Estimating Graphlet Statistics via Lifting

Kirill Paramonov
Google
San Bruno, CA, USA
kir.paramonov@gmail.com

Dmitry Shemetov
University of California, Davis
Davis, CA, USA
dshemetov@ucdavis.edu

James Sharpnack
University of California, Davis
Davis, CA, USA
jsharpna@gmail.com

ABSTRACT

Exploratory analysis over network data is often limited by the ability to efficiently calculate graph statistics, which can provide a model-free understanding of the macroscopic properties of a network. We introduce a framework for estimating the graphlet count—the number of occurrences of a small subgraph motif (e.g. a wedge or a triangle) in the network. For massive graphs, where accessing the whole graph is not possible, the only viable algorithms are those that make a limited number of vertex neighborhood queries. We introduce a Monte Carlo sampling technique for graphlet counts, called *Lifting*, which can simultaneously sample all graphlets of size up to k vertices for arbitrary k. This is the first graphlet sampling method that can provably sample every graphlet with positive probability and can sample graphlets of arbitrary size k. We outline variants of lifted graphlet counts, including the ordered, unordered, and shotgun estimators, random walk starts, and parallel vertex starts. We prove that our graphlet count updates are unbiased for the true graphlet count and have a controlled variance for all graphlets. We compare the experimental performance of lifted graphlet counts to the state-of-the art graphlet sampling procedures: Waddling and the pairwise subgraph random walk.

ACM Reference Format:
Kirill Paramonov, Dmitry Shemetov, and James Sharpnack. 2019. Estimating Graphlet Statistics via Lifting. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330995

1 INTRODUCTION

In 1970, [7] discovered that transitivity—the tendency of friends of friends to be friends themselves—is a prevalent feature in social networks. Since that early discovery, real-world networks have been observed to have many other common macroscopic features, and these discoveries have led to probabilistic models for networks that display these phenomena. The observation that transitivity and other common subgraphs are prevalent in networks motivated the exponential random graph model (ERGM) [8]. [2] demonstrated that many large networks display a scale-free power law degree distribution, and provided a model for constructing such graphs. Similarly, the small world phenomenon—that networks display surprisingly few degrees of separation—motivated the network model in [23]. While network science is often driven by the observation and modelling of common properties in networks, it is incumbent on the practicing data scientist to explore network data using statistical methods.

One approach to understanding network data is to fit free parameters in these network models to the data through likelihood-based or Bayesian methods [20, 22]. Network statistics, such as the clustering coefficient, algebraic connectivity, and degree sequence, are more flexible tools. A good statistic can be used to fit and test models, for example, [23] used the local clustering coefficient, a measure of the number of triangles relative to wedges, to test if a network is a small-world graph. It was discovered that re-occurring subgraph patterns can be used to differentiate real-world networks, and that genetic networks, neural networks, and internet networks all presented different common interconnectivity patterns, [13]. In this work, we will propose a new method for counting the occurrences of any subgraph pattern, otherwise known as *graphlets*—a term coined in [15]—or motifs.

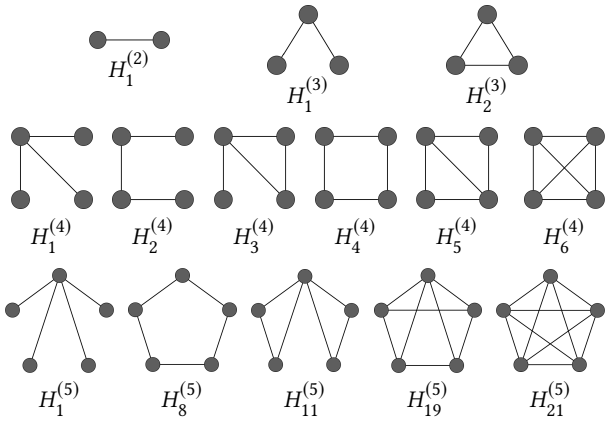

Figure 1: Examples of graphlets

A graphlet is a small connected graph topology, such as a triangle, wedge, or k-clique, which we will use to describe the local behavior of a larger network (example graphlets of size 3, 4, and 5, can be seen in Figure 1). Let the graph in question be $G = (V, E)$ where V is a set of vertices and E is a set of unordered pairs of vertices (G is assumed to be connected, undirected, and unweighted). Imagine specifying a k-graphlet and testing for every induced subgraph of the graph (denoted $G|\{v_1, \ldots, v_k\}$ where $v_1, \ldots, v_k \in V$), if it is isomorphic to the subgraph (it has the same topology). We would like to compute the number of Connected Induced Subgraphs of size k (denoted by k-*CIS* throughout) for which this match holds.

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330995

We call this number the *graphlet counts* and the proportion of the number of such matches to the total number of k-CISs is called the *graphlet coefficient*.

Graphlets are the graph analogue of wavelets (small oscillatory functions that are convolved with a signal to produce wavelet coefficients) because they are small topologies that are matched to induced subgraphs of the original graph to produce the graphlet coefficients. Graphlet coefficients, also referred to as graph moments, are used to fit certain graph models by the method of moments, [4], and also are used to understand biological networks [16]. A naive graphlet counting method simply counts every induced subgraph which takes on the order of n^k iterations. In a typical graph, the majority of induced subgraphs are disconnected, which would not count as a graphlet, so the majority of these iterates would not count toward the graphlet coefficient. We propose a class of Monte Carlo sampling methods called lifting that allow us to quickly estimate graphlet coefficients. The lifting step takes a CIS of size $k-1$ and produces a CIS of size k by adding an adjacent vertex to it (according to a specific scheme), thereby forming graphlet samples in an inductive, bottom-up fashion (see 2).

Monte Carlo sampling procedures perform random walks on graphlets of a certain size within a large network. These methods have the advantage of only requiring local graph information at every step, which makes them memory efficient in computation. The challenge in designing such an algorithm is showing that the sampling procedure is unbiased in its graphlet estimates, has low variance, and is sample efficient. Two such methods are GUISE algorithm of [3] and the pairwise subgraph random walk (PSRW) of [21], which differ in the way they perform a random walk between CIS samples. Another option is to generate a sequence of vertices that induces a CIS sample, which has been done in [11] using an algorithm called the Waddling random walk. Very efficient exact count methods exist [1, 5, 14, 17], but they have not been extended to counting graphlets larger than $k = 5$.

We note that graphlet frequencies are one type of graph feature that relate to the proportion of motifs in a graph. However, they do not reflect more global properties of a graph, and are not comparable to graph embeddings such as GraphSAGE [10] or node2vec [9]. Hence we do not offer such comparisons.

1.1 Our contributions

We provide two methods, the ordered lift estimator and the unordered lift estimator, which differ in the way that subgraphs are represented and counted. The ordered estimator allows for a modification, called *shotgun sampling* that samples multiple subgraphs in one shot, which effectively gives it more samples per iteration. For our theoretical component, we prove that the estimated graphlet coefficients are unbiased, and prove that the variance of the estimator scales like Δ^{k-2} where Δ is the maximum degree. We conclude with real-world network experiments that reinforce the contention that graphlet lifting is competitive with a specialized Waddling implementation and has better accuracy than subgraph random walks. We implement 6-graphlet lifting on a 2.9M vertex Facebook graph, demonstrating that lifting is the first sampling scheme that can scale to 1M sized graphs and k-graphlets where $k > 5$, and do so without any specialized modifications.

