An Efficient Streaming Algorithm for Approximating Graphlet Distributions

MARCO BRESSAN*, University of Milan, Italy
T-H. HUBERT CHAN, The University of Hong Kong, China
QIPENG KUANG, The University of Hong Kong, China
MAURO SOZIO, LUISS University, Italy

In recent years, the problem of computing the frequencies of the induced k-vertex subgraphs of a graph, or *k-graphlets*, has become central. One approach for this problem is to sample k-graphlets randomly. Classic algorithms for k-graphlet sampling require loading the entire graph into main memory, making them impractical for massive graphs. To bypass this limitation, Bourreau et al. (NeurIPS 2024) introduced a *streaming* algorithm that through nontrivial techniques makes only $O(\log n)$ passes using $O(n \log n)$ memory. In this work we break their $O(\log n)$-pass bound by giving an algorithm that, for any fixed $c > 0$, makes $O(1/c)$ passes using $\tilde{O}(n^{1+c})$ memory. As a consequence of their lower bound, our algorithm is optimal up to a factor of $\tilde{O}(n^c)$ in the memory usage. We use this sampling algorithm to obtain an efficient method of approximating k-graphlet distributions. Experiments on real-world and synthetic graphs show that our algorithm is always at least as good as the one of Bourreau et al., and outperforms it by orders of magnitude on mildly dense graphs.

CCS Concepts: • **Theory of computation → Streaming, sublinear and near linear time algorithms**; • **Information systems** → *Data mining*.

Additional Key Words and Phrases: Graphlet Distribution, Streaming

ACM Reference Format:
Marco Bressan, T-H. Hubert Chan, Qipeng Kuang, and Mauro Sozio. 2026. An Efficient Streaming Algorithm for Approximating Graphlet Distributions. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 133 (June 2026), 26 pages. https://doi.org/10.1145/3802010

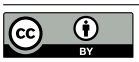
1 Introduction

A *k-graphlet* of a simple graph G is a connected, induced subgraph formed by exactly k vertices. Two k-vertex subsets are *equivalent* if their induced subgraphs are isomorphic. The so-called *k-graphlet distribution* of G is the frequency vector that provides the relative size of each such equivalence class. It is known that the k-graphlet distribution captures important structural information [34] that is relevant in many real-world applications, such as graph classification [43], graph kernels [40], graph neural networks [37], and federated learning [23]. In this work, we study the problem of approximating the k-graphlet distribution when the input graph cannot be stored in memory.

A first major distinction must be made between *exact* and *approximate* algorithms. A trivial exact algorithm is the one that enumerates and checks all k-vertex subsets of the input graph. This algorithm is obviously inefficient, and there is strong evidence that one cannot perform substantially better (see classic lower bounds from parameterized complexity, such as [18] or [26]).

*Authors appear in alphabetical order.

Authors' Contact Information: Marco Bressan, University of Milan, Milan, Italy, marco.bressan@unimi.it; T-H. Hubert Chan, The University of Hong Kong, Hong Kong, China, hubert@cs.hku.hk; Qipeng Kuang, The University of Hong Kong, Hong Kong, China, kuangqipeng@connect.hku.hk; Mauro Sozio, LUISS University, Rome, Italy, msozio@luiss.it.



ACM 2836-6573/2026/6-ART133
https://doi.org/10.1145/3802010

Our focus is instead on approximate algorithms, which (roughly speaking) compute the k-graphlet distribution within some small additive error $\alpha > 0$. It is clear that this problem can be reduced, in the obvious way, to sampling k-graphlets randomly from the underlying graph. Many algorithms for sampling k-graphlets have been proposed, broadly divided in those based on random walks [1, 7, 19, 24, 38, 44] and those based on color coding [13–16]. Eventually, [9, 12] described Ugs, an algorithm that preprocesses the graph with n vertices and m edges in time $O(nk^2 \log k + m)$ and then produces independent uniform k-graphlets in $k^{O(k)} \log n$ time per sample, yielding an efficient algorithm to approximate the k-graphlet distribution.

Although Ugs allows one to sample k-graphlets efficiently, it does so only in the standard RAM model, where the input must fit entirely in memory. Unfortunately, in practice the graph is often too large for the machine's memory; this is commonplace for real-world graphs, such as those representing social networks, the Web, the human brain and many others. In these cases, Ugs simply does not work, and the same holds for the other algorithms listed above.

To bypass these limitations, [8] has recently initiated the study of the k-graphlet sampling problem in the *streaming model*. In this model, the input edges can only be accessed sequentially in an arbitrary order, while using an amount of memory sublinear in the size of the input graph. Streaming algorithms process the graph over multiple *passes*, with each pass involving the processing of all edges in the input graph. Studying the k-graphlet distribution problem in the streaming model avoids the obstacle of storing the graph in memory, and turns the optimization target to the number of passes versus the memory usage.

The bottleneck of the algorithm provided by [8] is in the preprocessing phase, called ApproxDD, which computes an approximate *degree-dominating* order of vertices—a vertex ordering where each vertex has approximately the highest degree in the subgraph induced by its suffix. This preprocessing phase uses $O(n \log n)$ memory and $O(\log n)$ passes or, alternatively, $O(n)$ memory and $O(\log n \cdot \log \log n)$ passes. (This regime, where the memory allowed is linear or near-linear in n, is called the *semi-streaming model*). Unfortunately, ApproxDD is still inefficient for at least two reasons. The first reason is that it requires $O(\log n)$ passes. It is not clear that this is optimal; the very lower bounds from [8] only rule out $O(1)$ passes with $o(n)$ memory, but not, say, $O(1)$ or $O(\log \log n)$ passes with $O(n)$ or $O(n \log n)$ memory. The second reason is that, as shown by our experiments, ApproxDD basically breaks down on mildly dense graphs: it takes thousands of passes, or even fails to run, even with a memory of (say) 10% the size of the input graph. In this work we aim at bypassing these limitations, by seeking an algorithm that runs in $O(1)$ passes and guarantees a good performance regardless of the density of the graph.

1.1 Contributions

We develop a streaming algorithm to approximate the k-graphlet distribution, with the following guarantees:

Theorem 1.1. *There is a two-phase semi-streaming algorithm with the following guarantees. Given $c > 0$, a simple graph G, a positive integer $k \geq 3$, an error parameter $\alpha \in (0, 1)$ and a failure probability $\delta \in (0, 1)$, the algorithm returns a distribution $\hat{\boldsymbol{p}}$ such that:*

- *The first phase ("preprocessing") uses $O(1/c)$ passes and $\tilde{O}\left(n^{1+c}\right)$ bits of memory[1], with probability at least $1 - \frac{1}{n}$.*
- *The second phase ("sampling") always uses $O\left(\frac{k^{O(k)}}{\alpha^4 M} \ln \frac{1}{\delta}\right)$ passes and M bits of memory.*
- *The L_∞ distance between $\hat{\boldsymbol{p}}$ and the k-graphlet distribution of G is at most α, with probability at least $1 - \delta$.*

[1]The standard $\tilde{O}(\cdot)$ notation hides $\operatorname{poly}\log(\cdot)$ factors.

Our algorithm follows the STREAM-UGS framework introduced by Bourreau et al. [8]:

Preprocessing Phase

(a) Compute an *approximate degree-dominating order* (*DD order*, see Definition 2.1), a special total order of the vertices that enables fast uniform graphlet sampling [9, 12];
(b) Compute the initial probabilities;

Sampling Phase

(c) Repeatedly sample k-graphlets.
(d) Either (i) execute probabilistic rejection (in [8]), or (ii) obtain the k-graphlet distribution (in this work).

On the theoretical side, the main contribution behind Theorem 1.1 is a new streaming algorithm for computing a DD order in Step (a). Unlike the algorithm of [8], which intrinsically requires $O(\log n)$ passes because of its "halving degree" strategy, our algorithm requires only $O(1/c)$ passes[2], at the cost of higher memory usage. This is thanks to a strategy that, speaking broadly, consists in sampling enough edges from the input graph so as to make, in a single pass, much more progress than the algorithm of [8]. We remark that, by a lower bound of [8], no algorithm using $O(1)$ passes and $o(n)$ memory can even decide if the input graph contains some k-graphlet, for a fixed $k \geq 2$. Since computing the graphlet distribution implies solving such a decision problem, the memory used by our graphlet distribution algorithm is optimal up to a factor of $\tilde{O}(n^c)$.

Our algorithm also differs from the corresponding algorithm in [8] in Step (d): in particular, our work employs the Horvitz-Thompson-based algorithm [25] to approximate the k-graphlet distribution within L_∞ distance α, whereas [8] focuses on uniform k-graphlet sampling using probabilistic rejection.

On the practical side, although our memory usage goes with n^{1+c} rather than just n, it turns out that picking c small enough makes our algorithm very competitive compared to [8]. In our experiments, by choosing $c = 0.1$, our algorithm makes around 20 preprocessing passes on most datasets, and yet its memory usage is always within one and a half times that of [8]. In fact, on datasets that are only *mildly* dense, our algorithm drastically outperforms the one of [8]. For example, on a dataset with $n \approx 2 \times 10^5$ vertices and average degree $\approx 500 = 0.0025n$, both algorithms use around 20MB of memory, but ours makes less than 20 passes and the one of [8] more than 3000. Our algorithm thus performs well both in theory and in practice, and is scalable and predictable regardless of the structure of the input graph.

1.2 Related Work

Counting and sampling triangles can be considered as a special case of computing the k-graphlet distribution as $k = 3$. Most streaming algorithms for this task in the literature, such as [2, 4, 5, 27, 31, 36, 41], do not generalize or give guarantees for $k > 3$. Streaming algorithms for counting or sampling k-graphlets for a generic k exist, such as [28] and [42]. Notice that their streaming setting is different from ours, as they consider only *single*-pass algorithms, they ask to maintain a good approximation throughout the entire stream, and they have stricter memory requirements. The downside is that both works give guarantees only for specific graphlets.

Many algorithms exist for counting and sampling k-graphlets, or estimating the k-graphlet distribution, in the standard computation model [1, 7, 13–16, 19, 24, 32, 35, 38, 44]. All those

[2]Note that the guarantees of [8] are probabilistic for number of passes and deterministic for the memory, while our guarantees are probabilistic for both.

algorithms have poor memory consumption, as they need to store the whole graph in memory. Among those algorithms, one that is still relatively efficient is Motivo [16], which is based on the color-coding technique. Motivo has formal approximation guarantees, and in practice it can approximate well the k-graphlet distribution on large graphs in a matter of minutes or hours (depending on k) when running on a workstation. In this work we use it to compute the ground-truth k-graphlet distributions for our experiments.

Topological orders play an important role in subgraph mining problems. For example, the degeneracy (or core) order, one of the most commonly used orders, enables efficient subgraph-counting algorithms [6, 11, 20, 33]. There are also streaming algorithms involving computing topological orders [3, 8, 39]. We remark that computing the core order is technically different from computing the *degree-dominating order* that is used in this work. Indeed, while a core order is obtained by repeatedly removing a vertex of minimum degree, a degree-dominating order is obtained by removing one of maximum degree, and it is easy to find examples where the two orders are not obviously related in any way. Therefore, existing streaming algorithms for core orders [3, 39] cannot be employed here.

We face a technical challenge similar to the one of [22], where the authors approximate the k-core decomposition in parallel: keeping accurate estimates of vertex degrees while iteratively removing vertices in a large graph. Maintaining the degrees by scanning the entire edge list after each removal would require too many passes. Their solution is to work on a sub-sampled graph, produced by sampling each edge with an appropriate probability p. On the one hand, by standard concentration results, the degrees in the sub-sampled graph provide a good estimate of the true degree. On the other hand, only a small fraction of edges is sampled, thus one may fit the memory requirements of the streaming algorithm.

We use a Horvitz–Thompson estimator [25] to estimate the distribution of a set of objects when only a non-uniform sampling of the objects can be accessed. This estimator helps us approximate the k-graphlet distribution without sampling k-graphlets uniformly at random; see also [16].

1.3 Organization

We define notation and key notions in Section 2. The new algorithm for computing the DD order is presented in Section 3. The algorithm for estimating the k-graphlet distribution is then completed in Section 4. Finally, the experiments to evaluate our algorithm and compare with existing work are shown in Section 5.

2 Preliminaries

This section introduces basic notation and the main technical concepts.

2.1 Notation and Computational Model

Let $G = (V, E)$ be a simple graph and let $n = |V|$. For a subset $U \subseteq V$ of vertices, $G[U]$ denotes the subgraph of G induced by U, and $G - U$ denotes the subgraph $G[V \setminus U]$. For a total order $\prec$ over V, we let $G(v)$ be the subgraph $G[\{u : v \preceq u\}]$. For $u \in V$, we denote by d_u the degree of u in G; if H is a subgraph of G and $u \in V(H)$, then we denote by $d(u|H)$ the degree of u in H. This applies in particular to $H = G(v)$, so we may write $d(u|G(v))$. We denote by Δ the maximum degree of G.

