# Estimating Graphlet Statistics via Lifting

Kirill Paramonov
University of California, Davis
kir.paramonov@gmail.com

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

## ABSTRACT

Exploratory analysis over network data is often limited by our ability to efficiently calculate graph statistics, which can provide a model-free understanding of macroscopic properties of a network. This work introduces 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 which act locally by making 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. We outline three variants of lifted graphlet counts: the ordered, unordered, and shotgun estimators. We prove that our graphlet count updates are unbiased for the true graphlet count, have low correlation between samples, and have a controlled variance. 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 and James Sharpnack. 2018. Estimating Graphlet Statistics via Lifting. In *Proceedings of KDD conference (KDD'18).* ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.475/123_4

## 1 INTRODUCTION

In 1970, [**?** ] 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) [**?** ]. [**?** ] 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 [**?** ]. 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. In [**?** ], a pseudolikelihood method was used with graphlet counts to fit an ERGM designed to display transitivity, and Monte Carlo Markov Chain (MCMC) methods were developed in [**?** ] for fitting general ERGMs. Fitting such models from data can be cumbersome, and to do so implicitly assumes that the network follows such a model exactly. 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, [**?** ] 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. The clustering coefficient is also used to understand social network graphs, [**?** ]. More generally, 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, [**?** ]. 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 [**?** ]—or motifs.

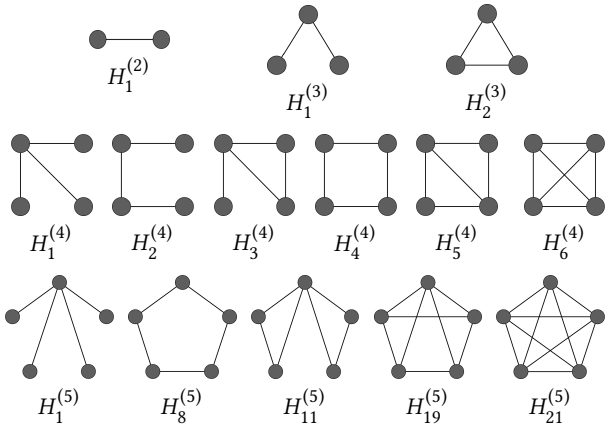


**Figure 1: Examples of graphlets**

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 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. We call this number the


*KDD'18, August 2018, London, UK*

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

*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 in the method of moments to fit certain graph models by selecting parameters that match the empirical graph moments to their expectations [**?** ]. Graphlet coefficients are used to understand biological networks, such as the protein-protein interaction network, and reoccuring patterns are thought to indicate evolutionary conserved modules [**?** ]. If the graphlet size, $k$, is small then testing for isomorphism, a problem called graph matching [**?** ], is feasible, but testing every induced subgraph can require on the order of $n^k$ iterations in its most naive implementation. We propose a class of methods called lifting that allow us to quickly estimate graphlet coefficients.

While there exist several algorithms that can estimate the proportion of triangles, wedges, and graphlets with four or five vertices (for example, [**? ?** ]), there are few algorithms that can efficiently estimate the proportion of larger graphlets. Many methods that can handle arbitrary graphlets are Monte Carlo sampling procedures that traverse through the space of all graphlets of a certain size within the large network. Two such methods are GUISE algorithm of [**?** ] and the pairwise subgraph random walk of [**?** ], which differ in the way that they perform a random walk between the CIS samples. Another option is to generate a sequence of vertices that induces a CIS sample, which has been done in [**?** ] using an algorithm called Waddling random walk. Alternatives to random Monte Carlo schemes include the color coding scheme of [**?** ], but it processes the whole graph, while the Monte Carlo schemes can traverse the network locally. In this work, we propose a new Monte Carlo algorithm, called lifting, for estimating the graphlet counts within a large network. The lifting step takes a smaller CIS of size $k-1$ and produces a subgraph of size $k$ by adding an adjacent vertex to it (according to a specific scheme). In this paper we consider procedures that start from vertices or edges and lift to sample CIS of size $k$. Lifting is a simple, flexible procedure that can be easily distributed to accommodate massive network datasets.