2 SAMPLING GRAPHLETS

2.1 Definitions and notation

Recall our definitions thus far: $G = (V, E)$ is a simple graph, $G|W$ is the induced subgraph for $W \subset V$. The set of all connected induced k-subgraphs (or k-CISs) of G is denoted by $\mathcal{V}_k(G)$ (or simply $\mathcal{V}_k$). An unordered set of vertices is denoted $\{v_1, \ldots, v_k\}$ while an ordered list is denoted $[v_1, \ldots, v_k]$. Let $H_1, H_2, \ldots, H_l$ be all non-isomorphic motifs for which we would like the graphlet counts. For $T \in \mathcal{V}_k(G)$, we say that "T is subgraph of type m" if T is isomorphic to H_m, and denote this with $T \sim H_m$. The number of k-subgraphs in G of type m is equal to $N_m(G) = \sum_{T \in \mathcal{V}_k(G)} \mathbb{1}(T \sim H_m)$, where $\mathbb{1}(A)$ is the indicator function. For a subgraph $S \subseteq G$, denote V_S to be the set of its vertices, E_S to be the set of its edges. Denote $\mathcal{N}_v(S)$ (vertex neighborhood of S) to be the set of all vertices adjacent to some vertex in S not including S itself. Denote $\mathcal{N}_e(S)$ (edge neighborhood of S) to be the set of all edges that connect a vertex from S and a vertex outside of S. Also, denote $\deg(S)$ (degree of S) to be the number of edges in $\mathcal{N}_e(S)$, and denote $\deg_S(u)$ (S-degree of u) to be the number of vertices from S that are connected to u. Note that $\deg(S) + 2|E_S| = \sum_{v \in V_S} \deg(v)$.

2.2 Prior graphlet sampling methods

The ideal Monte Carlo procedure would sequentially sample CISs uniformly at random from the set $\mathcal{V}_k(G)$, classify their type, and update the corresponding counts. Unfortunately, uniformly sampling CISs is not a simple task because a random set of k vertices is unlikely to be connected. CIS sampling methods require Monte Carlo Markov Chains (MCMCs) for which one can calculate the stationary distribution, π, over the elements of $\mathcal{V}_k$. First, let us consider how we update the graphlet counts, $N_m(G)$, given a sample of CISs, $T_1, T_2, \ldots, T_n$. Then we use Horvitz-Thompson inverse probability weighting to estimate the graphlet counts,

$$\hat{N}_m(G) := \frac{1}{n} \sum_{i=1}^{n} \frac{\mathbb{1}(T_i \sim H_m)}{\pi(T_i)}. \tag{1}$$

It is simple to see that this is an unbiased estimate of the graphlet counts as long as π is supported over all elements of $\mathcal{V}_k$.

Let us describe the subgraph random walk in [21] called the pairwise subgraph random walk (PSRW). In order to perform a random walk where the states are subgraphs $\mathcal{V}_k$, we form the CIS-relationship graph. Two k-CISs, $T, S \in \mathcal{V}_k$ are connected with an edge if and only if vertex sets of T and S differ by one element, i.e. when $|V(T) \cap V(S)| = k - 1$. Given the graph structure, we sample k-CISs by a random walk on the set $\mathcal{V}_k$, which is called Subgraph Random Walk (SRW). Because the transition from state $S \in \mathcal{V}_k$ is made uniformly at random to each adjacent CIS, we know that the stationary distribution will sample each edge in the CIS-relationship graph with equal probability. This fact enables [21] to provide a local estimator of the stationary probability $\pi(S)$. PSRW is a modification of the SRW algorithm, where each transition is performed from S to T in $\mathcal{V}_{k-1}$ and then the k-CIS $S \cup T$ is returned.

Being a random walk-based procedure, insufficient mixing can cause PSRW to be biased if the burn-in period is not long enough. It was pointed out in [5] that the mixing time of the SRW can be of order $O(n^{k-2})$, even if the mixing time of the random walk on the original graph G is of constant order $O(1)$. PSRW also requires

global constants based on the CIS-graph, which can be computationally intractable (super-linear time). It should also be noted that a burn-in period is required for PSRW to converge to the stationary distribution, so any distributed sampling scheme will require all runs to perform this burn-in.

A naive method for sampling CIS's would be to perform a random walk on the graph, G, and then sample the k vertices most recently visited. This scheme is appealing because it has an easy to compute stationary distribution, and can 'inherit' the mixing rate from the random walk on G (which is relatively small). Despite these advantages, certain graphlet topologies, such as stars, will never be sampled, and modifications are needed to remedy this defect. [6] combined this basic idea with the SRW by maintaining a l length history of the SRW on CISs of size $k - l + 1$, and unioning the history, but this suffers from the same issues as SRW, such as slow mixing and the need to calculate global constants based on the CIS-graph.

[11] introduced a *Waddling* protocol which retains a memory of the last s vertices in the random walk on G and then extends this subgraph by $k - s$ vertices from either the first or last vertex visited in the s-subgraph (this extension is known as the 'waddle'). Waddling requires that one samples from the stationary distribution over G, but this can be achieved by selecting an edge uniformly at random from the graph, thus avoiding the burn-in. The authors provide recommendations for calculating the stationary distribution for this MCMC, and prove a bound on the error for the graphlet coefficients. The upside to this method is that the precise Waddling protocol used should depend on the desired graphlet, and the algorithm involves a rejection step which may lead to a loss of efficiency. This is simultaneously a downside of the method: the general specification of the method makes the algorithm implementation difficult. In contrast, lifting requires little tuning, perhaps at the expense of customizability. Finally, lifting has the advantage of never rejecting graphlets, has similar theoretical guarantees, and has simple parallel extensions.

3 SUBGRAPH LIFTING

The lifting algorithm is based on a randomized protocol of attaching a vertex to a given CIS. For any $(k-1)$-CIS, S, we lift it to a k-subgraph by adding a vertex from its neighborhood, $\mathcal{N}_v(S)$ at random according to some probability distribution. Note that this basic lifting operation can explore any possible subgraph in $\mathcal{V}_k$.

You can see an example of the lifting sampling scheme in Figure 2, where the algorithm iteratively builds a 4-CIS from a chosen node. First assume we have a node v_1 sampled from the distribution π_1, a base distribution that can be computed from local information (step (a)). We assume that $\pi_1(v) = \frac{f(\deg(v))}{K}$, where $f(x)$ is some function (usually a polynomial) and K is some global normalizing constant which is assumed to be precomputed. Denote $S_1 = \{v_1\}$. To start our procedure, sample an edge (v_1, v_2) uniformly from $\mathcal{N}_e(S_1)$ (step (b)). The vertex v_2 is then attached to S_1, forming a subgraph $S_2 = G|(V_{S_1} + v_2)$ (step (c)). After that, we sample another edge (v_i, v_3) (with $1 \le i \le 2$) uniformly from $\mathcal{N}_e(S_2)$, and the vertex v_3 is then attached to S_2 (steps (d-f)). At each step we sample an edge (v_i, v_{r+1}) (with $1 \le i \le r$) from $\mathcal{N}_e(S_r)$ uniformly at random, and attach the vertex v_{r+1} to the subgraph S_r (steps (g-h)). After

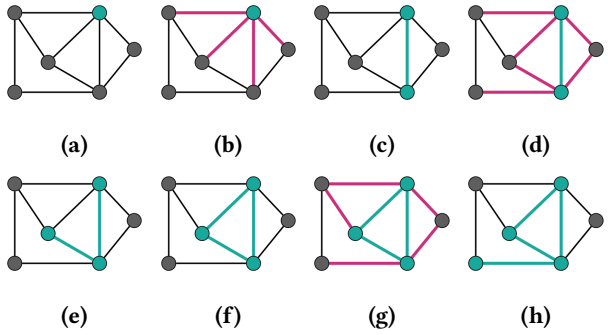

Figure 2: Lifting procedure

$k-1$ operations, we obtain a k-CIS, $T = S_k$. We'll refer to the procedure above as the *lifting procedure starting at vertex* v_1.