The smallest value for which the problem is non-trivial is $k \geq 3$. A *k-graphlet* of G is an induced, connected, k-vertex subgraph of G. We denote by $g_1, \cdots, g_{m_k}$, the representatives of the isomorphism classes of connected graphs on k vertices, in any order. Thus, every k-graphlet of G is isomorphic to precisely one g_i. For each $i = 1, \ldots, m_k$ let l_i be the number of induced subgraphs of

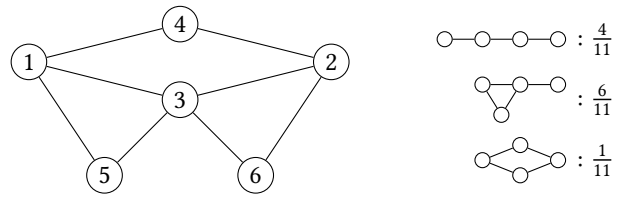


Fig. 1. A toy graph (left) and its 4-graphlet distribution (right). 4-graphlets with probability 0 are not shown.

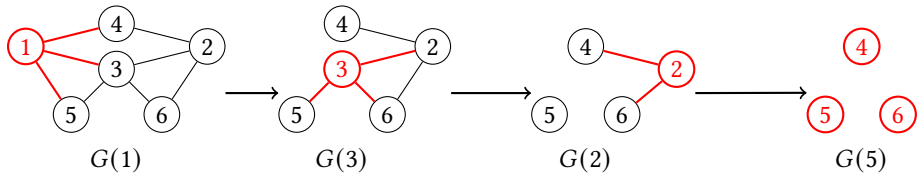


Fig. 2. Computation of a $\frac{1}{1.5}$-DD ordering of a toy graph, by iterative removal of a high-degree vertex. The resulting $\frac{1}{1.5}$-DD ordering is $1 \prec 3 \prec 2 \prec 5 \prec 6 \prec 4$.

G isomorphic to g_i, and let $L = \sum_{i=1}^{m_k} l_i$. The *k-graphlet distribution* of G is the vector

$$\boldsymbol{\mu}_k \triangleq (\mu_1, \cdots, \mu_{m_k})$$

where $\mu_i = \frac{l_i}{L}$ for each i. Our goal is to estimate $\boldsymbol{\mu}_k$. An example of the k-graphlet distribution is presented in Figure 1.

A notion central to this approach is the *degree-dominating* ordering (DD-ordering) introduced in [10], as well as its approximate version from [8].

Definition 2.1 ((Approximate) DD-ordering). Given a simple graph $G = (V, E)$, a total order $\prec$ over V is degree-dominating if, for each $v \in V$, the maximum degree in $G(v)$ is achieved by v itself; in other words, $d(v|G(v)) \geq d(u|G(v))$ for all $u \succeq v$. For $\vartheta > 0$, a *ϑ-DD order* $\prec$ is a total order over V such that, for all $v \in V$ with $d(v|G(v)) \geq 1$ and all $u \succeq v$, it holds that $d(v|G(v)) \geq \vartheta \cdot d(u|G(v))$.

We often use $\vartheta = \frac{1}{1+\epsilon}$ for some $\epsilon > 0$. An example of DD-ordering is shown in Figure 2. A possible $\frac{1}{1.5}$-DD order of the graph is $1 \prec 3 \prec 2 \prec 5 \prec 6 \prec 4$. Note that for the first position of the order, we have $d(1|G(1)) = 3$ and $\max_{u \geq 1} d(u|G(1)) = d(3|G(1)) = 4$, which satisfies $d(1|G(1)) \geq \frac{1}{1.5} \cdot \max_{u \geq 1} d(u|G(1))$.

We consider the *semi-streaming* model of [21]. In this model, the memory consists of $M = \tilde{O}(n^{1+c})$ words of $O(\log n)$ bits for some constant $c > 0$ (for example, $c = 0.1$). The algorithm can access the graph by scanning its edges, sequentially, in what is called a *pass*. The order in which edges appear in each pass is arbitrary (that is, decided by an adversary), and the algorithm can make any number of passes. We assume $m = \Omega(M)$ to avoid trivialities. All computation will be in the standard RAM model.

2.2 Uniform Graphlet Sampling

Our work builds on STREAM-UGS, the streaming algorithm of Bourreau et al. [8]. STREAM-UGS consists of a preprocessing phase and a sampling phase. The preprocessing phase is run once, and computes a ϑ-DD order $\prec$, where $\vartheta = \frac{1}{1+\epsilon}$ for some desired $\epsilon > 0$, together with an initial distribution $\boldsymbol{p}$ over V. With $\prec$ and $\boldsymbol{p}$ one can then implement a k-pass sampling routine ensuring

that any given k-graphlet of G has the same sampling probability except for a multiplicative factor of $((1+\epsilon)k)^{O(k)}$. Using rejection sampling one can remove that factor and get truly uniform sampling, with an overhead of $((1+\epsilon)k)^{O(k)}$ expected trials per sample. The crucial part is therefore computing $\prec$ and $\boldsymbol{p}$ with as few passes as possible.

Computing the ϑ-DD order. As shown in [8, Theorem 3.1], one can compute $\prec$ with high probability in $O\left(\frac{\log n}{\epsilon^2}\log\frac{\log n}{\epsilon}\right)$ passes by using memory $O(n)$, or in $O\left(\frac{\log n}{\epsilon^2}\right)$ passes by using memory $O\left(\frac{n\log n}{\epsilon}\right)$. The idea is to select a subset S of vertices that have degree at least $\frac{\Delta}{1+\epsilon}$, where Δ is the maximum degree of G. This can be done with high probability with a constant number of passes. The set S is then appended to $\prec$ in arbitrary order, and S is removed from G (meaning that *the algorithm* keeps track, in some way, of the subgraph obtained by removing S). Therefore, every $O(1)$ passes the maximum degree of G decreases by a factor $\frac{1}{1+\epsilon}$. As a consequence, the algorithm terminates within $\log_{\frac{1}{1+\epsilon}} n$ passes.

Initial Distribution. Given an $\frac{1}{1+\epsilon}$-DD order, let N_v be the number of k-graphlets in $G(v)$ containing v. Computing $d(v|G(v))$ and deciding whether $N_v > 0$ for all vertices in V can be done in 1 pass and $O(kn)$ memory by [8, Lemma 3.3]. After that, the *normalization parameter*

$$Z = \sum_{v\in V,\ N_v>0} d(v|G(v))^{k-1}$$

as well as the *initial distribution* $\boldsymbol{p} = \{p(v)|v\in V\}$ such that

$$p(v) = \begin{cases} \frac{d(v|G(v))^{k-1}}{Z}, & N_v > 0, \\ 0, & N_v = 0 \end{cases}$$

can be computed without any pass. This subroutine is formally described in Algorithm 1. The initial distribution will be used later to select the starting vertex and compute the relevant sampling probability in the sampling phase.

Algorithm 1: Initial Distribution [8, Algorithm 2]

Function InitDistrib *($G, \prec$)*
  Compute $d(v|G(v))$ and decide whether $N(v) > 0$ for each $v \in V$
  $Z \leftarrow \sum_{v\in V,\ N_v>0} d(v|G(v))^{k-1}$
  $p(v) \leftarrow \begin{cases} \frac{d(v|G(v))^{k-1}}{Z}, & N_v > 0, \\ 0, & N_v = 0 \end{cases}$
  return $\boldsymbol{p} = \{p(v)|v\in V\}$

Algorithm 2: Sampling A Graphlet [8, Algorithm 3]

Function SampleGraphlet *($G, \prec, k, \boldsymbol{p}$)*
  Sample v from the distribution $\boldsymbol{p}$
  $S \leftarrow \{v\}$
  for $i \leftarrow 1$ **to** $k-1$ **do**
    Uniformly sample an edge (x, y) in $G(v)$ such that $x \in S$ and $y \notin S$
    $S \leftarrow S \cup \{y\}$
  return S

Rejection Sampling. At a high level, we first sample a k-graphlet of G and then accept it with some probability so that the probability of each k-graphlet being sampled and accepted is the same. The first sampling step goes as follows. We sample a vertex v according to the initial distribution $\boldsymbol{p} = \{p(v)|v\in V\}$ and let $S = \{v\}$. Then we "grow" S for $k-1$ times, each time by selecting uniformly at random an edge in $G(v)$ with exactly one endpoint in S, and adding the other endpoint to S. In this way, we obtain a k-graphlet S with probability $p(S) = p(v)\cdot p(S|v)$, where $p(S|v)$ is the probability of obtaining S from v. All those probabilities can be computed efficiently. This sampling algorithm is formally described in Algorithm 2.

To achieve uniform sampling, let Γ be some constant such that $0 < \Gamma \leq \min_S p(S)$, where S ranges over all k-tuples of vertices such that $p(S) > 0$. We then accept S with probability $\frac{\Gamma}{p(S)}$, otherwise reject S and sample again. By [8, Lemma C.1], with high constant probability (say 0.99), after $(k(1+\epsilon))^{O(k)}$ trials of sampling we accept a k-graphlet, and that k-graphlet is drawn from the uniform distribution.

Parallel sampling. As Algorithm 2 uses memory $O(k^2)$, by using memory M one can run $\Theta\left(\frac{M}{k^2}\right)$ instances in parallel, and produce with high probability $\Theta\left(\frac{M}{(k(1+\epsilon))^{O(k)}}\right)$ independent uniformly random k-graphlets. Moreover, [8, Theorem 3.4] gives a trade-off between running time and number of passes: the parallel sampling can be achieved either in k passes and $O(M|E|)$ time, or in $2k-1$ passes and $O(M2^k + k|E|\log n)$ time.

2.3 Concentration Inequalities

Let $B(m,p)$ denote the binomial distribution with parameters m, p. We use the following Chernoff-style bounds in our analysis.

Fact 2.1 (Chernoff Bound Variant I). *Let $X \sim B(m,p)$ and $0 < \epsilon < 1$. Then, we have the following:*

- *If $m \geq N$, $\mathbf{Pr}\left[X \leq (1-\epsilon)Np\right] \leq \exp\left(-\frac{1}{3}\epsilon^2 Np\right)$.*
- *If $m \leq N$, $\mathbf{Pr}\left[X \geq (1+\epsilon)Np\right] \leq \exp\left(-\frac{1}{3}\epsilon^2 Np\right)$.*

Fact 2.1 follows from stochastic dominance together with another variant of the Chernoff Bound.

Lemma 2.2 (Stochastic Dominance). *Let $n \leq m$, let $X \sim B(n,p)$ and $Y \sim B(m,p)$. For any c, $\mathbf{Pr}\left[X > c\right] \leq \mathbf{Pr}\left[Y > c\right]$.*

Proof. Let $x_1, \cdots, x_m$ be Bernoulli variables with positive probability p. Let $A = \sum_{i=1}^{n} x_i$, $B = \sum_{i=n+1}^{m} x_i$, and $C = A + B$. Thus A and X have the same distribution, and C and Y have the same distribution. Now observe that, for any c,

$$\mathbf{Pr}\left[X > c\right] = \mathbf{Pr}\left[A > c\right] \leq \mathbf{Pr}\left[C > c\right] = \mathbf{Pr}\left[Y > c\right]. \qquad \square$$

Fact 2.2 (Chernoff Bound Variant II). *Let D be the distribution given by $\mathbf{Pr}\left[X = r_i\right] = p_i$ for $i = 1, \cdots, m$ where $\sum_{i=1}^{m} p_i = 1$. A variable following D has expectation $E = \sum_{i=1}^{m} p_i r_i$. Let $X_1, \cdots, X_T$ be i.i.d. variables following D. For any $0 < \epsilon < 1$, we have the following inequality:*

$$\mathbf{Pr}\left[\left|\frac{\sum_{i=1}^{T} X_i}{T} - E\right| \geq \epsilon E\right] \leq 2\exp\left(-\frac{\epsilon^2 TE}{3\max_{i=1}^{m} r_i}\right).$$

We also use Hoeffding's inequality.