### 1.1 Our contributions

Graphlet coefficients are multipurpose statistical tools that can be used for model fitting and testing, network regression, and exploratory analysis for network data. Any CIS Monte Carlo sampling scheme has three goals: that it provides unbiased estimates of the graphlet coefficients, that the variance of the estimated coefficients is small, and that it does so in as few iterations as possible. Because massive graphs are often stored in distributed databases, we would like the sampling scheme to require only neighborhood queries (requests to the database returns the neighbors of a node) and we will avoid storing or processing the full graph. Because communication is often the bottleneck in distributed computing, neighborhood queries are the basic unit for measuring computational time complexity.

After discussing the precise requirements for any CIS sampling procedure, we will introduce the lifting scheme for subgraphs. The key difficulty in any Monte Carlo method for graphlet counting is calculating the sampling distribution. We provide two methods, the ordered lift estimator and the unordered lift estimator, which differ in the way that subgraphs are represented and counted in the graphlet count. The ordered estimator allows for a modification, called *shotgun sampling* that samples multiple subgraphs in one shot. For our theoretical component, we prove that the estimated graphlet coefficients are unbiased when the underlying MCMC has reached the stationary distribution (called perfect mixing). We also prove that under perfect mixing, the variance of the estimator scales like $\Delta^{k-2}$ where $\Delta$ is the maximum degree, and show that the lifting scheme can have significantly lower sample correlations than the subgraph random walk. All proofs can be found in the supplement. We conclude with real-world network experiments that reinforce the contention that subgraph lifting has a lower variance than Waddling and lower sample correlation than subgraph random walks.

## 2 SAMPLING GRAPHLETS

### 2.1 Definitions and notation

Throughout we will assume that our graph $G = (V, E)$ is simple, connected and undirected. For a subset $W \subseteq V$, a subgraph of $G$ induced by $W$, $G|W$ is a graph with vertices in $W$ and edges in $\binom{W}{2} \cap E$. We call a connected motif on $k$ vertices a $k$-graphlet. Given a $k$-graphlet $H$, we'll be interested in the number of $k$-subgraphs of $G$ isomorphic to $H$. 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 they are required to be connected—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, $\pi$, 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, $T_1, T_2, \ldots, T_n$.

The desire to sample subgraphs uniformly is natural, because the empirical counts will be unbiased estimates of their population quantities. Instead, suppose that our CIS sample, with $T_i \in \mathcal{V}_k, i = 1, \ldots, n$, is drawn with known sampling distribution $\pi$. Then we use Horvitz-Thompson inverse probability weighting to estimate the graphlet counts,

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

It is simple to see that this is an unbiased estimate of the graphlet counts as long as $\pi$ is supported over all elements of $\mathcal{V}_k$. Alternative updating methods include rejection sampling, which can be combined with inverse probability weighting, but we will use (1) for ease of presentation.

We find ourselves in a game, where we can choose any CIS Monte Carlo algorithm that induces a stationary distribution $\pi$, but we must be able to quickly and accurately compute $\pi$ in order to use (1). We will analyze the theoretical implications of the sampling algorithm based on mixing times of the Markov chain and the variance of the graphlet count estimates. Before we explore the lifting procedure, this paper's algorithmic contribution, we would like to discuss some existing MCMC CIS sampling methods.

The subgraph random walk is described in [**?** ], where they make a modification called the pairwise subgraph random walk (PSRW). In order to perform a random walk where the states are $\mathcal{V}_k$, 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, then we know that the stationary distribution will sample each edge in the CIS-relationship graph with equal probability. This fact enables [**?** ] 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.

The mixing time is a critical issue for any MCMC, and subgraph sampling is no exception. Dependence between consecutive samples results in a higher variability of $\hat{N}_m(G)$, and sufficient mixing is required for the stationary distribution to approximate the sampling distribution. It was pointed out in [**?** ] 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).

Another approach is to sample on ordered sequences of vertices $[v_1, \ldots, v_k]$, denoted by $A$, which would induce a $k$-CIS, $T = G|A$. Given a sampling scheme of such sequences with probability $\tilde{\pi}(A)$, the estimator for graphlet counts is given by

$$\hat{N}_m(G) := \frac{\omega_m}{n} \sum_{i=1}^{n} \frac{\mathbb{1}(G|A_i \sim H_m)}{\tilde{\pi}(A_i)} \quad (2)$$

for some fixed weights $\omega_m$. 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.

A naive method for sampling such sequences 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. [**?** ] 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.