Once a k-CIS, T, has been sampled we need to classify its graphlet topology, $H_m \sim T$. Because lifting does not target specific graphlet topologies, we need to be prepared to modify the coefficient for any graphlet (the coefficients are elaborated on in the next section).

By induction, we can see that every k-CIS has a non-zero probability of being visited, assuming that π_1 is supported on every vertex. We consider two options for the starting vertex, π_1: uniform distribution over vertices, and the stationary distribution for a simple random walk on G. Lifting and waddling both can 'inherit' the mixing time of a simple random walk by initializing with the stationary distribution. In addition, lifting can be parallelized by having each thread start at a random vertex uniformly, while waddling requires us to start from the stationary distribution. In the next section, we show how to calculate the probability of sampling the k-CIS, $\pi(S)$, using only its local information.

3.1 Unordered lift estimator

We can recursively compute the marginal probability of sampling the graphlet, $\pi_U(T)$, for the lifted CIS $T \in \mathcal{V}_k(G)$. We say that this method is unordered because we ignore the order in which we visit the vertices in the graphlet. One advantage of this approach is that this probability is a function of only the degrees of vertices V_T. This can be done either recursively or directly. Throughout, let the set of vertices of T be $v_1, \ldots, v_k$.

We begin the algorithm by querying the probability of obtaining any vertex in T, $\pi_1(v_i), i = 1, \ldots, k$. We will build the probability of obtaining any connected subgraph of T inductively. This is possible because the probability of getting T via lifting is given by the sum $\pi_U(T) = \sum_S \mathbb{P}(T|S)\pi_U(S)$, where the sum is taken over all connected $(k-1)$-subgraphs $S \subset T$, and $\mathbb{P}(T|S)$ denotes the probability of getting from S to T in the lifting procedure. Then

$$\begin{aligned}\pi_U(T) &= \sum_{S \subset T} \pi_U(S) \frac{\deg_S(V_T \setminus V_S)}{|\mathcal{N}_e(S)|} \\ &= \sum_{S \subset T} \pi_U(S) \frac{|E_T| - |E_S|}{\sum_{u \in S} \deg(u) - 2|E_S|},\end{aligned} \tag{2}$$

where the sum is taken over all connected $(k-1)$-subgraphs $S \subset T$.

Consider the sampled k-CIS $T := S_k$. Denote the set of possible sequences $A = [v_1, \ldots, v_k]$ that would form T in the lifting process

as co(T). Notice that $S_r = G|\{v_1, \ldots, v_r\}$ must be a connected subgraph for all r. Thus,

$$\text{co}(T) = \big\{ [v_1, \ldots, v_k] \in V_G^k \mid \{v_1, \ldots, v_k\} = V_T, \\ T|\{v_1, \ldots, v_r\} \text{ is connected} \big\}. \quad (3)$$

Since the elements of co(T) are just certain orderings of vertices in T, we call an element from co(T) a *compatible ordering* of T. Note that $|\text{co}(T)|$ only depends on the type of the graphlet isomorphic to T, and it can be precomputed using dynamic programming. Thus, when $T \sim H_m$, the number of compatible orderings are equal: $|\text{co}(H_m)| = |\text{co}(T)|$. Note that $|\text{co}(H_m)|$ can vary from 2^{k-1} (for k-path) to $k!$ (for k-clique). For a direct formula, we notice that $\pi_U(T) = \sum_{A \in \text{co}(T)} \tilde{\pi}(A)$, and $\tilde{\pi}(A)$ is the probability of getting sequence $A \in \text{co}(T)$ in the lifting process (see (3),(8)). Then

$$\pi_U(T) = \sum_{A \in \text{co}(T)} \frac{f(\deg(A[1]))}{K} \prod_{r=1}^{k-1} \frac{|E_{S_{r+1}(A)}| - |E_{S_r(A)}|}{\sum_{i=1}^{r} \deg(A[i]) - 2|E_{S_r(A)}|}, \quad (4)$$

where, given $A = [v_1, \ldots, v_k]$, $A[i]$ is the ith vertex in A and $S_r(A) = G|\{v_1, \ldots, v_r\}$.

Although calculation of this probability on-the-fly is cost-prohibitive, we can greatly reduce the number of operations by noticing that the probability $\pi_k(T)$ is a function of degrees of the vertices: for a CIS T of type m, let $[v_1, \ldots, v_k]$ be an arbitrary labelling of the vertices of T with $d_i = \deg(v_i)$, then the probability of T is

$$\pi_U(T) = \frac{1}{K} F_m(d_1, \ldots, d_k)$$

for a cached function F_m given by (4).

Example. Consider a triangle, which is a 3-graphlet with edges (v_1, v_2), (v_2, v_3) and (v_1, v_3). Given the degrees d_1, d_2, d_3 of the corresponding vertices, the probability function is

$$\pi_U(\text{triangle}) = \left(\frac{\pi_1(d_1)}{d_1} + \frac{\pi_1(d_2)}{d_2} \right) \frac{2}{d_1 + d_2 - 2} \\ + \left(\frac{\pi_1(d_2)}{d_2} + \frac{\pi_1(d_3)}{d_3} \right) \frac{2}{d_2 + d_3 - 2} \\ + \left(\frac{\pi_1(d_3)}{d_3} + \frac{\pi_1(d_1)}{d_1} \right) \frac{2}{d_3 + d_1 - 2}. \quad (5)$$

Example. Consider a wedge, which is a 3-graphlet with edges (v_1, v_2) and (v_1, v_3). Given the degrees d_1, d_2, d_3 of the corresponding vertices, the probability function is

$$\pi_U(\text{wedge}) = \left(\frac{\pi_1(d_1)}{d_1} + \frac{\pi_1(d_2)}{d_2} \right) \frac{1}{d_1 + d_2 - 2} \\ + \left(\frac{\pi_1(d_1)}{d_1} + \frac{\pi_1(d_3)}{d_3} \right) \frac{1}{d_1 + d_3 - 2}. \quad (6)$$

We need to only compute functions F_m once before starting the algorithm. When a k-CIS T is sampled via lifting procedure, we find the natural labelling of vertices in T via the isomorphism $H_m \to T$, and use the function F_m together with the degrees $d_1, \ldots, d_k$ of vertices of T to compute the value of $\pi_U(T) = \frac{1}{K} F_m(d_1, \ldots, d_k)$.