Fact 2.3 (Hoeffding's inequality). *Let D be the distribution given by $\mathbf{Pr}\left[X = r_i\right] = p_i$ for $i = 1, \cdots, m$ where $\sum_{i=1}^{m} p_i = 1$. A variable following D has expectation $E = \sum_{i=1}^{m} p_i r_i$. Let $X_1, \cdots, X_T$ be i.i.d. variables following D. For any $0 < \alpha < 1$, we have the following inequality:*

$$\mathbf{Pr}\left[\left|\frac{\sum_{i=1}^{T} X_i}{T} - E\right| \geq \alpha\right] \leq 2\exp\left(-\frac{2T\alpha^2}{\left(\max_{i=1}^{m} r_i\right)^2}\right).$$

3 Improved Algorithms to Compute the Degree-Dominating Order

As recalled above, the sampling phase of the algorithm of [8] requires just k or $2k-1$ passes, depending on the implementation. The preprocessing phase is the one that dominates the streaming complexity, with $O(\log n)$ passes or $O(\log n \log\log n)$ passes depending on the desired memory

usage. The idea of [8], described in Section 2.2, is to use every pass to detect and remove the subset of vertices with degree at least $\frac{\Delta}{1+\epsilon}$, where Δ is the current maximum degree of the graph. This leads the maximum degree of G to decrease by a multiplicative factor of $(1+\epsilon)$ at each pass, making the algorithm terminate within $O(\log_{1+\epsilon} n)$ passes, and at the same time the removal sequence yields precisely a $\frac{1}{1+\epsilon}$-DD ordering.

Our key challenge is to break through the $O(\log n)$ passes bound. In order to do that we must, at each pass, make more progress than just removing the vertices with degree at least $\frac{\Delta}{1+\epsilon}$. Note that, whenever *any* vertex is removed, the degrees of the remaining vertices can change. Therefore there is a delicate tradeoff between (a) how many vertices one removes, and (b) how good the current degree estimates still are. In the approach of [8] described above, this tradeoff still works, because after each vertex-removal pass the degrees are recomputed exactly using one additional pass. In our case, however, we can make such a pass only $O(1/c)$ times.

Using similar ideas to [22], we modify the algorithm of [8] by computing, in a single pass, several approximations of the input graph, each one obtained by sub-sampling edges with a different probability. We then employ these sub-sampled graphs to provide the degree estimates required by several subsequent vertex-removal passes of [8]. In this way we are essentially "accelerating" the algorithm of [8] by performing several passes at once.

For the sake of readability, we start by introducing in Section 3.1 a "warm-up" algorithm which still uses $O\left(\frac{\log n}{\epsilon}\right)$ passes and $O\left(\frac{n}{\epsilon^2}\log\frac{n}{\epsilon\delta}\right)$ memory. Then, in Section 3.2, we show how for any desired $c > 0$ we can bring the passes down to $O\left(\frac{1}{c}\right)$ passes at the price of using $O\left(\frac{n^{1+c}}{\epsilon^3}\log\frac{n}{\epsilon\delta}\right)$ memory.

3.1 A Simple Approach Using $O(\log n)$ Passes

The idea of the algorithm is as follows. We sample a directed graph $\tilde{G}$ by adding each direction of each edge of G into $\tilde{G}$ with some probability p. By picking an appropriate p, we have that: (1) the number of edges in $\tilde{G}$ is small with high probability; and (2) the out-going degree of each vertex is a good estimate of its degree. As we sample the two directions of the original edge independently, the estimations of vertex degrees are independent, even if vertices are removed sequentially.

We state a warm-up algorithm in Algorithm 3. Let U be the set of vertices of G, ϵ be the parameter of the DD order, and $\prec$ be an empty list. Define a probability p and we sample $\tilde{G}$ from $G[U]$ as described above using one pass such that with high probability, $\tilde{G}$ contains $\tilde{O}\left(\frac{n}{\epsilon^2}\right)$ edges. The degree of a vertex v in the subgraph $G[H]$ for any $H \subseteq U$ is then estimated by $\frac{1}{p}$ times the out-degree of v in $\tilde{G}[H]$. Then we execute a subroutine Select-Large to append vertices with large estimated degree to $\prec$ and remove them from U, where a vertex is said to have large estimated degree if its estimated degree is at least $\frac{\Delta}{1+\Theta(\epsilon)}$ for the maximum degree Δ of the current graph G (or its estimation). Vertices are scanned in an arbitrary order. When a vertex v is considered, if its estimated degree in the current remaining graph is at least $\frac{\Delta}{1+\Theta(\epsilon)}$, then we are supposed to add v to $\prec$ and remove v together with its incident edges. In this way, we claim that after an execution of Select-Large, only vertices with actual degree larger than $\frac{\Delta}{1+\epsilon}$ will be appended to $\prec$ and removed from U and thus $\prec$ is always a $\frac{1}{1+\epsilon}$-DD order. Moreover, the remaining vertices have actual degree at most $\frac{\Delta}{1+\Theta(\epsilon)}$ so that the maximum degree of G decreases by a factor $\frac{1}{1+\Theta(\epsilon)}$.

Theorem 3.1. *With probability at least $1-\delta$, Algorithm 3 returns a $\frac{1}{1+\epsilon}$-DD order using $O\left(\frac{\log n}{\epsilon}\right)$ passes and $O\left(\frac{n}{\epsilon^2}\log\frac{n}{\epsilon\delta}\right)$ words of memory.*

Algorithm 3: Approximate DD Order via Edge Sampling I

Function ApproxDD-Warmup *($G = (V, E), \epsilon, \delta$)*
- $\prec \leftarrow$ empty list
- $U \leftarrow V$
- $\hat{\epsilon} \leftarrow \frac{\epsilon}{4+3\epsilon}$
- $\Delta \leftarrow n - 1$ ▷ Max degree estimate; alternatively, computed with one pass.
- $K \leftarrow 1 + \epsilon$ ▷ A removed vertex should have current degree at least $\frac{\Delta}{K}$
- $T \leftarrow \log_{1+\frac{\epsilon}{2}} n$ ▷ Estimated number of iterations in while loop
- **while** $U \neq \emptyset$ **do**
  - Using one pass on the edge list, construct a directed graph $\tilde{G}$ on U by sampling each direction of each edge in $G[U]$ independently with probability p: $p = \min\left\{\frac{3K}{\hat{\epsilon}^2\Delta} \ln \frac{2nT}{\delta}, 1\right\}$
  - $(U, \prec) \leftarrow$ Select-Large$(U, \tilde{G}, p, \Delta, \prec)$
  - $\Delta \leftarrow \frac{\Delta}{1+\frac{\epsilon}{2}}$
- **return** $\prec$

Function Select-Large *($U, \tilde{G}, p, \Delta, \prec$)*
- $\alpha \leftarrow \frac{3\epsilon}{4}$
- $H \leftarrow U$
- **foreach** $v \in U$ *in an arbitrary order* **do**
  - $\hat{d}(v) \leftarrow \frac{1}{p} \times$ (Out-degree of v in $\tilde{G}[H]$) ▷ Unbiased degree estimator in $G[H]$
  - **if** $\hat{d}(v) \geq \frac{\Delta}{1+\alpha}$ **then**
    - Append v to $\prec$
    - $H \leftarrow H \setminus \{v\}$ ▷ Remove vertex v
- **return** $(H, \prec)$

Proof. **Proof structure.** The proof separates the *probabilistic* part from the *deterministic* part. In the probabilistic part, we first define a small set of *bad events* that, if they occur, could violate either (i) the memory bound, (ii) the pass bound, or (iii) correctness of the returned order. We then bound the probability of each bad event using Chernoff bounds (Fact 2.1) and take a union bound over all vertices and all iterations. In the deterministic part, we condition on the event that none of these bad events occur in the first T iterations, and prove deterministically that the algorithm (a) makes geometric progress in Δ and thus terminates in T iterations, (b) outputs a $\frac{1}{1+\epsilon}$-DD order, and (c) never exceeds the claimed space.

Bad Events in One Iteration. Suppose the maximum degree of $G[U]$ at the beginning of Select-Large is at most Δ. To simplify the notation, for a vertex $v \in U$ at the moment it is considered in Select-Large, we use d_v to denote its true degree in the remaining graph $G[H]$ and $\hat{d}_v$ to denote the unbiased estimation of d_v, which has the distribution $\frac{1}{p} \times B(d_v, p)$.

The algorithm makes decisions using only the sampled graph $\tilde{G}$ and the estimates $\hat{d}_v$. Thus, there are two distinct failure modes to control: (1) $\tilde{G}$ can be too large (breaking the memory budget), and (2) $\hat{d}_v$ can deviate enough to change a peel/keep decision (breaking progress or correctness). Accordingly, we consider the following bad events.

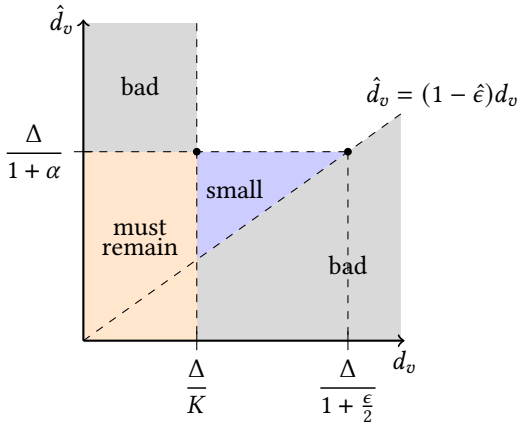


Fig. 3. The relation between the true degree d_v and the estimated degree $\hat{d}_v$. The area of bad events is filled in gray. The blue triangle area indicates that if v is not removed in Select-Large, we have $d_v \le \frac{\Delta}{1+\frac{\epsilon}{2}}$. The orange rectangle area indicates that if $d_v < \frac{\Delta}{K}$, v must remain after Select-Large.

Type I: The sampled directed graph $\tilde{G}$ has more than $(1+\hat{\epsilon}) \cdot n\Delta p = \Theta\left(\frac{n}{\epsilon^2} \log \frac{n}{\epsilon\delta}\right)$ edges. By Fact 2.1, this happens with probability at most $\exp\left(-\frac{1}{3}\hat{\epsilon}^2 n\Delta p\right) \le \frac{\delta}{2T}$.
Intuition. The number of sampled directed edges is a sum of independent Bernoulli variables. At the start of the iteration, the total number of directed edges in $G[U]$ is at most $n\Delta$, and each is kept with probability p, so the expectation is at most $n\Delta p$. A multiplicative Chernoff bound then shows that exceeding $(1+\hat{\epsilon})n\Delta p$ has exponentially small probability, which is why choosing p so that $n\Delta p = \Theta\left(\frac{n}{\epsilon^2} \log \frac{n}{\epsilon\delta}\right)$ suffices for a union bound over T iterations.

Type II: For a vertex v, if $d_v \ge \frac{\Delta}{K}$, the bad event is $\hat{d}_v < (1-\hat{\epsilon}) \cdot d_v$; if $d_v < \frac{\Delta}{K}$, the bad event is $\hat{d}_v \ge (1+\hat{\epsilon}) \cdot \frac{\Delta}{K}$. By Fact 2.1, the bad event for each vertex is at most $\exp\left(-\frac{1}{3}\hat{\epsilon}^2 \frac{\Delta p}{K}\right) \le \frac{\delta}{2nT}$.
Intuition. We split Type II into two cases because the harmful direction of estimation error depends on the true degree. If d_v is large ($d_v \ge \Delta/K$), then underestimating it could prevent peeling a vertex that should be removed, slowing the decrease of Δ. If d_v is small ($d_v < \Delta/K$), then overestimating it could wrongly peel v, which would violate the DD property. The Chernoff bound is applied to the binomial $B(d_v, p)$, and we upper bound the failure probability uniformly by lower bounding the relevant mean by $(\Delta/K)p$, enabling a union bound over all vertices and all iterations. Figure 3 helps to illustrate the area of type II bad events.

The union of all bad events within one iteration has probability at most $\frac{\delta}{T}$. Therefore, it suffices to show that if no bad event happens within the first T iterations, then the algorithm must terminate and return a correct result within the required memory.

The Number of Passes. We consider one execution of Select-Large. If no type II bad event happens, we argue that if a vertex v is not removed, it must be the case that at the moment it is considered, we have $d_v \le \frac{\Delta}{1+\frac{\epsilon}{2}}$, whose degree may drop even further as other vertices are removed later.

We do not need to worry about the case $d_v < \frac{\Delta}{K} < \frac{\Delta}{1+\frac{\epsilon}{2}}$. For the case $d_v \geq \frac{\Delta}{K}$, since v is not removed, we have $\hat{d}_v < \frac{\Delta}{1+\alpha}$. Moreover, the type II bad event at v does not happen, and so we have $d_v \leq \frac{\hat{d}_v}{1-\hat{\epsilon}} < \frac{\Delta}{(1+\alpha)(1-\hat{\epsilon})} = \frac{\Delta}{1+\frac{\epsilon}{2}}$. This corresponds to the blue triangle area in Figure 3.

Therefore, If no type II bad event happens within the first T iterations, any vertex that survives the scan already has degree at most $\frac{\Delta}{1+\frac{\epsilon}{2}}$ at the moment it is considered. This means that Δ will drop below 1 after $T = \log_{1+\frac{\epsilon}{2}} n = O\left(\frac{\log n}{\epsilon}\right)$ iterations at which point the algorithm must terminate. Since each iteration uses 1 pass to build $\tilde{G}$, the total passes are T.