[**?** ] 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'). The authors provide a method for calculating the stationary distribution for this MCMC, and prove a bound on the error for the graphlet coefficients. The downside 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. In contrast, the lifting scheme has the advantage of inheriting the mixing time of the random walk on $G$, and it can simultaneously sample many CISs of differing sizes without any rejections.

## 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 $\mathcal{V}_k$. In this work, we show that one can lift from a random walk on the vertices, or another vertex or edge sampling distribution, and achieve favorable properties.

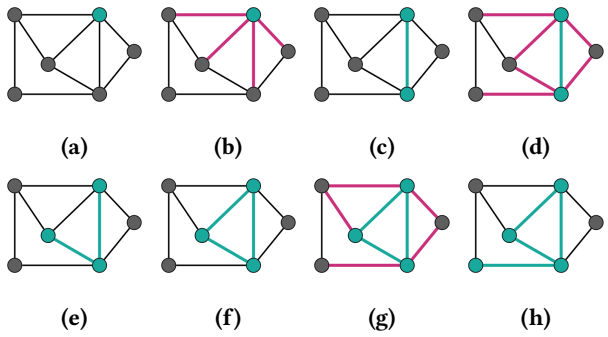


**Figure 2: Lifting procedure**

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 $\pi_1$, 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 $k-1$ operations, we obtain a $k$-CIS $T = S_k$. We'll refer to the procedure above as the *lifting procedure starting at vertex* $v_1$.

By induction, we can see that every $k$-CIS has a non-zero probability of being visited, assuming that $\pi_1$ is supported on every vertex. The Waddling method was motivated by the fact that a simple random walk would not visit all subgraphs in $\mathcal{V}_k$, yet Waddling only partially solved this issue, because not every wadding protocol can sample every $k$-graphlet. Typically, $\pi_1$ is assumed to be the stationary distribution of a simple random walk, but it could be another distribution such as uniform sampling over the vertices or edges. One motivation for lifting is that if we sample the vertices from a simple random walk, then lifting 'inherits' its mixing time, much like Waddling. Hence, lifting is a simple algorithm that can sample any CIS with good mixing rates and no rejections. It remains to be demonstrated that we can calculate the probability of sampling the $k$-CIS $\pi(S)$ using only local information.

### 3.1 Ordered lift estimator

We can think of a lifting procedure as a way of sampling a sequence $A = [v_1, \dots, v_k]$, 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 $S_r = G|\{v_1, \dots, v_r\}$ 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

$$\tilde{\pi}(A) := \pi_1(v_1) \prod_{r=1}^{k-1} \mathbb{P}(v_{r+1}|S_r) = \frac{f(\deg(v_1))}{K} \prod_{r=1}^{k-1} \frac{|E_{S_{r+1}}| - |E_{S_r}|}{\sum_{i=1}^{r} \deg(v_i) - 2|E_{S_r}|}. \quad (3)$$

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 (2) by enumerating the number of possible orderings.

Consider the sampled $k$-CIS $T := S_k$. Denote the set of possible sequences $A = [v_1, \dots, v_k]$ that would form $T$ in the lifting process as $\mathrm{co}(T)$. Notice that $S_r = G|\{v_1, \dots, v_r\}$ must be a connected subgraph for all $r$. Thus,

$$\mathrm{co}(T) = \Big\{ [v_1, \dots, v_k] \in V_G^k \mid \{v_1, \dots, v_k\} = V_T, \; T|\{v_1, \dots, v_r\} \text{ is connected} \Big\}. \quad (4)$$

Since the elements of $\mathrm{co}(T)$ are just certain orderings of vertices in $T$, we call an element from $\mathrm{co}(T)$ a *compatible ordering* of $T$. Note that $|\mathrm{co}(T)|$ 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: $|\mathrm{co}(H_m)| = |\mathrm{co}(T)|$. Note that $|\mathrm{co}(H_m)|$ can vary from $2^{k-1}$ (for $k$-path) to $k!$ (for $k$-clique). We set up the estimator from (2) as

$$\hat{N}_{O,m} := \frac{1}{n} \frac{1}{|\mathrm{co}(H_m)|} \sum_{i=1}^{n} \frac{\mathbb{1}(G|A_i \sim H_m)}{\tilde{\pi}(A_i)}. \quad (5)$$