3.2 Ordered lift estimator

The sample estimator, (1), does not track the order of the vertices as they are sampled to form a graphlet. We can, however, track the vertex information and thus define an estimator on ordered sequences of vertices $[v_1, \ldots, v_k]$, denoted by A. Given a sampling scheme of such sequences with probability $\tilde{\pi}(A)$, the estimator for graphlet counts is given by

$$\hat{N}_m(G) := \frac{\omega_m}{n} \sum_{i=1}^{n} \frac{\mathbb{1}(G|A_i \sim H_m)}{\tilde{\pi}(A_i)} \quad (7)$$

for some fixed weights ω_m. The main difference between these types of sampling is that we maintain the ordering of the vertices, while a CIS is an unordered set of vertices.

We can think of a lifting procedure as a way of sampling a sequence $A = [v_1, \ldots, v_k]$, ordered from the first vertex sampled to the last, that is then used to generate a CIS. Denote the set of such sequences as V_G^k. Let $S_r = G|\{v_1, \ldots, v_r\}$ be the r-CIS obtained by the lifting procedure on step r. The probability of sampling vertex v_{r+1} on the step $r+1$ is equal to

$$\mathbb{P}(v_{r+1}|S_r) := \frac{\deg_{S_r}(v_{r+1})}{|\mathcal{N}_e(S_r)|} = \frac{|E_{S_{r+1}}| - |E_{S_r}|}{\sum_{i=1}^{r} \deg(v_i) - 2|E_{S_r}|}.$$

Thus, the probability of sampling a sequence $A \in V_G^k$ is equal to

$$\tilde{\pi}(A) := \pi_1(v_1) \prod_{r=1}^{k-1} \mathbb{P}(v_{r+1}|S_r) \\ = \frac{f(\deg(v_1))}{K} \prod_{r=1}^{k-1} \frac{|E_{S_{r+1}}| - |E_{S_r}|}{\sum_{i=1}^{r} \deg(v_i) - 2|E_{S_r}|}. \quad (8)$$

Algorithm 1 Unordered Lift Estimator

input Graph G, graphlet size k
output $\hat{N}_m(G)$

```
For each k-graphlet in canonical form, H_m, precompute the function F_m(d_1, ..., d_k) and the global constant K
Initialize v at an arbitrary node, n ← 0, N̂_m(G) ← 0
while stopping criteria is not met do
    Sample initial vertex v from π_1(v)
    Initialize V_T ← {v} and E_T ← {}
    Initialize N_e(T) ← N_e(v)
    while |V_T| < k do
        Sample an edge e = (v, u) uniformly from N_e(T), with v ∈ V_T and u ∉ V_T
        Set E_T(u) ← {(v, u) ∈ N_e(T)}
        Update V_T ← V_T ∪ {u} and E_T ← E_T ∪ E_T(u)
        Query N_e(u)
        Update N_e(T) ← [N_e(T) ∪ N_e(u)] \ E_T(u)
    end while
    Set H_m = hash(T)
    Determine the ordering [v_1, ..., v_k] of vertices in V_T induced by the isomorphism (V_T, E_T) ~ H_m
    Set d_i = |N_e(v_i)| for all i = 1, ..., k
    Set π(T) = (1/K) F_m(d_1, ..., d_k)
    Update N̂_m(G) ← N̂_m(G) + π^{-1}(T)
    Update n ← n + 1
end while
Normalize N̂_m(G) ← (1/n) N̂_m(G)
```

Critically, this equation can be computed with only neighborhood information about the involved vertices, so it takes $O(k)$ neighborhood queries. Because there are many orderings that could have led to the same CIS T, then we need to apply proper weights in the graphlet count estimate (7) by enumerating the number of possible orderings.

We set up the estimator from (7) as

$$\hat{N}_{O,m} := \frac{1}{n}\frac{1}{|\mathrm{co}(H_m)|}\sum_{i=1}^{n}\frac{\mathbb{1}(G|A_i \sim H_m)}{\tilde{\pi}(A_i)}. \tag{9}$$

We call it the *ordered lift estimator* for the graphlet count.

Algorithm 2 Ordered Lift Estimator (with optional shotgun sampling)

input Graph G, graphlet size k
output $\hat{N}_m(G)$
  Count $|\mathrm{co}(H_m)|$- the number of compatible orderings in H_m.
  Initialize v at an arbitrary node, $n \leftarrow 0$, $\hat{N}_m(G) \leftarrow 0$
  while stopping criteria is not met **do**
    Sample v_1 from $\pi_1(v)$
    Initialize $V_S \leftarrow \{v_1\}$ and $E_S \leftarrow \{\}$
    Initialize $\mathcal{N}_e(S) \leftarrow \mathcal{N}_e(v_1)$
    Initialize $\pi(S) \leftarrow \pi_1(v_1)$
    while $|V_S| < k-1$ **do**
      Sample an edge $e = (v,u)$ uniformly from $\mathcal{N}_e(S)$, with $v \in V_S$ and $u \notin V_S$
      Set $E_S(u) \leftarrow \{(v,u) \in \mathcal{N}_e(S)\}$
      Update $\pi(S) \leftarrow \pi(S)\frac{|E_S(u)|}{|\mathcal{N}_e(S)|}$
      Update $V_S \leftarrow V_S \cup \{u\}$ and $E_S \leftarrow E_S \cup E_S(u)$
      Query $\mathcal{N}_e(u)$
      Update $\mathcal{N}_e(S) \leftarrow [\mathcal{N}_e(S) \cup \mathcal{N}_e(u)] \setminus E_S(u)$
    end while
    if not shotgun sampling **then**
      Sample an edge $e = (v,u)$ uniformly from $\mathcal{N}_e(S)$, with $v \in V_S$ and $u \notin V_S$
      Set $E_S(u) \leftarrow \{(v,u) \in \mathcal{N}_e(S)\}$
      Set $\pi(T) \leftarrow \pi(S)\frac{|E_S(u)|}{|\mathcal{N}_e(S)|}$
      Set $V_T \leftarrow V_S \cup \{u\}$ and $E_T \leftarrow E_S \cup E_S(u)$
      Set $H_m = \mathrm{hash}(T)$
      Update $\hat{N}_m(G) \leftarrow \hat{N}_m(G) + \pi^{-1}(T)$
    end if
    if shotgun sampling **then**
      for all $u \in \mathcal{N}_v(S)$ **do**
        Set $E_S(u) \leftarrow \{(v,u) \in \mathcal{N}_e(S)\}$
        Set $V_T \leftarrow V_S \cup \{u\}$ and $E_T \leftarrow E_S \cup E_S(u)$
        Set $H_m = \mathrm{hash}(T)$
        Update $\hat{N}_m(G) \leftarrow \hat{N}_m(G) + \pi^{-1}(S)$
      end for
    end if
    Update $n \leftarrow n+1$
  end while
  Normalize $\hat{N}_m(G) \leftarrow \frac{1}{n}\frac{1}{|\mathrm{co}(H_m)|}\hat{N}_m(G)$

A drawback of the algorithm is that it takes $k-1$ queries to lift the CIS plus the number of steps required to sample the first vertex (when sampled from Markov chain). To increase the number of samples per query, notice that if we sample $B = [v_1, \ldots, v_{k-1}]$ via lifting, we can get subgraphs induced by $A = [v_1, \ldots, v_{k-1}, u]$ for all $u \in \mathcal{N}_v(B)$ without any additional queries.