Correctness. Within one iteration of Select-Large, it suffices to show that only vertices v satisfying $d_v \geq \frac{\Delta}{K}$ could possibly be removed, since for any other vertex u remaining before this iteration, we have $d(u|G(v)) \leq \Delta$ and hence $d(v|G(v)) = d_v \geq \frac{\Delta}{K} \geq \frac{1}{1+\epsilon} d(u|G(v))$.

Since the type II bad event for a vertex v satisfying $d_v < \frac{\Delta}{K}$ does not happen, we must have $\hat{d}_v < (1+\hat{\epsilon}) \cdot \frac{\Delta}{K} = \frac{\Delta}{1+\alpha}$, which means such a vertex v will not be appended to $\prec$ when being considered. This corresponds to the orange rectangle area in Figure 3.

Therefore, if no type II bad event happens within the first T iterations, then $\prec$ is a $\frac{1}{1+\epsilon}$-DD order.

Memory. The memory usage is dominated by the number of edges in the sampled graph $\tilde{G}$ in each iteration. Since the type I bad event does not happen within the first T iterations, we conclude that the number of edges sampled in each iteration is at most $O\left(\frac{n}{\epsilon^2} \log \frac{n}{\epsilon\delta}\right)$.

□

3.2 From $O(\log n)$ to $O(1/c)$ Passes

One idea to decrease the number of passes in Algorithm 3 is to generate and remember several sampled directed graphs during a single pass, at the cost of using more memory. In Algorithm 4, instead of having one pass per iteration, a pass is made every $q = \lfloor \log_{1+\frac{\epsilon}{2}} n^c \rfloor$ iterations for some constant $c > 0$. The sampling probabilities are computed by the estimated maximum degree for each graph assuming its dropping by a factor of $1 + \frac{\epsilon}{2}$. The same subroutine Select-Large is executed on the sampled graphs sequentially. By choosing the appropriate c, we ensure that it decreases the number of passes to $O\left(\frac{1}{c}\right)$, and each pass can lead to a factor n^c decrease in the maximum degree Δ.

Theorem 3.2. *With probability at least $1-\delta$, Algorithm 4 returns a $\frac{1}{1+\epsilon}$-DD order using $O\left(\frac{1}{c}\right)$ passes and $O\left(\frac{n^{1+c}}{\epsilon^3} \log \frac{n}{\epsilon\delta}\right)$ words of memory.*

Proof. Observe that the difference between Algorithms 3 and 4 is that in Algorithm 4, during one pass of the edge lists, random directed graphs from several iterations are sampled and saved.

Correctness and the Number of Passes. The key idea is a coupling argument via the directed edge sampling in each iteration: even though Algorithm 4 samples multiple graphs in one pass, it can be seen as using exactly the same randomness that Algorithm 3 would have used over the corresponding iterations.

Therefore, the two algorithms produce exactly the same order $\prec$, and the correctness follows from Theorem 3.1. As before, given that there is no bad event in the first T iterations, the algorithm must terminate and uses $O(\frac{T}{q}) = O(\frac{1}{c})$ passes.

Memory. Recall that in each pass, we sample and save q directed subgraphs. Assuming that no bad event has occurred in previous passes, the maximum degree of $G[U]$ is at most Δ at the beginning of the current pass. For the i-th graph, the expected number of sampled directed edges is on the order of $n\Delta \cdot p_i$. Hence, in this pass, the expected number of sampled edges in all the q subgraphs is at most:

Algorithm 4: Approximate DD Order via Edge Sampling II

Function ApproxDD-ES $(G = (V, E), \epsilon, \delta, c)$
- $\prec \leftarrow$ empty list; $U \leftarrow V$
- $\hat{\epsilon} \leftarrow \frac{\epsilon}{4+3\epsilon}$
- $\Delta \leftarrow n - 1$ ▷ Max degree estimate; alternatively, computed with one pass.
- $K \leftarrow 1 + \epsilon$
- $T \leftarrow \log_{1+\frac{\epsilon}{2}} n$ ▷ Estimated number of sampled directed graphs
- $q \leftarrow \lfloor \log_{1+\frac{\epsilon}{2}} n^c \rfloor$
- $i \leftarrow q$
- **while** $U \neq \emptyset$ **do**
  - **if** $i = q$ **then**
    - **for** j *from* 0 *to* $q - 1$ **do** ▷ Using one pass on edge list, sample q directed graphs.
      - $\Delta_j \leftarrow \frac{\Delta}{(1+\frac{\epsilon}{2})^j}$
      - $p_j = \min\left\{\frac{3K}{\hat{\epsilon}^2 \Delta_j} \ln \frac{2nT}{\delta}, 1\right\}$
      - Construct a directed graph $\tilde{G}_j$ on U by sampling each direction of each edge in $G[U]$ independently with probability p_j
    - $i \leftarrow 0$
  - $(U, \prec) \leftarrow$ Select-Large$(U, \tilde{G}_i, p_i, \Delta_i, \prec)$ ▷ Same subroutine as in Algorithm 3
  - $i \leftarrow i + 1$
  - **if** $i = q$ **then**
    - $\Delta \leftarrow \frac{\Delta_{q-1}}{1+\frac{\epsilon}{2}}$
- **return** $\prec$

$$\sum_{i=0}^{q-1} n\Delta p_i \leq O\left(\frac{n^{1+c}}{\epsilon^3} \log \frac{n}{\epsilon\delta}\right).$$

To get a high probability statement for memory usage, observe that we are no worse than the situation in Theorem 3.1. First, the expected number of saved edges is larger in each pass, which means Chernoff Bound will give a smaller failure probability for each pass. Second, the number of passes are fewer, which means there are fewer terms in the union bound.

In conclusion, with probability at least $1 - \delta$, all the required properties as stated are satisfied. □

4 Approximating the k-Graphlet Distribution

As stated in Section 2.2, provided a $\frac{1}{1+\epsilon}$-DD order $\prec$ of a graph G, one can compute an initial distribution $\boldsymbol{p}$ and use it to sample a k-graphlet S of G with probability $p(S)$. In this section, we complete the task of approximating the k-graphlet distribution $\boldsymbol{\mu}_k = (\mu_1, \cdots, \mu_{m_k})$ of G based on this k-graphlet sampling.

4.1 Sampling with Counters

The approximation to the k-graphlet distribution is described in Algorithm 5. Using Algorithm 2, a graphlet S can be sampled with some probability $p(S)$ computed by the initial distribution $\boldsymbol{p}$. The

original algorithm in [8] further achieves uniform sampling by accepting the sampling of S with probability $\frac{\Gamma}{p(S)}$ for a constant Γ smaller than every $p(S)$. In our algorithm, we do not reject any sampling, but employ the technique of Horvitz–Thompson estimators. We maintain a collection of counters $c_1, \cdots, c_{m_k}$ for isomorphism classes $1, \cdots, m_k$ respectively. When we sample a graphlet S, recall that there is precisely one integer $r \in [1, m_k]$ such that g_r is isomorphic to S. We then add $\frac{1}{p(S)}$ to c_r. The graphlet distribution is then obtained by normalizing these counters.

Algorithm 5: Approximate k-Graphlet Distribution via Counters (Abstract Description)

```
Function Counter (G, ≺, k, p, T)
    c_1, ··· , c_{m_k} ← 0
    for i ← 1 to T do (sequentially/in parallel)
        S ← SampleGraphlet (G, ≺, k, p)      ▷ Sample a k-graphlet S with probability p(S)
        r ← the index such that S is isomorphic to g_r
        c_r ← c_r + 1/p(S)
    Ĉ ← Σ_{i=1}^{m_k} c_i
    μ̂_i ← c_i/Ĉ for each 1 ≤ i ≤ m_k
    return μ̂_1, ··· , μ̂_{m_k}
```

Recall that given a $\frac{1}{1+\epsilon}$-DD order $\prec$, Z defined in Section 2.2 is the sum of $d(v|G(v))^{k-1}$ over those vertices v with $N_v > 0$. We will use the property that both $\frac{1}{p(S)}$ and the number L of graphlets in G are bounded by Z.

Fact 4.1. *([8, Lemma C.1]) For each k-graphlet S, it holds that*

$$\frac{1}{p(S)} \leq (k-1)!(1+\epsilon)^{k-1} Z.$$

Lemma 4.1. *It holds that*

$$L \geq \frac{Z}{(k-1)^{k-1}}.$$

To prove Lemma 4.1, we need the following lemma from [12].

Fact 4.2. *([12, Lemma 24]) For every $v \in V$ such that $N_v > 0$, we have*

$$N_v \geq \frac{d(v|G(v))^{k-1}}{(k-1)^{k-1}}.$$

Proof of Lemma 4.1. By Fact 4.2, we have

$$L = \sum_{v \in V,\ N_v > 0} N_v \geq \sum_{v \in V,\ N_v > 0} \frac{d(v|G(v))^{k-1}}{(k-1)^{k-1}} = \frac{Z}{(k-1)^{k-1}}.$$

□

We give the analysis of Algorithm 5.

Lemma 4.2. *For any $\frac{1}{1+\epsilon}$-DD order, k and $0 < \epsilon_0, \delta < 1$, let T be a positive integer satisfying*

$$T \geq \frac{3(k-1)!(k-1)^{k-1}(1+\epsilon)^{k-1}}{\epsilon_0^2} \ln \frac{2}{\delta},$$

then Algorithm 5 computes $\hat{C}$ such that $\left|\frac{\hat{C}}{T} - L\right| < \epsilon_0 L$ with probability at least $1 - \delta$.

Proof. Let $X_1, \cdots, X_T$ be the value of $\frac{1}{p(S)}$ of each iteration. Then we have $\hat{C} = \sum_{i=1}^{T} X_i$. In fact, $X_1, \cdots, X_T$ are independent random variables such that X_i takes value $\frac{1}{p(S)}$ with probability $p(S)$ for each k-graphlet S in G. Each X_i is at most $(k-1)!(1+\epsilon)^{k-1}Z$ by Fact 4.1 and has expectation L. By Fact 2.2 and lemma 4.1 we have

$$\mathbf{Pr}\left[\left|\frac{\hat{C}}{T} - L\right| \geq \epsilon_0 L\right] \leq 2\exp\left(-\frac{\epsilon_0^2 LT}{3(k-1)!(1+\epsilon)^{k-1}Z}\right) \leq 2\exp\left(-\frac{\epsilon_0^2 T}{3(k-1)!(k-1)^{k-1}(1+\epsilon)^{k-1}}\right) = \delta.$$

□

Theorem 4.3. *For any $\frac{1}{1+\epsilon}$-DD order, k and $0 < \alpha, \delta < 1$, let T be a positive integer satisfying*

$$T \geq \frac{12((k-1)!)^2(k-1)^{2k-2}(1+\epsilon)^{2k-2}}{\alpha^2(1-\alpha)^2} \ln\frac{4m_k}{\delta},$$

then Algorithm 5 computes $\hat{\mu}_1, \cdots, \hat{\mu}_{m_k}$ such that with probability at least $1-\delta$, $|\hat{\mu}_i - \mu_i| \leq \alpha$ for each $1 \leq i \leq m_k$.

Proof. We want to bound the deviation of each normalized counter $\hat{\mu}_i = c_i/\hat{C}$ from the true proportion $\mu_i = l_i/L$, uniformly over all isomorphism classes $i \in [m_k]$. The main technical issue is that $\hat{\mu}_i$ is a ratio with random denominator $\hat{C}$. To handle this cleanly, we first condition on the event that $\hat{C}$ concentrates around its mean (so the denominator is well-controlled), and then apply a concentration bound to each numerator c_i. Finally we union bound over the m_k classes and combine with the probability that the conditioning event holds.