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

**Algorithm 1** Ordered Lift Estimator (with optional shotgun sampling)

**input** Graph $G$, $k$-graphlet $H_m$
**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 $E_T \leftarrow E_S \cup E_S(u)$
      **if** $(V_T, E_T) \sim H_m$ **then**
        Update $\hat{N}_m(G) \leftarrow \hat{N}_m(G) + \pi^{-1}(T)$
      **end if**
    **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 $E_T \leftarrow E_S \cup E_S(u)$
        **if** $(V_T, E_T) \sim H_m$ **then**
          Update $\hat{N}_m(G) \leftarrow \hat{N}_m(G) + \pi^{-1}(S)$
        **end if**
      **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 $B = [v_1, \dots, v_{k-1}]$ via lifting, we can get subgraphs induced by $A = [v_1, \dots, v_{k-1}, u]$ 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 (2) is

$$\hat{N}_{S,m} = \frac{1}{n} \frac{1}{|\mathrm{co}(H_m)|} \sum_{i=1}^{n} \frac{\sum_{u \in \mathcal{N}_v(B_i)} \mathbb{1}(G|B_i \cup \{u\} \sim H_m)}{\tilde{\pi}(B_i)}. \quad (6)$$

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.

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 $N_m$.*

## 3.2 Unordered lift estimator

The ordered lift estimators computed the probability of sampling the CIS and the order of included vertices. Alternatively, we can compute the marginal probability of sampling the unordered graphlet, $\pi_U(T)$ for the lifted CIS $T \in \mathcal{V}_k(G)$. 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 $v_1, \dots, v_k$.

We begin the algorithm by querying the probability of obtaining any vertex in $T$, $\pi_1(v_i), i = 1, \dots, k$. 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

$$\pi_U(T) = \sum_{S \subset T} \pi_U(S) \frac{\deg_S(V_T \setminus V_S)}{|\mathcal{N}_e(S)|} = \sum_{S \subset T} \pi_U(S) \frac{|E_T| - |E_S|}{\sum_{u \in S} \deg(u) - 2|E_S|}, \quad (7)$$

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

For a direct formula, we notice that $\pi_U(T) = \sum_{A \in \mathrm{co}(T)} \tilde{\pi}(A)$, where $\mathrm{co}(T)$ is the set of compatible orderings of $T$ from previous section, and $\tilde{\pi}(A)$ is the probability of getting sequence $A \in \mathrm{co}(T)$ in the lifting process (see (4),(3)). Then

$$\pi_U(T) = \sum_{A \in \mathrm{co}(T)} \frac{f(\deg(A[1]))}{K} \prod_{r=1}^{k-1} \frac{|E_{S_{r+1}(A)}| - |E_{S_r(A)}|}{\sum_{i=1}^{r} \deg(A[i]) - 2|E_{S_r(A)}|}, \quad (8)$$

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

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 $[v_1, \dots, v_k]$ be an arbitrary labelling of the vertices of $T$ with $d_i = \deg(v_i)$, then the probability of $T$ is

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

for a cached function $F_m$ given by (8).

**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

$$\pi_U(\text{triangle}) = \left(\frac{\pi_1(d_1)}{d_1} + \frac{\pi_1(d_2)}{d_2}\right) \frac{2}{d_1 + d_2 - 2} + \left(\frac{\pi_1(d_2)}{d_2} + \frac{\pi_1(d_3)}{d_3}\right) \frac{2}{d_2 + d_3 - 2} + \left(\frac{\pi_1(d_3)}{d_3} + \frac{\pi_1(d_1)}{d_1}\right) \frac{2}{d_3 + d_1 - 2}. \quad (9)$$

**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

$$\pi_U(\text{wedge}) = \left(\frac{\pi_1(d_1)}{d_1} + \frac{\pi_1(d_2)}{d_2}\right) \frac{1}{d_1 + d_2 - 2} + \left(\frac{\pi_1(d_1)}{d_1} + \frac{\pi_1(d_3)}{d_3}\right) \frac{1}{d_1 + d_3 - 2}. \quad (10)$$

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 $d_1, \dots, d_k$ of vertices of $T$ to compute the value of $\pi_U(T) = \frac{1}{K} F_m(d_1, \dots, d_k)$.

## 3.3 Sampling a starting vertex

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 $\pi_1$ (although, we did require that it was a function of the degrees). There are two method 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 $\pi_1$ 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

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

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 $\pi_1$, i.e. the starting distribution.