Thus, for each sampled sequence $B_i \in V_G^{k-1}$, we can compute the sum $\sum_{u\in\mathcal{N}_v(B_i)}\mathbb{1}(G|B_i\cup\{u\} \sim H_m)$ to incorporate the information about all k-CISs in the neighborhood of B_i. We call this procedure *shotgun sampling*. The corresponding estimator based on (7) is

$$\hat{N}_{S,m} = \frac{1}{n}\frac{1}{|\mathrm{co}(H_m)|}\sum_{i=1}^{n}\frac{\sum_{u\in\mathcal{N}_v(B_i)}\mathbb{1}(G|B_i\cup\{u\}\sim H_m)}{\tilde{\pi}(B_i)}. \tag{10}$$

Shotgun sampling produces more CIS samples with no additional query cost, but the CIS samples generated in a single iteration will be highly dependent. The following proposition states that the resulting estimators are unbiased (see Appendix for the proof).

Proposition 3.1. *The ordered lifted estimator, $\hat{N}_{O,m}$, and the shotgun estimator, $\hat{N}_{S,m}$, are unbiased for the graphlet counts N_m.*

4 LIFTING VARIANCE

One advantage of the lifting protocol is that it can be decoupled from the selection of a starting vertex, and our calculations remained agnostic to the distribution π_1 (although, we did require that it was a function of the degrees). There are two methods that we would like to consider: one is the uniform selection over the set of vertices and the other is from a random walk on the vertices, that presumably has reached its stationary distribution.

Consider sampling the starting vertex v independently and from an arbitrary distribution π_1 when we have access to all the vertices. The advantage of sampling vertices independently, is that the lifting process will result in independent CIS samples. A byproduct of this is that the variance of the graphlet count estimator (1) can be decomposed into the variance of the individual CIS samples. Given iid draws, the variance of the estimator $\hat{N}_m(G)$ is then

$$\begin{aligned} V_m^{\perp\!\!\!\perp}(\hat{N}_{U,m}) &:= \frac{1}{n}\mathrm{Var}\left(\frac{\mathbb{1}(T_n \sim H_m)}{\pi_U(T_n)}\right) \\ &= \frac{1}{n}\left(\sum_{T\in\mathcal{V}_k}\frac{\mathbb{1}(T\sim H_m)}{\pi_U(T)} - N_m(G)^2\right), \end{aligned} \tag{11}$$

which is small when the distribution of $\pi_U(T)$ is close to uniform distribution on $\mathcal{V}_m(G)$. Equation (11) demonstrates fundamental property that when $\pi_U(T)$ is small then it contributes more to the variance of the estimator. The variation in (11) can be reduced by an appropriate choice of π_1, i.e. the starting distribution.

For example, if $k = 3$, let $\pi_1(v) = \frac{1}{K}\deg(v)(\deg(v)-1)$, where $K = \sum_{u\in V_G}\deg(u)(\deg(u)-1)$. Then by (5) and (6)

$$\pi_U(\text{triangle}) = \frac{6}{K}, \quad \pi_U(\text{wedge}) = \frac{2}{K}.$$

Calculating K takes $O(|V_G|)$ operations (preparation), sampling starting vertex v takes $O(\log(|V_G|))$ operations, and lifting takes $O(\Delta)$, where Δ is the maximum vertex degree in G.

When we don't have access to the whole graph structure, a natural choice is to run a simple random walk (with transitional probabilities $p(i \to j) = \frac{1}{\deg(i)}$ whenever j in connected to i with an edge). Then the stationary distribution is $\pi_1(v) = \deg(v)/(2|E_G|)$,

and we can calculate all probabilities π_k accordingly. One feature of the simple random walk is that the resulting edge distribution is uniform: $\pi_U(e) = \frac{1}{|E_G|}$ for all $e \in E_G$ (edges are 2-graphlets). Therefore, the probabilities π_U are the same as if sampling an edge uniformly at random and start Lifting procedure from that edge.

4.1 Theoretical variance bound

As long as the base vertex distribution, π_1, is accurate then we have that the graphlet counts are unbiased for each of the aforementioned methods. The variance of the graphlet counts will differ between these methods and other competing algorithms such as Waddling and PSRW. The variance of sampling algorithms can be decomposed into two parts, an independent sample variance component and a between sample covariance component. As we have seen the independent variance component is based on the properties of π resulting from the procedure (see (11)). We have three different estimators: Ordered Lift estimator $\hat{N}_{O,m}$, Shotgun Lift estimator $\hat{N}_{S,m}$ and Unordered Lift estimator $\hat{N}_{U,m}$. For each estimator, we sample different objects: sequences $A_i \in V_G^k$ for Ordered, sequences $B_i \in V_G^{k-1}$ for Shotgun, and CISs $T_i \in \mathcal{V}_k(G)$ for Unordered estimator. Throughout this section, we will denote

(1) for the Ordered Lift estimator,

$$\phi_{O,i} = \frac{\mathbb{1}(G|A_i \sim H_m)}{|\mathrm{co}(H_m)|\tilde{\pi}(A_i)}, \tag{12}$$

(2) for the Shotgun Lift estimator,

$$\phi_{S,i} = \frac{\sum_{u \in \mathcal{N}_v(B_i)} \mathbb{1}(G|B_i \cup \{u\} \sim H_m)}{|\mathrm{co}(H_m)|\tilde{\pi}(B_i)}, \tag{13}$$

(3) for the Unordered Lift estimator,

$$\phi_{U,i} = \frac{\mathbb{1}(T_i \sim H_m)}{\pi_U(T_i)}. \tag{14}$$

Let ϕ_1 be shorthand for $\phi_{X,1}$, where $X \in \{O, S, U\}$, and note that $N_m(G) = \mathbb{E}\phi_1$, and $\hat{N}_m(G) = \frac{1}{n}\sum_i \phi_i$ for the corresponding estimators.

The variance can be decomposed into the independent sample variance and a covariance term,

$$\mathrm{Var}(\hat{N}_m(G)) = \frac{1}{n} V_m^{\perp\!\!\!\perp}(\phi_1) + \frac{2}{n^2} \sum_{i<j} \mathrm{Cov}\left(\phi_i, \phi_j\right). \tag{15}$$

For Markov chains, the summand in the second term will typically decrease exponentially as the lag $j - i$ increases, due to mixing. If we start from a random vertex then the samples are uncorrelated and the covariance term disappears. For an analysis of the mixing time for random walk-based graphlet Lifting, see the Appendix.

Let us focus on the first term, with the goal of controlling this for either choice of base vertex distribution, π_1, and the lifting scheme.