Assume that the condition $|\frac{\hat{C}}{T} - L| < \frac{\alpha}{2}L$ holds. Then $\hat{C}$ lies in the interval $[(1-\alpha/2)LT,\ (1+\alpha/2)LT]$. For each $1 \leq i \leq m_k$, recall that $\mu_i = \frac{l_i}{L}$ where l_i is the number of k-graphlets in G which are isomorphic to g_i. We have

$$\mathbf{Pr}\left[|\hat{\mu}_i - \mu_i| \geq \alpha\right] = \mathbf{Pr}\left[\left|\frac{c_i}{\hat{C}} - \frac{l_i}{L}\right| \geq \alpha\right] \leq \mathbf{Pr}\left[\frac{c_i}{\left(1-\frac{\alpha}{2}\right)LT} - \frac{l_i}{L} \geq \alpha\right] + \mathbf{Pr}\left[\frac{l_i}{L} - \frac{c_i}{\left(1+\frac{\alpha}{2}\right)LT} \geq \alpha\right]. \tag{1}$$

We analyze $\mathbf{Pr}\left[\frac{c_i}{\left(1-\frac{\alpha}{2}\right)LT} - \frac{l_i}{L} \geq \alpha\right]$, and the other side is similar. We rewrite the deviation in terms of $\frac{c_i}{T}$:

$$\begin{aligned}
\mathbf{Pr}\left[\frac{c_i}{\left(1-\frac{\alpha}{2}\right)LT} - \frac{l_i}{L} \geq \alpha\right] &= \mathbf{Pr}\left[\frac{c_i}{T} - \left(1-\frac{\alpha}{2}\right)l_i \geq \alpha\left(1-\frac{\alpha}{2}\right)L\right] \\
&= \mathbf{Pr}\left[\frac{c_i}{T} - l_i \geq \alpha\left(1-\frac{\alpha}{2}\right)L - \frac{\alpha}{2}l_i\right] \\
&\leq \mathbf{Pr}\left[\frac{c_i}{T} - l_i \geq \frac{\alpha}{2}(1-\alpha)L\right],
\end{aligned}$$

where the last inequality is due to $l_i \leq L$. Similarly to the proof of Lemma 4.2, let $X_1, \cdots, X_T$ be the increment of c_i after each iteration, i.e., $c_i = \sum_{j=1}^{T} X_j$. In fact, $X_1, \cdots, X_T$ are independent random variables such that X_j takes value $\frac{1}{p(S)}$ with probability $p(S)$ for each k-graphlet S in G that is isomorphic to g_i, and 0 with the rest probability. Each X_i is at most $(k-1)!(1+\epsilon)^{k-1}Z$ by Fact 4.1 and has expectation l_i. By Fact 2.3 and lemma 4.1,

$$\begin{aligned}
\mathbf{Pr}\left[\frac{c_i}{T} - l_i \geq \frac{\alpha}{2}(1-\alpha)L\right] &\leq 2\exp\left(-\frac{T\alpha^2(1-\alpha)^2L^2}{2((k-1)!(1+\epsilon)^{k-1}Z)^2}\right) \\
&\leq 2\exp\left(-\frac{T\alpha^2(1-\alpha)^2}{2((k-1)!)^2(1+\epsilon)^{2k-2}(k-1)^{2k-2}}\right) \leq \frac{\delta}{4m_k}.
\end{aligned}$$

One can argue similarly that

$$\mathbf{Pr}\left[\frac{l_i}{L} - \frac{c_i}{\left(1+\frac{\alpha}{2}\right)LT} \geq \alpha\right] \leq \frac{\delta}{4m_k}.$$

Recall that the computation above is conditioned on $|\frac{\hat{C}}{T} - L| < \frac{\alpha}{2}L$. We denote this condition by C. Applying a union bound to both sides of Equation (1) for each $1 \leq i \leq m_k$, we obtain that

$$\mathbf{Pr}\left[\bigcup_{i\in[m_k]} |\hat{\mu}_i - \mu_i| > \alpha \,\middle|\, C\right] \leq \frac{\delta}{2}.$$

Also note that

$$T \geq \frac{12(k-1)!(k-1)^{k-1}(1+\epsilon)^{k-1}}{\alpha^2} \ln\frac{4}{\delta},$$

which gives $\mathbf{Pr}\left[\bar{C}\right] \leq \frac{\delta}{2}$ by Lemma 4.2. Therefore, we have the upper bound for overall failure probability:

$$\mathbf{Pr}\left[\bigcup_{i\in[m_k]} |\hat{\mu}_i - \mu_i| > \alpha\right] \leq \mathbf{Pr}\left[\bar{C}\right] + \mathbf{Pr}\left[\bigcup_{i\in[m_k]} |\hat{\mu}_i - \mu_i| > \alpha \,\middle|\, C\right] \leq \delta.$$

□

Now consider the number of passes of Algorithm 5. Recall that, with a memory of M words, we can run Algorithm 5 in parallel so that it returns $\Theta\left(\frac{M}{k^2}\right)$ samples using $O(k)$ passes as described in Section 2.2. Therefore, by Theorem 4.3, $O\left(\frac{(k(1+\epsilon))^{O(k)}}{\alpha^4 M} \ln\frac{1}{\delta}\right)$ passes are sufficient to obtain an estimated k-graphlet distribution with probability at least $1-\delta$ in which each probability in the distribution differs at most α from the standard one.

4.2 Counters vs. Rejection: a Variance View

As mentioned before, the algorithm originally described in [8] uses *rejection* in the sense that it accepts the sampling of S with probability $\frac{\Gamma}{p(S)}$ where $\Gamma \leq \min_S p(S)$. One can transform this algorithm into an approximation of the k-graphlet distribution: set a collection of counters $c_1, \cdots, c_{m_k}$, and whenever a graphlet S is sampled and accepted with probability $\frac{\Gamma}{p(S)}$, add 1 to c_r where g_r is isomorphic to S. The pseudocode of this algorithm is shown in Algorithm 6.

Algorithm 6: Approximate k-Graphlet Distribution via Rejection (Abstract Description)

```
Function Rejection (k, p, T)
    c_1, ..., c_{m_k} ← 0
    for i ← 1 to T do (sequentially/in parallel)
        Sample a k-graphlet S uniformly
        r ← the index such that S is isomorphic to g_r
        c_r ← c_r + 1
    Ĉ ← Σ_{i=1}^{m_k} c_i
    μ̂_i ← c_i / Ĉ for each 1 ≤ i ≤ m_k
    return μ̂_1, ..., μ̂_{m_k}
```

We give an intuitive explanation of why using Counter in place of Rejection can only increase the concentration of the frequency estimates. To this end, consider a modified version of Counter that, instead of adding $\frac{1}{p(S)}$ to the counter of the isomorphism class of S, adds $\frac{\Gamma}{p(S)}$, where Γ is the same used by Rejection. Note that this modification is immaterial as far as the output is concerned: since the output estimates are always rescaled in order to sum to 1, multiplying all counters by a constant Γ does not make any difference. However, now one can see how Rejection is the randomized version of Counter. Indeed, while Rejection increases the counter of S by one with probability $\frac{\Gamma}{p(S)}$, Counter increases it deterministically by $\frac{\Gamma}{p(S)}$. Thus, both algorithms increase the counter by the same amount *in expectation*, but Rejection adds the variance of a coin toss. This implies Counter can only yield higher concentration.

4.3 The Complete Algorithm

Algorithm 7 gives the complete pseudocode for approximating the k-graphlet distribution. It first calls Algorithm 4 to compute a vertex ordering that is a $\frac{1}{1+\epsilon}$-DD ordering with probability at least $1-\delta$. Then it calls Algorithm 1 to obtain the initial distribution (see Section 2.2 for the definition). Finally, it calls Algorithm 5 to estimate the k-graphlet distribution. By combining Theorems 3.2 and 4.3, one can easily verify that Algorithm 7 satisfies Theorem 1.1.

Algorithm 7: Approximate k-Graphlet Distribution

Input: an undirected graph G, the size of graphlets k, parameters ϵ, δ, c, T
Output: the estimated k-graphlet distribution $\hat{\mu}_1, \cdots, \hat{\mu}_{m_k}$

$\prec \leftarrow$ ApproxDD-ES (G, ϵ, δ, c)
$\boldsymbol{p} \leftarrow$ InitDistrib $(G, \prec)$
$\hat{\mu}_1, \cdots, \hat{\mu}_{m_k} \leftarrow$ Counter $(G, \prec, k, \boldsymbol{p}, T)$
return $\hat{\mu}_1, \cdots, \hat{\mu}_{m_k}$

5 Experiments

We conducted experiments to evaluate the practical performance of our algorithm and compare it to the algorithm of [8]. Our experiments are designed to answer the following questions:

(Q1) How do the preprocessing phase (computing an approximate DD order) of ApproxDD-ES and ApproxDD compare in terms of number of passes, peak memory and the quality of DD order?
(Q2) How is the performance of the overall algorithm (for estimating the k-graphlet distribution to achieve a target accuracy) in terms of number of passes and overall peak memory, and how do the "counter" variants compare to the "rejection" variant?

We organize the remainder of this section around these objectives: Section 5.2 addresses (Q1), and Section 5.3 addresses (Q2).

All algorithms were implemented in C++ (including our re-implementation of the baselines from [8] for a fair comparison) and experiments were run on a Ubuntu server equipped with an Intel Xeon Silver 4108 CPU (1.80GHz) and 28GB of main memory. [3]

5.1 Datasets

The computation of the graphlet distribution is relevant not only for sparse graphs, which are common in real-world data, but also for dense graphs such as the similarity graph among facial

[3] Our implementation is available at: https://github.com/l2l7l9p/GD-Streaming.

images. Our theoretical guarantee in Theorem 1.1 holds regardless of graph density. Therefore, we ran experiments on both sparse and dense graphs from various sources to show that our algorithm scales well in both cases. Table 1 reports all dataset statistics.

The datasets were prepared as follows:

- NY Times, Twitter(WWW), Twitter(MPI) and Friendster are from the KONECT website [29][4]. We removed edge directions, weights, self-loops, duplicate edges and any other irrelevant data, so as to retain only a list of undirected edges.
- Sim-0 through Sim-7 are constructed from the database CelebA[5], which consists of 202599 human face images, each one tagged with 40 binary attributes. We represent each image by a vertex, and add an edge between two vertices if the corresponding images differ in at most $0, \ldots, 7$ attributes.
- ER-0 through ER-6 are synthetic graphs, on the same number of vertices as Sim-*. Let n_i and m_i be the number of vertices and the number of edges of Sim-i respectively. Each ER-i is generated according to the Erdős-Rényi model by drawing each edge independently with probability $\frac{m_i}{n_i(n_i-1)/2}$ so that the expected density of ER-i is consistent with Sim-i.

This section gives plots for a representative subset of datasets; the remaining ones are similar.

Table 1. Dataset statistics. The first 4 graphs are real-world large graphs. Sim-* are the graphs indicating the similarity among images. ER-* are synthetic random graphs.

Dataset	$\|V\|$	$\|E\|$	$\|E\|/\|V\|$
NY Times [29]	401,388	69,654,798	173.53
Twitter(WWW) [30]	41,652,230	1,202,513,046	28.87
Twitter(MPI) [17]	52,579,682	1,614,106,187	30.70
Friendster [29]	68,349,466	1,811,849,342	26.51
Sim-0	202,599	948,690	4.68
Sim-1	"	9,825,190	48.50
Sim-2	"	51,840,951	255.88
Sim-3	"	185,601,680	916.10
Sim-4	"	507,709,457	2505.98
Sim-5	"	1,134,892,538	5601.67
Sim-6	"	2,165,414,225	10688.18
Sim-7	"	3,637,218,904	17592.80
ER-0	202,599	949,312	4.69
ER-1	"	9,821,921	48.48
ER-2	"	51,843,003	255.89
ER-3	"	185,602,510	916.11
ER-4	"	507,715,524	2506.01
ER-5	"	1,134,903,860	5601.72
ER-6	"	2,165,448,192	10688.35

Table 2. Memory used (in MB) by ApproxDD and ApproxDD-ES to compute a $\frac{1}{1+\epsilon}$-DD order, with $\epsilon = c = 0.1$. The memory is expressed both in megabytes and (bracketed) as a fraction of the dataset size on disk (which uses 8 bytes per edge).

Dataset	ApproxDD	ApproxDD-ES
NY Times	34.96 (6.58%)	38.38 (7.22%)
Twitter(WWW)	2432.10 (26.51%)	3177.75 (34.64%)
Twitter(MPI)	2904.21 (23.58%)	3735.16 (30.33%)
Friendster	3832.49 (27.72%)	5568.79 (40.29%)
Sim-0	17.10 (236.26%)	9.87 (136.36%)
Sim-1	17.37 (23.17%)	17.44 (23.27%)
Sim-2	20.64 (5.22%)	17.71 (4.48%)
Sim-3	16.42 (1.16%)	17.77 (1.26%)
Sim-4	19.14 (0.49%)	18.10 (0.47%)
Sim-5	15.86 (0.18%)	18.01 (0.21%)
Sim-6	-	17.84 (0.11%)
Sim-7	-	19.15 (0.07%)
ER-0	9.31 (128.54%)	7.76 (107.14%)
ER-1	16.98 (22.66%)	19.14 (25.54%)
ER-2	16.29 (4.12%)	19.07 (4.82%)
ER-3	17.43 (1.23%)	20.66 (1.46%)
ER-4	17.96 (0.46%)	21.03 (0.54%)
ER-5	21.16 (0.24%)	20.86 (0.24%)
ER-6	-	21.00 (0.13%)

5.2 Computing the DD Order

Setup. We evaluate ApproxDD-ES (our algorithm) and ApproxDD (the competitor from [8]) for computing a $\frac{1}{1+\epsilon}$-DD order, fixing $\epsilon = 0.1$. To test the passes-vs-memory tradeoff predicted by Theorem 1.1, we varied $c \in [0, 0.5]$ (the performance of ApproxDD does not depend on c and is therefore constant across the single plot). Each point shown in Figure 4 is the average of 5 executions; executions were terminated after 36 hours if still running.