**Algorithm 2** Unordered Lift Estimator

**input** Graph $G$, $k$-graphlet $H_m$
**output** $\hat{N}_m(G)$
  Set an ordering $[1,\dots,k]$ on the vertices of $H_m$, precompute the function $F_m(d_1,\dots,d_k)$ and the global constant $K$
  Initialize $v$ at an arbitrary node, $n \leftarrow 0$, $\hat{N}_m(G) \leftarrow 0$
  **while** stopping criteria is not met **do**
    Sample initial vertex $v$ from $\pi_1(v)$
    Initialize $V_T \leftarrow \{v\}$ and $E_T \leftarrow \{\}$
    Initialize $\mathcal{N}_e(T) \leftarrow \mathcal{N}_e(v)$
    **while** $|V_T| < k$ **do**
      Sample an edge $e = (v,u)$ uniformly from $\mathcal{N}_e(T)$, with $v \in V_T$ and $u \notin V_T$
      Set $E_T(u) \leftarrow \{(v,u) \in \mathcal{N}_e(T)\}$
      Update $V_T \leftarrow V_T \cup \{u\}$ and $E_T \leftarrow E_T \cup E_T(u)$
      Query $\mathcal{N}_e(u)$
      Update $\mathcal{N}_e(T) \leftarrow [\mathcal{N}_e(T) \cup \mathcal{N}_e(u)] \setminus E_T(u)$
    **end while**
    **if** $(V_T, E_T) \sim H_m$ **then**
      Determine the ordering $[v_1,\dots,v_k]$ of vertices in $V_T$ induced by the isomorphism $(V_T,E_T) \sim H_m$
      Set $d_i = |\mathcal{N}_e(v_i)|$ for all $i = 1,\dots,k$
      Set $\pi(T) = \frac{1}{K} F_m(d_1,\dots,d_k)$
      Update $\hat{N}_m(G) \leftarrow \hat{N}_m(G) + \pi^{-1}(T)$
    **end if**
    Update $n \leftarrow n+1$
  **end while**
  Normalize $\hat{N}_m(G) \leftarrow \frac{1}{n}\hat{N}_m(G)$

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 (9) and (10)

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

Calculating $K$ takes $O(|V_G|)$ operations (preparation), sampling starting vertex $v$ takes $O(\log(|V_G|))$ operations, and lifting takes $O(\Delta)$, where $\Delta$ 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 $\pi_k$ 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 $\pi_U$ are the same as if sampling an edge uniformly at random and start lifting procedure from that edge.

## 4 THEORETICAL ANALYSIS OF LIFTING

As long as the base vertex distribution, $\pi_1$, 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 $\pi$ resulting from the procedure (see (11)). The covariance component will be low if the samples are not highly dependent on the past, namely that the Markov chain is well mixed. 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,

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

(2) for the Shotgun Lift estimator,

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

(3) for the Unordered Lift estimator,

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

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,

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

For Markov chains, the summand in the second term will typically decrease exponentially as the lag $j-i$ increases, due to mixing. Let us focus on the first term, with the goal of controlling this for either choice of base vertex distribution, $\pi_1$, and the lifting scheme.

Theorem 4.1. *Let $\phi_1$ be as defined in* (12), (13) *or* (14). *Denote the first $k$ highest degrees of vertices in $G$ as $\Delta_1,\dots,\Delta_k$ and denote $D = \prod_{r=2}^{k-1}(\Delta_1+\ldots+\Delta_r)$.*
*(1) If $\pi_1$ is the stationary distribution of the vertex random walk then*

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

*(2) If $\pi_1$ is the uniform distribution over the vertices then*

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

This result is comparable to analogous theorems for Waddling, [**?** ], and PSRW, [**?** ].