Theorem 4.1. *Let ϕ_1 be as defined in* (12), (13) *or* (14). *Denote the first k highest degrees of vertices in G as $\Delta_1, \ldots, \Delta_k$ and denote $D = \prod_{r=2}^{k-1}(\Delta_1 + \ldots + \Delta_r)$.*

(1) If π_1 is the stationary distribution of the vertex random walk then

$$V_m^{\perp\!\!\!\perp}(\phi_1) \leq N_m(G)\frac{2|E_G|}{|\mathrm{co}(H_m)|}D. \tag{16}$$

(2) If π_1 is the uniform distribution over the vertices then

$$V_m^{\perp\!\!\!\perp}(\phi_1) \leq N_m(G)\frac{2\Delta_1|E_G|}{|\mathrm{co}(H_m)|}D. \tag{17}$$

This result is comparable to analogous theorems for Waddling, [11], and PSRW, [21]. Critically, Lifting works without modification for all graphlets up to a certain size. It should be noted that the variance of each lift method has the same bound in Theorem 4.1. We do not observe significant differences between the empirical variances of the unordered and ordered lifts. The shotgun method does significantly reduce the observed variance, because it samples more graphlets per iteration, but due to the dependence between samples within a single lift, this is not reflected in the theory.

5 EXPERIMENTS

5.1 Description of experiments

All experiments were implemented on Amazon Web Services 't2.xlarge' instances running Ubuntu 16.04 (January 2019). All algorithms were implemented in Python, the code for which is available on GitHub[1]. Throughout our experiments we only compare against graphlet Monte Carlo sampling algorithms and do not compare against exact graphlet counting methods (except in computing a ground truth). This is consistent with our thesis, that Lifting can accurately compute graphlet coefficients with a moderate number of samples that only require neighborhood look-ups (as opposed to processing the whole graph and counting all graphlets).

We implemented our own Waddle and PSRW protocols, for clean comparisons. To get true count values, we used ESCAPE [14] for $k = 5$ and PGD [1] for $k = 3, 4$. All the methods were studied under the same number of iterations where they had comparable run times. The ground truth algorithms, ESCAPE and PGD, were faster than our estimation method, but these methods are limited to $k < 6$; we are aware of no exact counting method that does not hit the hard complexity barrier for large graphlet counts.

The Lifting method for k graphlets was implemented as follows. The initialization proceeds by pre-computing the probability functions F_m for every graphlet in the atlas of graphlets of size k and caching them symbolically through SymPy. The probability functions, π, are stored in a dictionary keyed by a canonical graph labeling string certificate generated by *nauty* [12] to reduce the cost of graph isomorphism checks. In every iteration of Lifting we: sample a random node, lift up to a k-node graphlet, get the cached probability function F_m by graph hashing, and, finally, find an isomorphism between the sampled graph and the canonical graph to obtain the probability of sampling the graphlet. Summing the inverses of these probabilities gives the estimate.

For our experiments, we picked five networks of different size, density, and domain [18][2]. The size of the graphs is listed in Table 3.

- The CELE network is a list of edges of the metabolic network of *C. elegans*.
- The EMAIL network is a university email exchange network.
- The CAIDA network is a network of packet routing relationships between AS's (e.g. Internet Service Providers).

[1] github.com/dshemetov/GraphletLift
[2] Network names correspond online datasets at networkrepository.com

Network name	$\|V_G\|$	$\|E_G\|$	Avg. Deg.
bio-celegansneural (CELE)	297	2,148	15
ia-email-univ (EMAIL)	1,133	5,451	9
misc-as-caida (CAIDA)	26,475	52,281	1.97
misc-fullb (FULLB)	199,187	5.7M	28.9
socfb-B-anon (SOCFB)	2.9M	20.9M	14

Figure 3: Networks used in experiments (M = millions).

- The FULLB network corresponds to a large positive definite matrix arising from a finite-element method.
- The SOCFB network is a network of user friendships on Facebook circa September 2005.

5.2 Comparisons on 4-graphlets

We performed a full comparison over all 4-graphlets (6 topologies), all networks (5 datasets), and three methods (unordered lift, PSRW, Waddle). Using the relative error between the estimate $\hat{N}_m$ and the ground truth N_m defined by

$$\text{Relative Error} = \frac{|\hat{N}_m(G) - N_m(G)|}{N_m(G)},$$

we can compare the performance of the algorithms on estimating each graphlet. Fixing iterations to 40K, we produced the relative errors for the algorithms across all graphs and all 4-graphlets in Figure 4. On the CELE graph, lifting outperforms on all graphlets. On the EMAIL graph, PSRW rivals lifting on some of the graphlets. Lifting has its worst performance on the CAIDA dataset, which the authors suspect is because the graph is extremely sparse and is mostly stars; rare graphlets, such as $H_6^{(4)}$, are difficult to detect for all methods. However, lifting is only the worst of the three methods on the 3-star graph for CAIDA. On the plus-side, lifting demonstrates the ability to find rare graphlets in large graphs, such as $H_4^{(4)}, H_5^{(4)}, H_6^{(4)}$ in SOCFB.

To get a sense for the convergence rates, we can plot the convergence to the true count as a function of iterations. We show this in Figure 5 for the 4-graphlets on the FULLB graph. Overall, we find comparable performance among the three algorithms on the 3-star, 4-tailed triangle, and the 4-clique. In some cases, such as $H_5^{(4)}$ and $H_4^{(4)}$, PSRW does not converge to the truth in the allotted number samples. This may be due to the mixing rate of PSRW, which was not fast enough, leading to bias in the estimated sampling probability. Waddle and lifting do approximately equally well on all the graphlets.

5.3 Comparisons on graphlets up to $k = 6$

We can compute the total variation distance between a graphlet frequency distribution ($\hat{N}_m$) and a target distribution (N_m) as

$$TV(\hat{N}_m, N_m) = \sum_m |\hat{N}_m - N_m|.$$

We compare the performance of PSRW and the Unordered Lift with this metric as a function of iterations on all the data sets, with $k = 3, 4, 5, 6$. This comparison is demonstrated in Figure 6. For $k = 5, 6$, as the ground truth is unavailable for these data sets (due to the inability of existing methods to handle such large graphlets), we track the convergence of the total variation difference between successive graphlet distribution estimates.

Network/Graphlet			Relative Error		
Network	Graphlet	Freq	Lift	PSRW	Waddle
CELE	$H_1^{(4)}$	0.4668	**0.0075**	0.0180	0.2153
	$H_2^{(4)}$	0.3703	**0.0024**	0.0301	0.1938
	$H_3^{(4)}$	0.1336	**0.0118**	0.0225	0.2055
	$H_4^{(4)}$	0.0113	**0.0063**	0.3241	0.1802
	$H_5^{(4)}$	0.0163	**0.0079**	0.1184	0.1978
	$H_6^{(4)}$	0.0014	**0.0077**	0.0865	0.1831
EMAIL	$H_1^{(4)}$	0.2865	**0.0009**	0.0083	0.1934
	$H_2^{(4)}$	0.5803	0.0062	**0.0014**	0.1587
	$H_3^{(4)}$	0.1137	**0.0058**	**0.0058**	0.2049
	$H_4^{(4)}$	0.0066	**0.0462**	0.3213	0.1585
	$H_5^{(4)}$	0.0108	**0.0239**	0.1656	0.2134
	$H_6^{(4)}$	0.0017	0.0498	**0.0369**	0.1113
CAIDA	$H_1^{(4)}$	0.9588	0.0313	**0.0038**	0.0132
	$H_2^{(4)}$	0.03505	0.0525	0.0891	**0.0126**
	$H_3^{(4)}$	0.0058	0.0774	**0.0740**	0.0883
	$H_4^{(4)}$	5e-05	0.0355	0.2219	**0.0134**
	$H_5^{(4)}$	0.0002	**0.0039**	0.6531	0.1996
	$H_6^{(4)}$	6.6e-06	0.6534	1.0000	**0.2524**
FULLB	$H_1^{(4)}$	0.1083	0.0161	**0.0030**	0.0842
	$H_2^{(4)}$	0.4858	**0.0038**	0.0348	0.0685
	$H_3^{(4)}$	0.2719	0.0102	**0.0059**	0.0684
	$H_4^{(4)}$	0.0065	**0.1035**	0.3928	0.1429
	$H_5^{(4)}$	0.0901	**0.0083**	0.1379	0.0575
	$H_6^{(4)}$	0.0372	0.0007	**0.0003**	0.0439
SOCFB	$H_1^{(4)}$	0.5283	0.1137	**0.0051**	0.3652
	$H_2^{(4)}$	0.4279	0.0815	**0.0094**	0.3622
	$H_3^{(4)}$	0.0393	0.1187	**0.0043**	0.3287
	$H_4^{(4)}$	0.0018	**0.1931**	0.3014	0.4095
	$H_5^{(4)}$	0.0022	**0.1172**	0.2383	0.2368
	$H_6^{(4)}$	0.0001	**0.0668**	0.0682	0.2652

