. In *Proceedings of the APPROX/RANDOM*. 51:1–51:15. DOI: https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2021.51
[22] Talya Eden, Dana Ron, and C. Seshadhri. 2020. On approximating the number of k-cliques in sublinear time. *SIAM J. Comput.* 49, 4 (2020), 747–771. DOI: https://doi.org/10.1137/18M1176701
[23] Talya Eden and Will Rosenbaum. 2018. On sampling edges almost uniformly. In *Proceedings of the SOSA*. 7:1–7:9. DOI: https://doi.org/10.4230/OASIcs.SOSA.2018.7
[24] Guyue Han and Harish Sethu. 2016. Waddling random walk: Fast and accurate mining of motif statistics in large graphs. In *Proceedings of the IEEE ICDM*. 181–190. DOI: https://doi.org/10.1109/ICDM.2016.0029
[25] Madhav Jha, C. Seshadhri, and Ali Pinar. 2015. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proceedings of the WWW*. 495–505. DOI: https://doi.org/10.1145/2736277.2741101
[26] Tali Kaufman, Michael Krivelevich, and Dana Ron. 2004. Tight bounds for testing bipartiteness in general graphs. *SIAM J. Comput.* 33, 6 (2004), 1441–1483. DOI: https://doi.org/10.1137/S0097539703436424
[27] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. 2009. *Markov Chains and Mixing Times*. American Mathematical Society.
[28] Pan Li, Hoang Dau, Gregory Puleo, and Olgica Milenkovic. 2017. Motif clustering and overlapping clustering for social network analysis. In *Proceedings of the IEEE INFOCOM*. DOI: https://doi.org/10.1109/INFOCOM.2017.8056956
[29] Ryuta Matsuno and Aristides Gionis. 2020. Improved mixing time for k-subgraph sampling. In *Proceedings of the SIAM SDM*. 568–576. DOI: https://doi.org/10.1137/1.9781611976236.64
[30] David W. Matula and Leland L. Beck. 1983. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM* 30, 3 (July 1983), 417–427. DOI: https://doi.org/10.1145/2402.322385
[31] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. 2002. Network motifs: Simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827. DOI: https://doi.org/10.1126/science.298.5594.824
[32] Kirill Paramonov, Dmitry Shemetov, and James Sharpnack. 2019. Estimating graphlet statistics via lifting. In *Proceedings of the ACM KDD*. 587–595. DOI: https://doi.org/10.1145/3292500.3330995
[33] Hao Peng, Jianxin Li, Qiran Gong, Yuanxin Ning, Senzhang Wang, and Lifang He. 2020. Motif-matching based subgraph-level attentional convolutional network for graph classification. *Proc AAAI* 34, 04 (Apr. 2020), 5387–5394. DOI: https://doi.org/10.1609/aaai.v34i04.5987
[34] Nataša Pržulj. 2007. Biological network comparison using graphlet degree distribution. *Bioinformatics* 23, 2 (2007), e177–e183. DOI: https://doi.org/10.1093/bioinformatics/btl301
[35] Tanay Kumar Saha and Mohammad Al Hasan. 2015. Finding network motifs using MCMC sampling. In *Proceedings of the CompleNet*. 13–24. DOI: https://doi.org/10.1007/978-3-319-16112-9_2
[36] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *Proceedings of the AISTATS*. 488–495. Retrieved from https://proceedings.mlr.press/v5/shervashidze09a.html.
[37] Charalampos E. Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. 2017. Scalable motif-aware graph clustering. In *Proceedings of the WWW*. 1451–1460. DOI: https://doi.org/10.1145/3038912.3052653
[38] Kun Tu, Jian Li, Don Towsley, Dave Braines, and Liam D. Turner. 2019. Gl2vec: Learning feature representation using graphlets for directed networks. In *Proceedings of the IEEE/ACM ASONAM*. 216–221. DOI: https://doi.org/10.1145/3341161.3342908
[39] Johan Ugander, Lars Backstrom, and Jon Kleinberg. 2013. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *Proceedings of the WWW*. 1307–1318. DOI: https://doi.org/10.1145/2488388.2488502
[40] M. D. Vose. 1991. A linear algorithm for generating random numbers with a given distribution. *IEEE Trans. Softw. Eng.* 17, 9 (1991), 972–975. DOI: https://doi.org/10.1109/32.92917
[41] Pinghui Wang, John C. S. Lui, Bruno Ribeiro, Don Towsley, Junzhou Zhao, and Xiaohong Guan. 2014. Efficiently estimating motif statistics of large networks. *ACM Trans. Knowl. Discov. Data* 9, 2 (2014). DOI: https://doi.org/10.1145/2629564

Received 10 October 2021; revised 3 May 2023; accepted 3 May 2023