[4]http://konect.cc/networks/

[5]https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

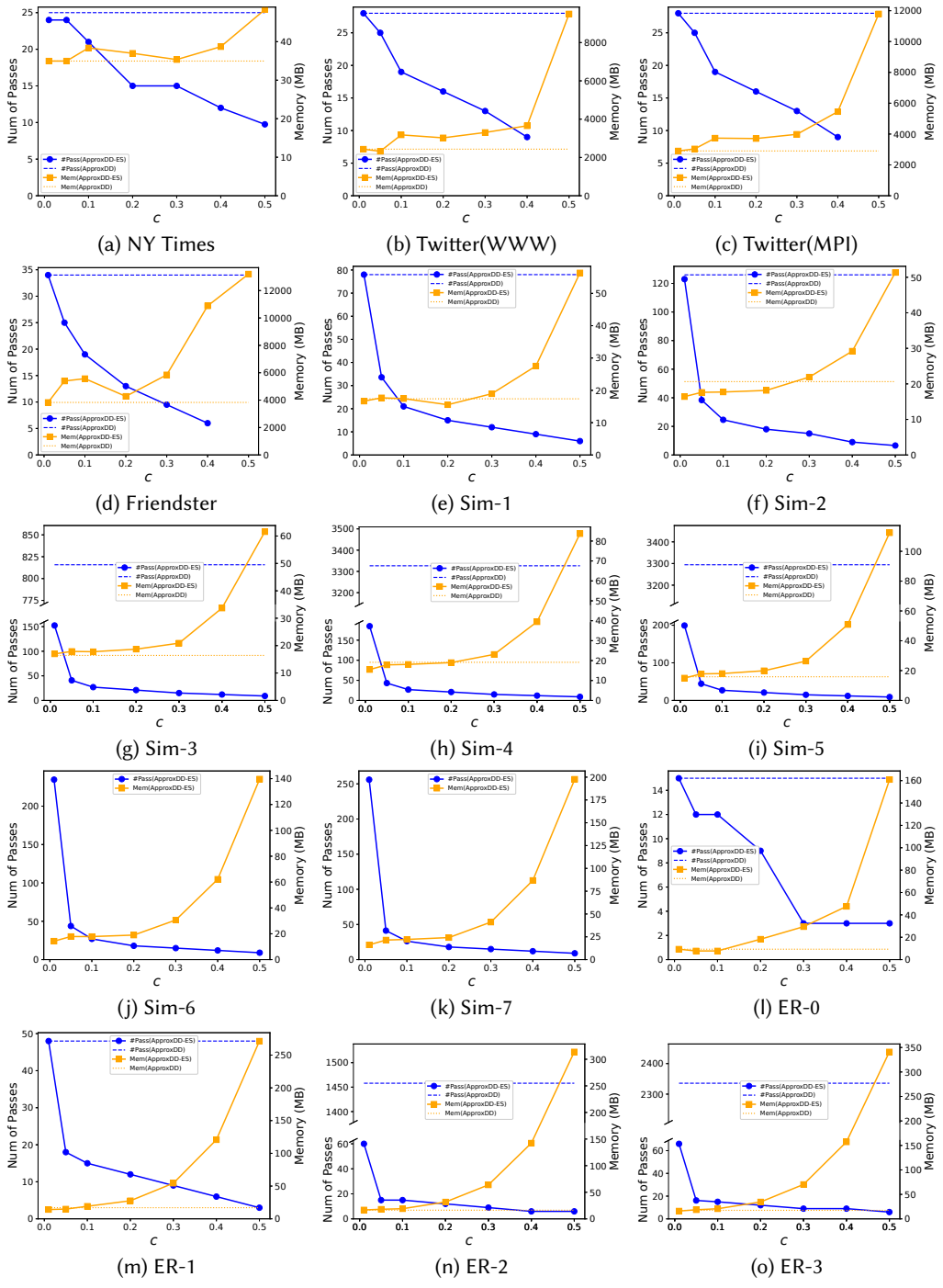
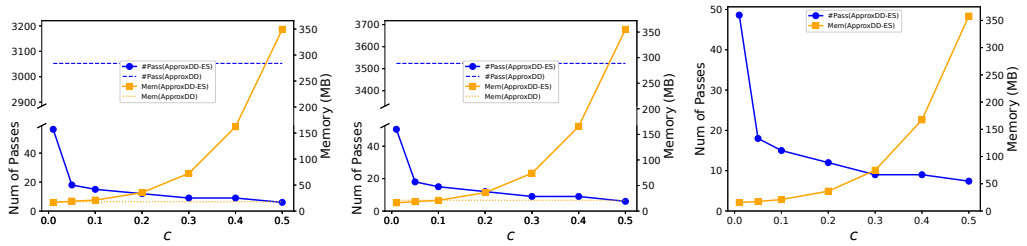


Fig. 4. Number of passes and peak memory usage, as a function of c, for ApproxDD and ApproxDD-ES. Each point shown is the average of 5 executions. Missing points for ApproxDD mean it did not terminate within 36 hours.



(p) ER-4 (q) ER-5 (r) ER-6

Fig. 4. (Continued) Number of passes and peak memory usage, as a function of c, for ApproxDD and ApproxDD-ES. Each point shown is the average of 5 executions. Missing points for ApproxDD mean it did not terminate within 36 hours.

For both algorithms we implemented the heuristic in [8, Appendix E], so as to improve the running time on large sparse graphs. Roughly speaking, at the beginning of each iteration, the heuristic sorts the vertices in nonincreasing order of degree and selects the largest prefix whose induced graph fits in memory. It then compares (i) running the next iteration of the peeling procedure (i.e., the while-loop in Algorithm 4) on this induced subgraph versus (ii) removing vertices in the prefix optimally, and keeps the option that leads to the largest drop in the current maximum degree.

We additionally evaluate the quality of the DD order produced by ApproxDD-ES under different choices of c. Given a graph $G = (V, E)$ and a vertex order $\prec$, for each vertex v such that $d(v|G(v)) > 0$, define

$$\vartheta_v := \frac{d(v|G(v))}{\Delta(G(v))}.$$

By definition, $\min_{v \in V : d(v|G(v))>0} \vartheta_v$ is exactly the largest value ϑ for which $\prec$ is a ϑ-DD order. To compute ϑ_v efficiently, write $\prec$ as $u_1, \ldots, u_n$ and index each edge $\{u_i, u_j\}$ $(i < j)$ by (i, j). Using an external sorting, we process edges in reverse-lexicographic order of (i, j) while maintaining the current local degree of each vertex and the maximum local degree seen so far. Whenever all edges of $G(v)$ have been processed for some v, we obtain $\vartheta_v = d(v|G(v))/\Delta(G(v))$. For readability we report $\epsilon_v := \vartheta_v^{-1} - 1$, i.e., $\prec$ is a $\frac{1}{1+\max_{v \in V : d(v|G(v))>0} \epsilon_v}$-DD order.

For each graph, we compute the empirical distribution of ϵ_v and take the average of 5 executions.

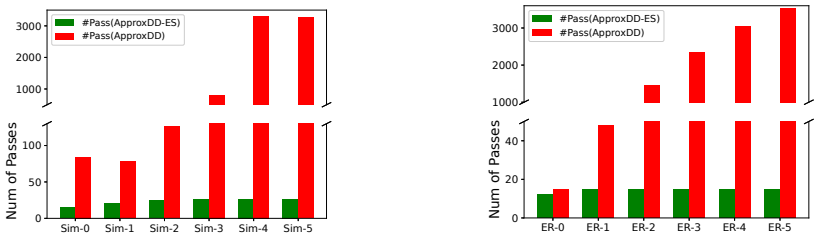
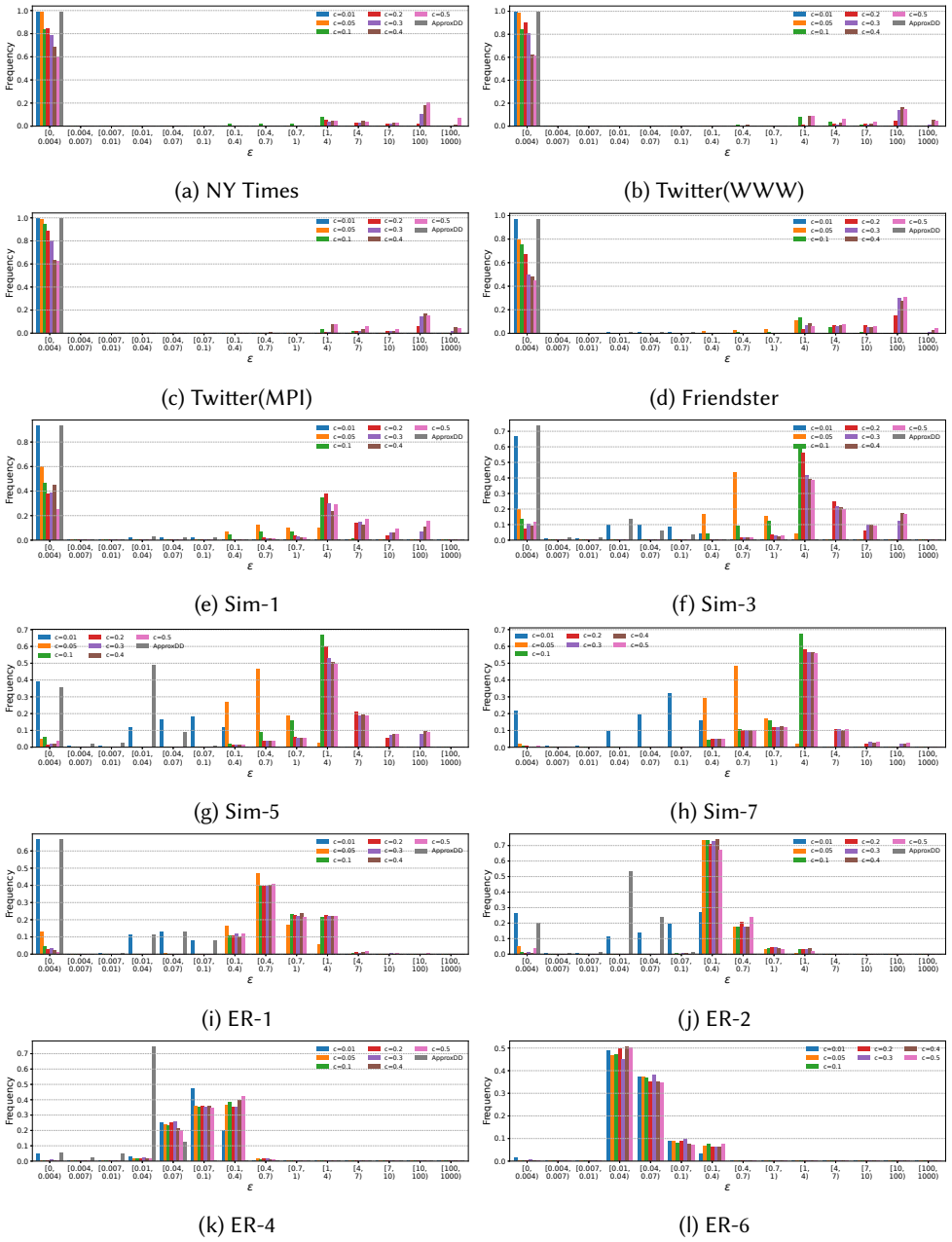


Fig. 5. Number of passes on graphs of increasing densities using ApproxDD and ApproxDD-ES, when $c = 0.1$.

Passes–Memory Tradeoff. Figure 4 shows the passes–memory tradeoff of ApproxDD-ES and compares it to ApproxDD (which is insensitive to c). Two remarks are in order. The first one is



(a) NY Times (b) Twitter(WWW)

(c) Twitter(MPI) (d) Friendster

(e) Sim-1 (f) Sim-3

(g) Sim-5 (h) Sim-7

(i) ER-1 (j) ER-2

(k) ER-4 (l) ER-6

Fig. 6. Empirical distribution of ϵ_v under different c, using ApproxDD-ES.

that the results agree with Theorem 3.2: ApproxDD-ES exhibits a clear memory-passes tradeoff, and the number of passes is very stable across different datasets (e.g., at most 25 for $c = 0.1$) in agreement with the $O(1/c)$ bound. In contrast, the number of passes used by ApproxDD goes with $\log n$, and therefore explodes on large datasets.