Figure 4: Graphlet frequencies for all networks with relative error PSRW, Waddle, and Unordered Lifting after 40K graphlet samples (including rejections for Waddle).

The $k = 3, 4$ plots show PSRW outperforming Lift on the SOCFB network, while underperforming on the other data sets. We suspect this is because PSRW is adapted to sampling the 3-star, the most common graphlets in SOCFB; accordingly, PSRW performs well on the CAIDA set which is dominated by '3-star' graphlets. This suspicion is confirmed by the advantage lift has on datasets such as FULLB, which concentrates on the '4-path' graphlet instead of the star. In this case, PSRW has trouble converging. The $k = 5, 6$ plots demonstrate an approximately equivalent convergence rate between the methods. Both methods get fast initial gains by obtaining a good estimate of the most common graphlets, while

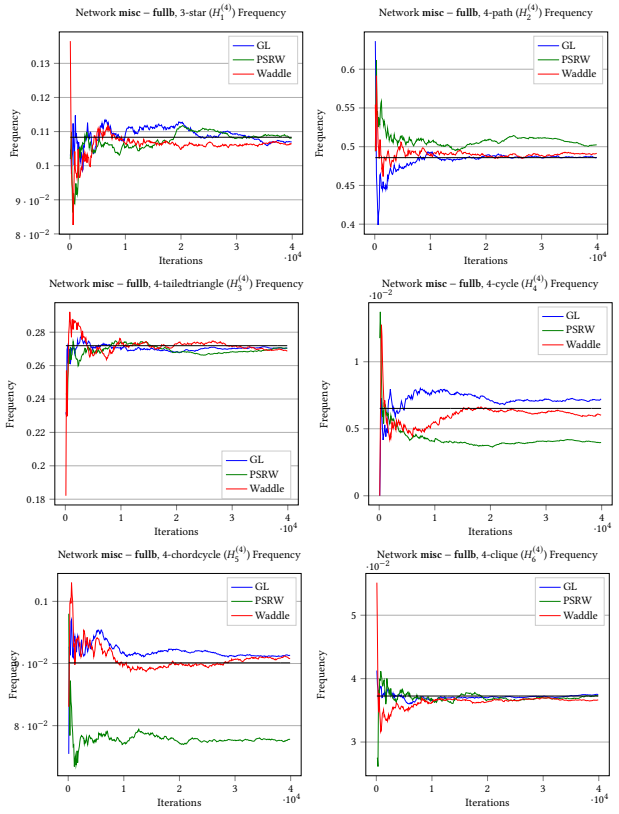

Figure 5: Convergence to the true frequency (shown in black) of the PSRW, Waddle, and Graphlet Lift (GL) methods.

the slow convergence that follows depends on sampling the rare graphlets. Note that PSRW demonstrates the correlation between its samples here by the 'plateau' pattern. (Note that we omitted Waddling from this comparison because in the case of size $k = 5, 6$ graphlets there was no clear extension of the Waddle protocol.)

We also compare the shotgun ordered Lifting relative error against Waddle for the 3-graphlets, the wedge ($H_1^{(3)}$) and the triangle ($H_2^{(3)}$). In Figure 7, we see that the shotgun procedure converges faster than Waddling in these cases. This advantage comes from shotgun's sampling of many graphlets essentially for free (with the same number of neighborhood queries), we consider all of the graphlets sampled from one shotgun sample to constitute one iteration. We have observed empirically, that although the shotgun approach produces batches of dependent samples, it is advantageous and we obtain faster convergence.

6 CONCLUSION

A reliable general purpose graphlet sampling algorithm is desireable because it can then be used out of the box without customizations and can scale to massive graphs. We provide three variants of the Lifting procedure: unordered, ordered, and the shotgun approach. We showed that the sampling probabilities in Lifting can be calculated from closed form, precomputed functions of the degree sequence of the subgraph. Lifting exemplifies the characteristics

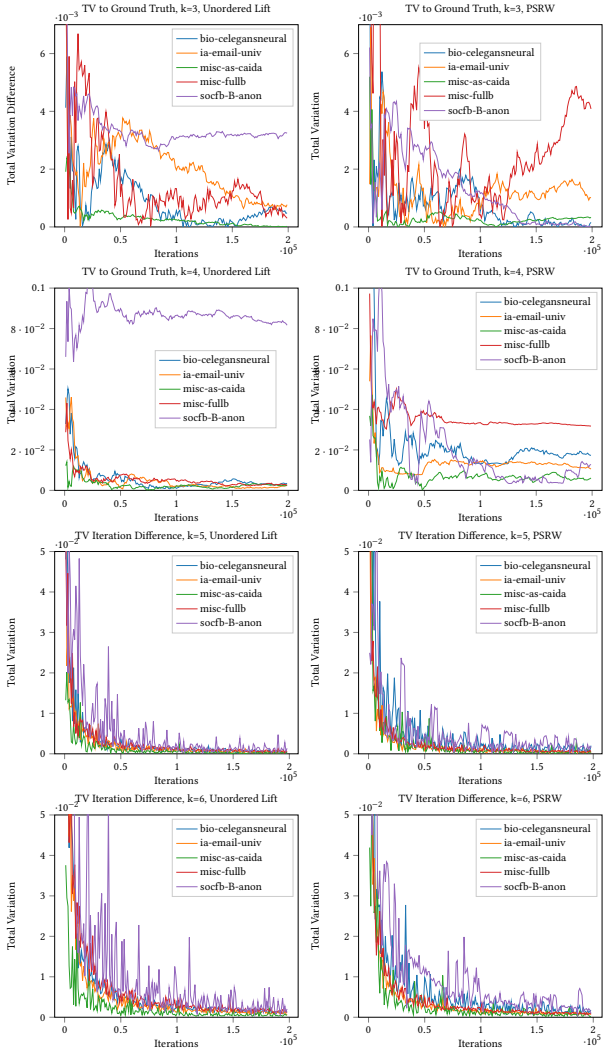

Figure 6: Here we compare the TV performance of Unordered Lift with the PSRW method on graphlets $k = 3, 4, 5, 6$. The top four figures show the total variation difference between the estimated counts and the ground truth as a function of iterations on all the data sets. The bottom four figures show the total variation between successive graphlet count estimates (i.e. $TV(\hat{N}_m(i-1), \hat{N}_m(i))$) by each method.

needed for a practical graphlet sampling method: it is easily parallelizable, samples all k-graphlets without modification, and can find rare graphlets. To the best of our knowledge, Lifting is the first graphlet sampling algorithm that enjoys each of these properties.