When the vertices are sampled independently, the covariance term disappears, so we will focus on the sampling vertices via random walk in this subsection. One advantage of the lifting procedure over the SRW is that it inherits the mixing properties from the vertex random walk. This can be thought of as a consequence of the data processing inequality in that the lifted CISs are no more dependent then the starting vertices from which they were lifted. To that end, let us review some basics about mixing of Markov chains,

*Definition 4.2.* Define the mixing coefficient of a stationary Markov chain with discrete state space $X_t \in \mathcal{X}$ as

$$\gamma_X(h) = \frac{1}{2}\max_{x_1\in\mathcal{X}}\sum_{x_2\in\mathcal{X}}|\mathbb{P}(X_{t+h}=x_2, X_t=x_1) - \pi(x_1)\pi(x_2)|, \tag{18}$$

where $\pi(x)$ is the stationary distribution of the Markov chain. Also, define the mixing time of a stationary Markov chain $\{X_t\}$ as

$$\tau_X(\varepsilon) = \min\{h \mid \gamma_X(h) < \varepsilon\}. \tag{19}$$

Theorem 4.3. *[? ] Given stationary Markov chain $\{X_t\}$ with $\mu < 1$ being the second largest eigenvalue of the transitional matrix,*

$$\gamma_X(h) \leq e^{-(1-\mu)h}. \tag{20}$$

There are two consequences of mixing for CIS sampling. First, an initial burn-in period is needed for the distribution $\pi$ to converge to the stationary distribution (and for the graphlet counts to be unbiased). Second, by spacing out the samples with intermediate burn-in periods and only obtaining CISs every $h$ steps we can reduce the covariance component of the variance of $\hat{N}_m$. Critically, if we wish to wait for $h$ steps, we do not need to perform the lifting scheme in the intervening iterations, since those graphlets will not be counted. So, unlike in other MCMC method, spacing in lifted CIS sampling is computationally very inexpensive. Because burn-in is a one-time cost and requires only a random walk on the graph, we will suppose that we begin sampling from the stationary distribution, and the remaining source of variation is due to insufficient spacing between samples. The following theorem illustrates the point that the lifted MCMC inherits mixing properties from the vertex random walk.

Theorem 4.4. *Consider sampling a starting vertex from a random walk, such that a sufficient burn in period has elapsed and stationarity has been reached. Let $h$ be the spacing between the CIS samples, $D$ be defined as in Theorem 4.1, and $\mu$ be the second largest eigenvalue of the transition matrix for the vertex random walk. Let $\phi_i$ be as defined in* (12), (13) *or* (14)*, then*

$$|\mathrm{Cov}(\phi_i, \phi_{i+1}))| \leq 8N_m(G)|E_G|^2 e^{-(1-\mu)h} D.$$

Corollary 4.5. *In the notation of the Theorem 4.4,*

$$\frac{2}{n}\left|\sum_{i<j} \mathrm{Cov}\left(\phi_i, \phi_j\right)\right| \leq 8N_m(G)|E_G|^2 \frac{e^{-(1-\mu)h}}{1-e^{-(1-\mu)h}} D.$$

Hence, if we allow $h$ to grow large enough then we can reduce the effect of the covariance term, and our CISs will seem as if they are independent samples.

## 5 EXPERIMENTS

For our experiments, we picked five networks of different size, density and domain. All networks are available online in the network repository [? ]. The corresponding graphs are undirected, and were preprocessed to be simple and unweighted.

| Network information | | |
|---|---|---|
| Network name | $\lvert V_G\rvert$ | $\lvert E_G\rvert$ |
| **bio-celegansneural** | 297 | 2148 |
| **ia-email-univ** | 1133 | 5451 |
| **misc-polblogs** | 1224 | 16718 |
| **misc-as-caida** | 26475 | 52281 |
| **misc-fullb** | 199187 | 5754445 |