The second remark is that ApproxDD-ES outperforms ApproxDD. On most datasets, for $c = 0.1$, ApproxDD-ES uses at least 32% fewer passes than ApproxDD by using at most 45% more memory. The advantage of ApproxDD-ES is amplified on relatively dense graphs. On the Similarity and ER families (see Figures 4e to 4r and Figure 5), increasing density causes ApproxDD to require dramatically more passes (and to fail to terminate within 36 hours on Sim-6 and Sim-7), whereas ApproxDD-ES remains within a small constant number of passes for $c = 0.1$. For example, on Sim-3 and Sim-5, ApproxDD-ES for $c = 0.1$ uses marginally more memory than ApproxDD, and yet it makes 20× and 110× fewer passes.

Finally, Table 2 reports the memory used for computing a $\frac{1}{1+\epsilon}$-DD order on all datasets when $c = 0.1$. Across datasets, ApproxDD-ES uses moderately more memory than ApproxDD, matching the tradeoff suggested by Theorem 1.1.

DD Order Quality. Figure 6 shows the empirical distribution of ϵ_v. On large sparse graphs (NY Times, Twitter(WWW), Friendster), ApproxDD-ES typically outputs orders that are extremely close to a perfect DD order: for small and moderate c, the mass is concentrated in the smallest ϵ bin (near 0). On dense graphs (ER and Sim), the distribution remains tightly concentrated on small ϵ (e.g., mostly $\epsilon \leq 0.4$ for ER and mostly $\epsilon \leq 4$ for Sim). Overall, these results suggest that ApproxDD-ES produces high-quality DD orders in practice.

5.3 Computing the Graphlet Distribution

We test the performance of our full algorithm for approximating the k-graphlet distribution, ApproxDD-ES +Counter (Algorithm 7), and of its competitor ApproxDD +Counter (the algorithm of [8]). Note that the algorithm originally described in [8] actually uses *rejection*, rather than using *counters*. We also test this "default" competitor, ApproxDD +Rejection, where Rejection is introduced in Algorithm 6. As expected, ApproxDD +Rejection is much more inefficient than the "counter" variants, as it gains information only upon accepting a graphlet, which happens with probability only $k^{-O(k)}$.

We ran the algorithms for $k = 4, 5, 6$, using as ground truth the average of 5 runs of Motivo [15]. We set the parameters of our algorithms to $\epsilon = 0.1$, $c = 0.1$, and $\delta = 0.02$. For each algorithm and each dataset, we start by computing a $\frac{1}{1+\epsilon}$-DD order. We then ran 5 batches of parallel sampling. After each batch, we computed the L_∞ distance between (i) the ground truth and (ii) the k-graphlet distribution estimate obtained so far. This experiment was then repeated 5 times, and we took the average.

Accuracy versus passes. Figure 7 shows the number of passes versus L_∞ distance achieved by the algorithms. (Due to space limitations we show the plots only for a representative subset of datasets). Note that the number of passes is the sum of the passes taken by preprocessing and of $2k - 1$ passes for each sampling batch. The first observation is that, on every dataset and for every value of k, save a few exceptions at $k = 4$, ApproxDD-ES +Counter and ApproxDD +Counter yield an error smaller than ApproxDD +Rejection. The gap between the two errors increases with k, with ApproxDD +Rejection lagging much behind ApproxDD-ES +Counter and ApproxDD +Counter for $k = 6$. This is explained by the aforementioned $k^{-O(k)}$ acceptance probability of ApproxDD +Rejection, and confirms that the "counter" approach is significantly better than the "rejection" approach. The second observation is that, again, ApproxDD-ES +Counter uses fewer passes than ApproxDD +Counter, outperforming it by orders of magnitude on dense graphs. Note that the sampling phase is identical between the two algorithms (but is ran with different DD-orders), and seems to have a relatively small impact on the error-vs-passes tradeoff, if compared with the preprocessing phase. This is coherent with the results of Section 5.2 and with our theoretical

Table 3. Overall memory usage of the algorithms (in MB). The percentages in brackets represent the ratio of memory usage over the disk size of the graph (each edge is represented in 8 bytes).

Dataset	ApproxDD-ES +Counter	ApproxDD +(Rejection or Counter)	Motivo		
			$k = 4$	$k = 5$	$k = 6$
NY Times	38.83 (7.31%)	34.97 (6.58%)	856.60 (161.19%)	913.18 (171.84%)	1115.45 (209.90%)
Twitter (WWW)	3370.27 (36.74%)	2691.84 (29.34%)	12407.77 (135.24%)	16687.46 (181.89%)	32576.35 (355.08%)
Twitter (MPI)	4391.69 (35.66%)	3382.18 (27.46%)	17173.68 (139.46%)	24232.61 (196.78%)	48619.79 (394.81%)
Friendster	5895.11 (42.65%)	4429.02 (32.04%)	20857.70 (150.89%)	26691.57 (193.09%)	45527.43 (329.35%)
Sim-0	16.50 (227.90%)	17.98 (248.37%)	99.21 (1370.71%)	122.89 (1697.89%)	192.33 (2657.24%)
Sim-1	17.46 (23.29%)	18.33 (24.45%)	181.23 (241.77%)	260.64 (347.71%)	513.17 (684.59%)
Sim-2	18.25 (4.61%)	19.17 (4.85%)	508.91 (128.67%)	611.73 (154.67%)	932.91 (235.87%)
Sim-3	18.84 (1.33%)	19.81 (1.40%)	1531.61 (108.16%)	1638.01 (115.68%)	1978.04 (139.69%)
Sim-4	19.16 (0.49%)	20.16 (0.52%)	3984.58 (102.87%)	4097.13 (105.77%)	4440.09 (114.63%)
Sim-5	19.39 (0.22%)	20.40 (0.24%)	8692.20 (100.39%)	8766.39 (101.25%)	9158.10 (105.77%)
Sim-6	19.82 (0.12%)	-	17176.55 (103.97%)	17297.12 (104.70%)	17656.74 (106.88%)
Sim-7	20.05 (0.07%)	-	28061.98 (101.13%)	28112.06 (101.31%)	28305.63 (102.00%)
ER-0	18.22 (251.50%)	17.78 (245.55%)	125.80 (1736.88%)	214.46 (2961.11%)	504.20 (6961.54%)
ER-1	20.96 (27.97%)	19.99 (26.67%)	194.11 (259.04%)	298.66 (398.56%)	643.80 (859.14%)
ER-2	20.55 (5.20%)	20.40 (5.16%)	511.20 (129.24%)	618.41 (156.35%)	964.53 (243.86%)
ER-3	21.18 (1.50%)	20.31 (1.43%)	1531.78 (108.17%)	1638.54 (115.71%)	1982.57 (140.01%)
ER-4	21.05 (0.54%)	20.16 (0.52%)	3985.01 (102.88%)	4101.33 (105.88%)	4441.28 (114.66%)
ER-5	21.66 (0.25%)	20.43 (0.24%)	8692.22 (100.39%)	8801.50 (101.65%)	9193.64 (106.18%)
ER-6	21.69 (0.13%)	-	16902.60 (102.31%)	16999.68 (102.90%)	17619.79 (106.65%)

bounds. Overall, ApproxDD-ES +Counter ensures an L_∞ error of at most 0.01 with < 50 passes for $k = 4$, of at most 0.02 with < 60 passes for $k = 5$, and at most 0.05 with < 80 passes for $k = 6$.

Memory usage. Table 3 shows the memory usage of all algorithms. In fact, in all our experiments, the total memory usage of ApproxDD-ES +Counter and ApproxDD +(Rejection or Counter) was dominated by the memory required for the preprocessing (which is already shown graphically in Section 5.2). Again, the memory usage of ApproxDD-ES +Counter is slightly worse than ApproxDD +Rejection and ApproxDD +Counter, but not by large factors. This confirms that ApproxDD-ES +Counter is competitive.

6 Conclusion

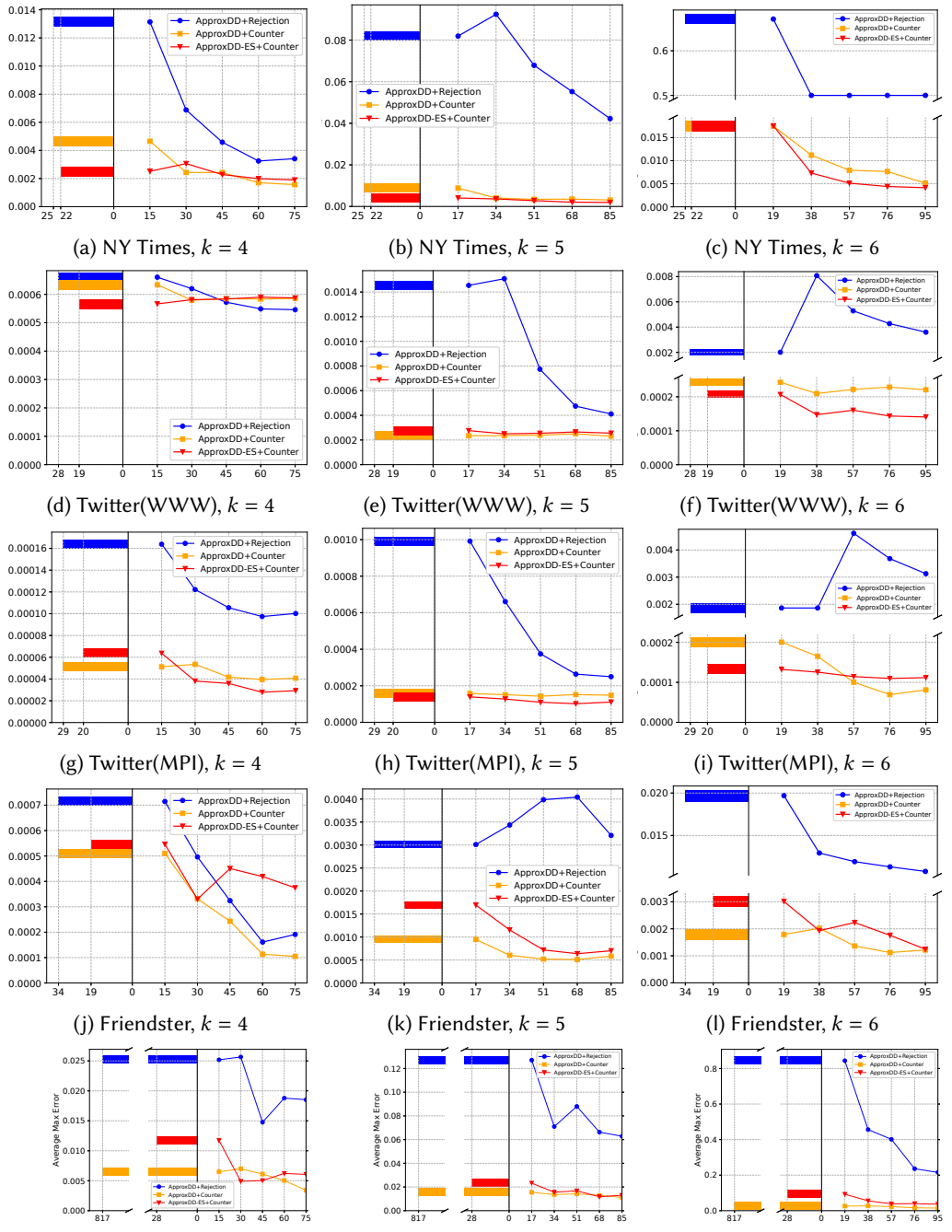
This paper presents a novel streaming algorithm for approximating k-graphlet distributions in large graphs that cannot fit entirely in memory. By introducing an improved method for computing an approximate degree-dominating (DD) order—a key preprocessing step—our algorithm reduces the number of passes from $O(\log n)$ (as in prior work) to $O(1/c)$, using $\tilde{O}(n^{1+c})$ memory. This result is nearly optimal, as a lower bound rules out $O(1)$-pass algorithms with sublinear memory.

Empirical evaluations on real-world and synthetic graphs demonstrate that our algorithm significantly outperforms the state-of-the-art, especially on moderately dense graphs, where it reduces the number of passes by orders of magnitude while maintaining comparable memory usage. The integration of a Horvitz–Thompson estimator further enhances efficiency by avoiding the high rejection rates of earlier sampling methods.

Overall, our work provides a scalable, predictable, and theoretically sound solution for graphlet analysis in the streaming model, bridging the gap between theoretical guarantees and practical performance.

Acknowledgments

T-H. Hubert Chan was partially supported by the Hong Kong RGC grants 17201823 and 17203122.