Our theoretical results bound the variance of Lifting estimated graphlet coefficients, which is based on the largest degrees in the graph. These results are comparable with the theoretical guarantees for PSRW (after sufficient mixing) and Waddling. Our experiments demonstrate that Lifting performs well in many cases, obtaining the lowest relative error, particularly for rare graphlets. We also see that the shotgun procedure can significantly boost the performance without additional neighborhood look-ups. We conclude by noting that Lifting is able to estimate the 5, 6-graphlet coefficients over a

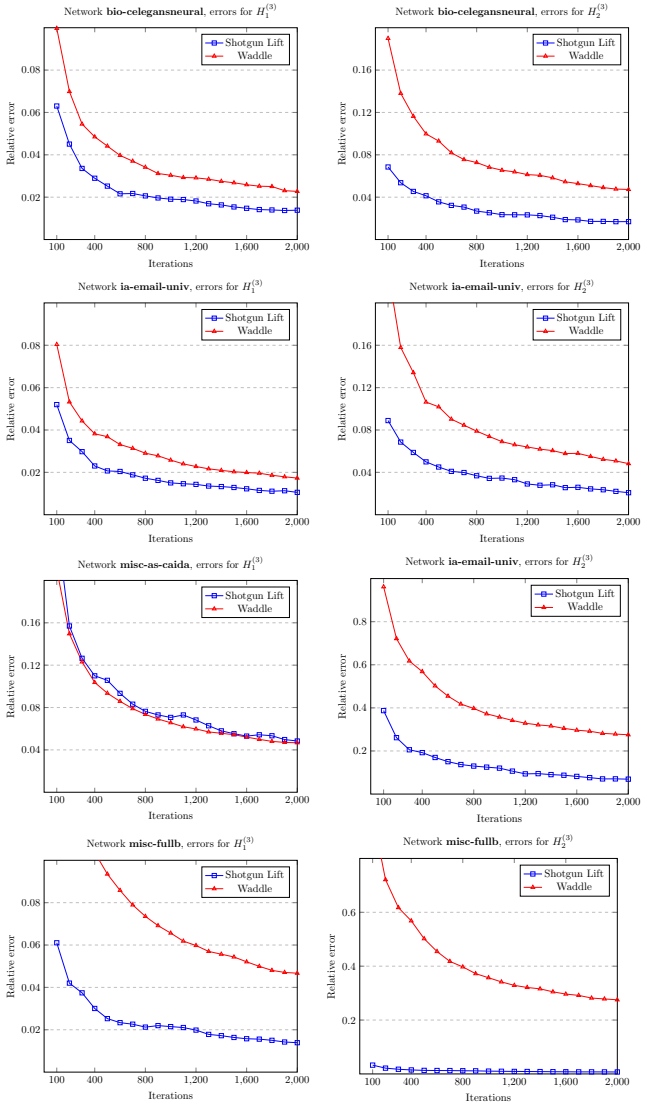

Figure 7: A comparison of shotgun-Lifting and Waddling for medium sized graphs, $H_1^{(3)}$ (wedge) and $H_2^{(3)}$ (triangle).

2.9M vertex graph and the solution converges in total variation in a moderate number of iterations.

ACKNOWLEDGMENTS

JS is supported by NSF DMS-1712996. We are grateful to Peter Dobcsányi for his open-source Pynauty package, which smoothed our Python implementation.

REFERENCES

[1] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, Nick G. Duffield, and Theodore L. Willke. 2017. Graphlet decomposition: framework, algorithms, and applications. *Knowledge and Information Systems* 50, 3 (01 Mar 2017), 689–722. DOI: http://dx.doi.org/10.1007/s10115-016-0965-5

[2] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *science* 286, 5439 (1999), 509–512.

[3] Mansurul A Bhuiyan, Mahmudur Rahman, and M Al Hasan. 2012. Guise: Uniform sampling of graphlets for large graph analysis. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*. IEEE, 91–100.

[4] Peter J Bickel, Aiyou Chen, Elizaveta Levina, and others. 2011. The method of moments and degree distributions for network models. *The Annals of Statistics* 39, 5 (2011), 2280–2301.

[5] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2017. Counting Graphlets: Space vs Time. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM '17)*. ACM, New York, NY, USA, 557–566. DOI: http://dx.doi.org/10.1145/3018661.3018732

[6] Xiaowei Chen, Yongkun Li, Pinghui Wang, and John Lui. 2016. A general framework for estimating graphlet statistics via random walk. *Proceedings of the VLDB Endowment* 10, 3 (2016), 253–264.

[7] James A Davis. 1970. Clustering and hierarchy in interpersonal relations: Testing two graph theoretical models on 742 sociomatrices. *American Sociological Review* (1970), 843–851.

[8] Ove Frank and David Strauss. 1986. Markov graphs. *Journal of the american Statistical association* 81, 395 (1986), 832–842.

[9] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[11] Guyue Han and Harish Sethu. 2016. Waddling Random Walk: Fast and Accurate Mining of Motif Statistics in Large Graphs. *2016 IEEE 16th International Conference on Data Mining (ICDM)* (2016), 181–190.

[12] Brendan D McKay and Adolfo Piperno. 2014. Practical graph isomorphism, II. *Journal of Symbolic Computation* 60 (2014), 94–112.

[13] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network motifs: simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827.

[14] Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. 2017. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1431–1440.

[15] Natasa Pržulj, Derek G Corneil, and Igor Jurisica. 2004. Modeling interactome: scale-free or geometric? *Bioinformatics* 20, 18 (2004), 3508–3515.

[16] N Pržulj, Derek G Corneil, and Igor Jurisica. 2006. Efficient estimation of graphlet frequency distributions in protein–protein interaction networks. *Bioinformatics* 22, 8 (2006), 974–980.

[17] Mahmudur Rahman, Mansurul Alam Bhuiyan, and Mohammad Al Hasan. 2014. Graft: An efficient graphlet counting method for large graph analysis. *IEEE Transactions on Knowledge and Data Engineering* 26, 10 (2014), 2466–2478.

[18] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. http://networkrepository.com

[19] Alistair Sinclair. 1992. *Improved bounds for mixing rates of Markov chains and multicommodity flow*. Springer Berlin Heidelberg, Berlin, Heidelberg, 474–487. DOI: http://dx.doi.org/10.1007/BFb0023849

[20] Tom AB Snijders. 2002. Markov Chain Monte Carlo Estimation of Exponential Random Graph Models. In *Journal of Social Structure*. Citeseer.

[21] 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, Article 8 (Sept. 2014), 27 pages. DOI: http://dx.doi.org/10.1145/2629564

[22] Stanley Wasserman and Philippa Pattison. 1996. Logit models and logistic regressions for social networks: I. An introduction to Markov graphs andp. *Psychometrika* 61, 3 (1996), 401–425.

[23] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *nature* 393, 6684 (1998), 440–442.