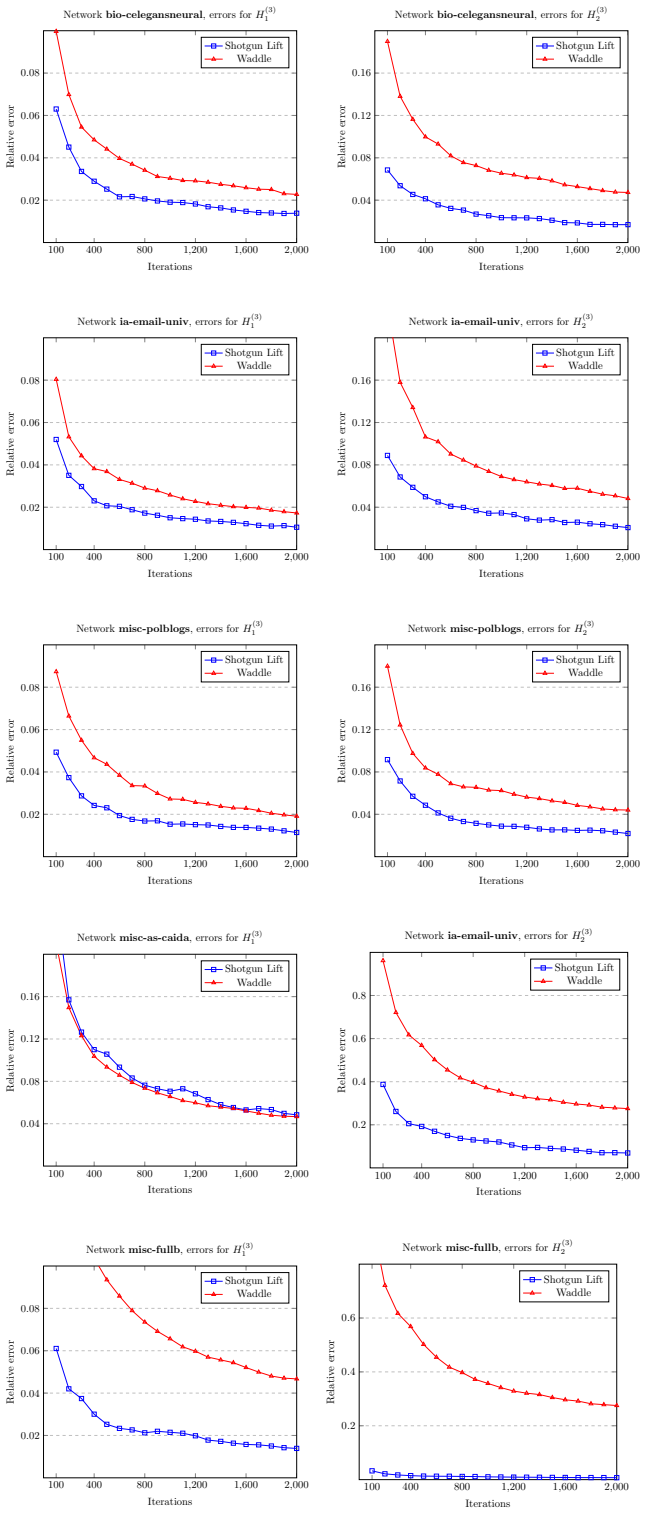


**Figure 3: Relative errors of count estimators for $H_1^{(3)}$ and $H_2^{(3)}$.**

We demonstrate performance of the shotgun and unordered lift method compared to two competitive methods: PSRW and Waddling. All algorithms were implemented in Python, and the code is available on GitHub[1].

We will compare the effectiveness of estimators in two ways:

(1) Relative error of a graphlet count estimator with $k = 3$ (wedges $H_1^{(3)}$ and triangles $H_2^{(3)}$) given limited number of queries (or, equivalently, limited number of samples).
(2) Variance and correlation of samples for a graphlet count estimator for 3-stars $H_1^{(4)}$ and 4-paths $H_2^{(4)}$.

In all methods, the first vertex of the lifting procedure is taken from a standard random walk on vertices. Whenever we specify the burn-in, that would correspond to the random walk steps taken to sample the starting vertex in case of Lifting and Waddling, and the number of steps taken between sampling CISs in case of PSRW. We compare the Shotgun Lift estimator against Waddling estimator for the first problem, and compare the Unordered Lift estimator against PSRW and Waddling for the second problem.

## 5.1 Relative Error given limited queries

The brute force method gives us the exact graphlet count for $k = 3$:

| Graphlet counts for $k = 3$ | | |
|---|---|---|
| Network name | $H_1^{(3)}$ | $H_2^{(3)}$ |
| **bio-celegansneural** | 4.408E+04 | 3.241E+03 |
| **ia-email-univ** | 8.038E+04 | 5.343E+03 |
| **misc-polblogs** | 1.038E+06 | 1.010E+05 |
| **misc-as-caida** | 1.479E+07 | 3.636E+04 |
| **misc-fullb** | 1.620E+08 | 6.021E+07 |

We will compare the relative error

$$\text{relative error} = \frac{|\hat{N}_m(G) - N_m(G)|}{N_m(G)}$$

between Shotgun Lift estimator and Waddling estimator with burn-in 3. The number of queries required for each sample $B \in V_G^{(2)}$ is 4 for the Shotgun Lift algorithm and the number of queries for each 3-CIS sample is 5 for the Waddling algorithm. We take the average error across 100 runs.