(a) NY Times, $k = 4$ (b) NY Times, $k = 5$ (c) NY Times, $k = 6$

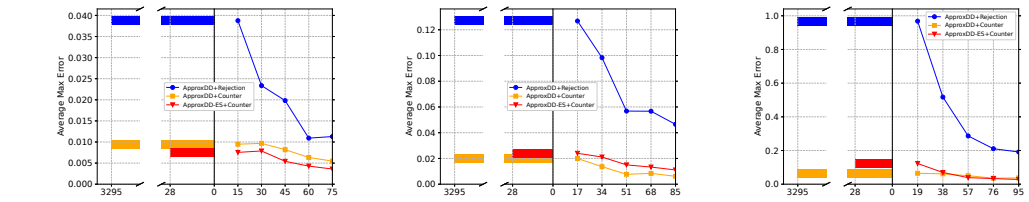
(d) Twitter(WWW), $k = 4$ (e) Twitter(WWW), $k = 5$ (f) Twitter(WWW), $k = 6$

(g) Twitter(MPI), $k = 4$ (h) Twitter(MPI), $k = 5$ (i) Twitter(MPI), $k = 6$

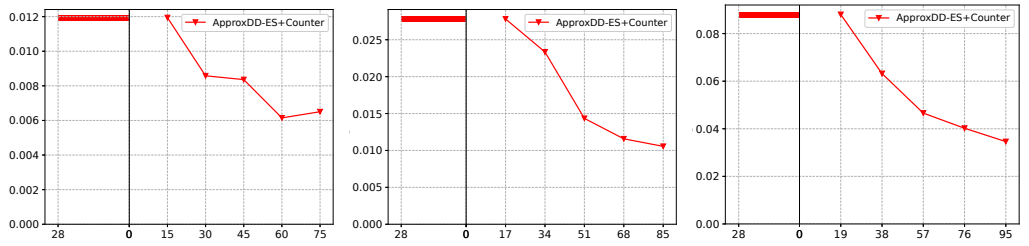
(j) Friendster, $k = 4$ (k) Friendster, $k = 5$ (l) Friendster, $k = 6$

(m) Sim-3, $k = 4$ (n) Sim-3, $k = 5$ (o) Sim-3, $k = 6$

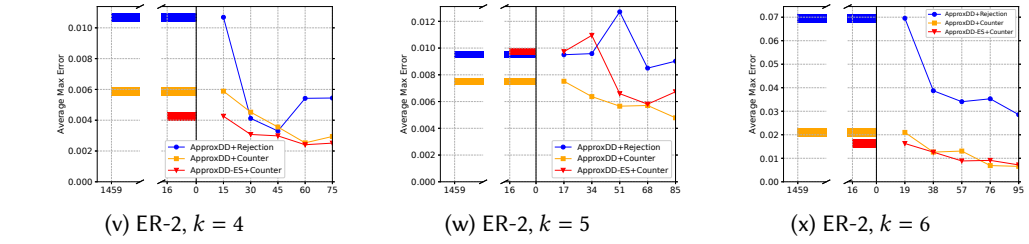
Fig. 7. L_∞ distance between the ground-truth and the estimated k-graphlet distribution as a function of the number of passes for sampling. The X-axis shows the number of passes (preprocessing on the left, sampling on the right). Missing points for ApproxDD mean it did not terminate within 36 hours.



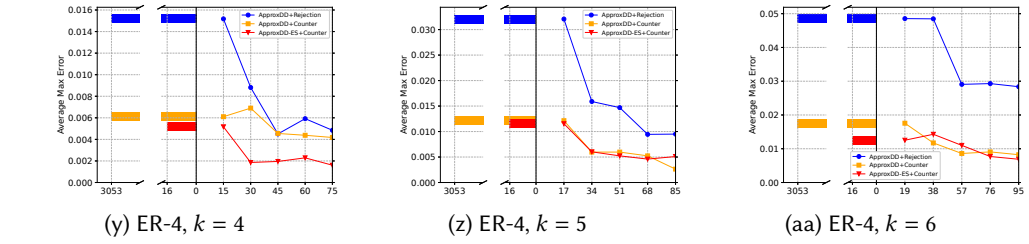
(p) Sim-5, $k = 4$ (q) Sim-5, $k = 5$ (r) Sim-5, $k = 6$



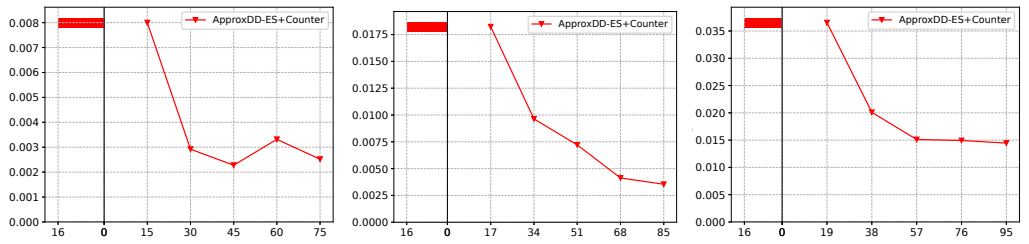
(s) Sim-7, $k = 4$ (t) Sim-7, $k = 5$ (u) Sim-7, $k = 6$



(v) ER-2, $k = 4$ (w) ER-2, $k = 5$ (x) ER-2, $k = 6$



(y) ER-4, $k = 4$ (z) ER-4, $k = 5$ (aa) ER-4, $k = 6$



(ab) ER-6, $k = 4$ (ac) ER-6, $k = 5$ (ad) ER-6, $k = 6$

Fig. 7. (Continued) L_∞ distance between the ground-truth and the estimated k-graphlet distribution as a function of the number of passes for sampling. The X-axis shows the number of passes (preprocessing on the left, sampling on the right). Missing points for ApproxDD mean it did not terminate within 36 hours.

References

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

[2] Nesreen K. Ahmed, Nick G. Duffield, Jennifer Neville, and Ramana Rao Kompella. 2014. Graph sample and hold: a framework for big-graph analytics. In *KDD*. ACM, 1446–1455.

[3] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Densest subgraph in streaming and MapReduce. *Proceedings of the VLDB Endowment* 5, 5 (Jan. 2012), 454–465. doi:10.14778/2140436.2140442

[4] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. 2002. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*. ACM/SIAM, 623–632.

[5] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. 2008. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, Las Vegas Nevada USA, 16–24. doi:10.1145/1401890.1401898

[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: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 *Proc. of IEEE ICDM 2012*. 91–100. doi:10.1109/ICDM.2012.87

[8] Yann Bourreau, Marco Bressan, T.-H. Hubert Chan, Qipeng Kuang, and Mauro Sozio. 2024. Efficient Streaming Algorithms for Graphlet Sampling. In *NeurIPS*.

[9] Marco Bressan. 2021. Efficient and Near-Optimal Algorithms for Sampling Connected Subgraphs. In *Proc. of ACM STOC* (Virtual, Italy). 1132–1143. doi:10.1145/3406325.3451042

[10] Marco Bressan. 2021. Efficient and near-optimal algorithms for sampling connected subgraphs. arXiv:2007.12102 [cs.DS]

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

[12] Marco Bressan. 2023. Efficient and Near-Optimal Algorithms for Sampling Small Connected Subgraphs. *ACM Trans. Algorithms* 19, 3, Article 26 (jun 2023), 40 pages. doi:10.1145/3596495

[13] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2017. Counting Graphlets: Space vs Time. In *Proc. of ACM WSDM*. 557–566. doi:10.1145/3018661.3018732

[14] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2018. Motif Counting Beyond Five Nodes. *ACM Trans. Knowl. Discov. Data* 12, 4, Article 48 (April 2018), 25 pages. doi:10.1145/3186586

[15] 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:10.14778/3342263.3342640

[16] 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, Article 96 (May 2021), 27 pages. doi:10.1145/3447397

[17] Meeyoung Cha, Hamed Haddadi, Fabrício Benevenuto, and P. Krishna Gummadi. 2010. Measuring User Influence in Twitter: The Million Follower Fallacy. In *ICWSM*. The AAAI Press.

[18] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. 2006. Strong computational lower bounds via parameterized complexity. *J. Comput. System Sci.* 72, 8 (2006), 1346 – 1367. doi:10.1016/j.jcss.2006.04.007

[19] 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:10.14778/3021924.3021940

[20] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and Subgraph Listing Algorithms. *SIAM J. Comput.* 14, 1 (1985), 210–223. doi:10.1137/0214017

[21] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. 2005. On graph problems in a semi-streaming model. *Theoretical Computer Science* 348, 2-3 (Dec. 2005), 207–216. doi:10.1016/j.tcs.2005.09.013

[22] Mohsen Ghaffari, Silvio Lattanzi, and Slobodan Mitrovic. 2019. Improved Parallel Algorithms for Density-Based Network Clustering. In *ICML (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, 2201–2210.

[23] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. 2022. An Efficient Framework for Clustered Federated Learning. *IEEE Trans. Inf. Theory* 68, 12 (2022), 8076–8091.

[24] Guyue Han and Harish Sethu. 2016. Waddling Random Walk: Fast and Accurate Mining of Motif Statistics in Large Graphs. In *Proc. of IEEE ICDM*. 181–190. doi:10.1109/ICDM.2016.0029

[25] Daniel G Horvitz and Donovan J Thompson. 1952. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association* 47, 260 (1952), 663–685.

[26] Mark Jerrum and Kitty Meeks. 2015. The parameterised complexity of counting connected subgraphs and graph motifs. *J. Comput. Syst. Sci.* 81, 4 (2015), 702–716. doi:10.1016/j.jcss.2014.11.015

[27] Madhav Jha, C. Seshadhri, and Ali Pinar. 2015. A Space-Efficient Streaming Algorithm for Estimating Transitivity and Triangle Counts Using the Birthday Paradox. *ACM Trans. Knowl. Discov. Data* 9, 3 (2015), 15:1–15:21.

[28] Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. 2012. Counting Arbitrary Subgraphs in Data Streams. In *ICALP (2) (Lecture Notes in Computer Science, Vol. 7392)*. Springer, 598–609.

[29] Jérôme Kunegis. 2013. KONECT: the Koblenz network collection. In *WWW (Companion Volume)*. International World Wide Web Conferences Steering Committee / ACM, 1343–1350.
[30] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue B. Moon. 2010. What is Twitter, a social network or a news media?. In *WWW*. ACM, 591–600.
[31] Yongsub Lim, Minsoo Jung, and U. Kang. 2018. Memory-Efficient and Accurate Sampling for Counting Local Triangles in Graph Streams: From Simple to Multigraphs. *ACM Transactions on Knowledge Discovery from Data* 12, 1 (Feb. 2018), 1–28. doi:10.1145/3022186
[32] Ryuta Matsuno and Aristides Gionis. 2020. Improved mixing time for k-subgraph sampling. In *Proc. of SIAM SDM*. 568–576. doi:10.1137/1.9781611976236.64
[33] 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:10.1145/2402.322385
[34] 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:10.1126/science.298.5594.824
[35] Kirill Paramonov, Dmitry Shemetov, and James Sharpnack. 2019. Estimating Graphlet Statistics via Lifting. In *Proc. of ACM KDD*. 587–595. doi:10.1145/3292500.3330995
[36] A. Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. 2013. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment* 6, 14 (Sept. 2013), 1870–1881. doi:10.14778/2556549.2556569
[37] 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. of AAAI* 34, 04 (Apr. 2020), 5387–5394. doi:10.1609/aaai.v34i04.5987
[38] Tanay Kumar Saha and Mohammad Al Hasan. 2015. Finding Network Motifs Using MCMC Sampling. In *Proc. of CompleNet*. 13–24. doi:10.1007/978-3-319-16112-9_2
[39] Ahmet Erdem Saríyüce, Buğra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, and Ümit V. Çatalyürek. 2013. Streaming algorithms for k-core decomposition. *Proceedings of the VLDB Endowment* 6, 6 (April 2013), 433–444. doi:10.14778/2536336.2536344
[40] Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *AISTATS (JMLR Proceedings, Vol. 5)*. JMLR.org, 488–495.
[41] Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. 2017. Triest: Counting local and global triangles in fully dynamic streams with fixed memory size. *ACM TKDD* 11, 4 (2017), 43.
[42] Lorenzo De Stefani, Erisa Terolli, and Eli Upfal. 2017. Tiered sampling: An efficient method for approximate counting sparse motifs in massive graph streams. In *IEEE BigData*. IEEE Computer Society, 776–786.
[43] Kun Tu, Jian Li, Don Towsley, Dave Braines, and Liam D. Turner. 2019. Gl2vec: Learning Feature Representation Using Graphlets for Directed Networks. In *Proc. of IEEE/ACM ASONAM*. 216–221. doi:10.1145/3341161.3342908
[44] Pinghui Wang, John C. S. Lui, Bruno Ribeiro, Don Towsley, Junzhou Zhao, and Xiaohong Guan. 2014. Efficiently Estimating Motif Statistics of Large Networks. *ACM TKDD* 9, 2, Article 8 (2014), 27 pages. doi:10.1145/2629564

Received October 2025; revised January 2026; accepted February 2026