As we see from the plots in Figure 3, Shotgun Lift method outperforms Waddling by a factor of 2 in most cases. This agrees with our theory, since the number of actual CISs sampled by the Shotgun method is much bigger than Waddle given the same number of queries.

## 5.2 Variation and correlation of samples

We will use the Unordered Lift estimator, to count all motifs of size 4, but the performance would be measured in terms of the variance of the estimator, both the "variance under independence" and "covariance" parts. To get an idea about the distribution of 4-graphlets, we count the 4-graphlets with the Unordered Lift estimator.

Denote $\phi_i = \frac{\mathbb{1}(T_i \sim H_m)}{\pi(T_i)}$ for all estimators. Note that $\mathbb{E}\phi_1 = N_m(G)$. We compare the variance of the estimator $\hat{N}_m$ using equation (15). The table below shows estimates for $V_m^{\mathbb{1}}(\phi_1)$:

[1] github.com/KirillP23/LiftSRW

| Graphlet estimates for $k = 4$ | | | |
|---|---|---|---|
| Network name | $H_1^{(4)}$ | $H_2^{(4)}$ | $H_3^{(4)}$ |
| **bio-celegansneural** | 6.48E+05 | 5.16E+05 | 1.86E+05 |
| **ia-email-univ** | 5.40E+05 | 1.11E+06 | 2.20E+05 |
| **misc-polblogs** | 3.94E+07 | 3.10E+07 | 1.58E+07 |
| **misc-as-caida** | 7.82E+09 | 2.85E+08 | 4.62E+07 |
| **misc-fullb** | 1.07E+09 | 4.83E+09 | 2.71E+09 |

| Variation under independence | | | | |
|---|---|---|---|---|
| Network Name | ID | Lift | Waddle | PSRW |
| **bio-celegansneural** | $H_1^{(4)}$ | 6.876 | 11.064 | 0.652 |
| | $H_2^{(4)}$ | 5.470 | 3.956 | 5.185 |
| **ia-email-univ** | $H_1^{(4)}$ | 5.266 | 4.164 | 1.179 |
| | $H_2^{(4)}$ | 3.272 | 2.958 | 2.216 |
| **misc-polblogs** | $H_1^{(4)}$ | 5.033 | 7.210 | 0.820 |
| | $H_2^{(4)}$ | 6.214 | 5.369 | 6.093 |
| **misc-as-caida** | $H_1^{(4)}$ | 4.076 | 9.028 | 0.019 |
| | $H_2^{(4)}$ | 106.06 | 153.11 | 78.87 |
| **misc-fullb** | $H_1^{(4)}$ | 21.617 | 10.185 | 6.179 |
| | $H_2^{(4)}$ | 6.014 | 4.041 | 3.785 |

We can see that PSRW generally has smaller variation under independence, mostly because of the probability $\pi(T)$ being independent of the degrees of vertices of $T$. On the other hand, variation under independence of Lift and Waddle methods is comparable, meaning that $\pi(T)$ varies similarly for those two methods. Next, we compare the dependence of $\phi_i$ and $\phi_{i+1}$ using correlation for different values of the burn-in $h$ (see Fig.4). For Lift and Waddling, the burn-in between $\phi_i$ and $\phi_{i+1}$ is the number of steps taken after sampling $T_i$ to get a new starting vertex for $T_{i+1}$. For PSRW, burn-in is the number of steps between CIS samples in the random walk on subgraphs. From the graphs in Figure 4, we see that PSRW produces highly correlated samples compared to Lift and Waddling methods. This agrees with our analysis of PSRW, since it takes many more steps for the subgraph random walk to achieve desired mixing compared to the random walk on vertices.

# 6 CONCLUSION

We introduced a flexible methodology for sampling graphlets, called Lifting. Our experimental results demonstrate our hypothesis that lifting can have smaller variance than Waddling, and better mixing rates than Subgraph Random Walking. This was reinforced by our theoretical results that control the variance of and covariance between graphlet updates. We anticipate that this lifting scheme will have more far reaching implications, as we may apply graphlet sampling to supervised learning over graphs and other machine learning applications.

**Acknowledgements:** JS is supported by NSF DMS-1712996.

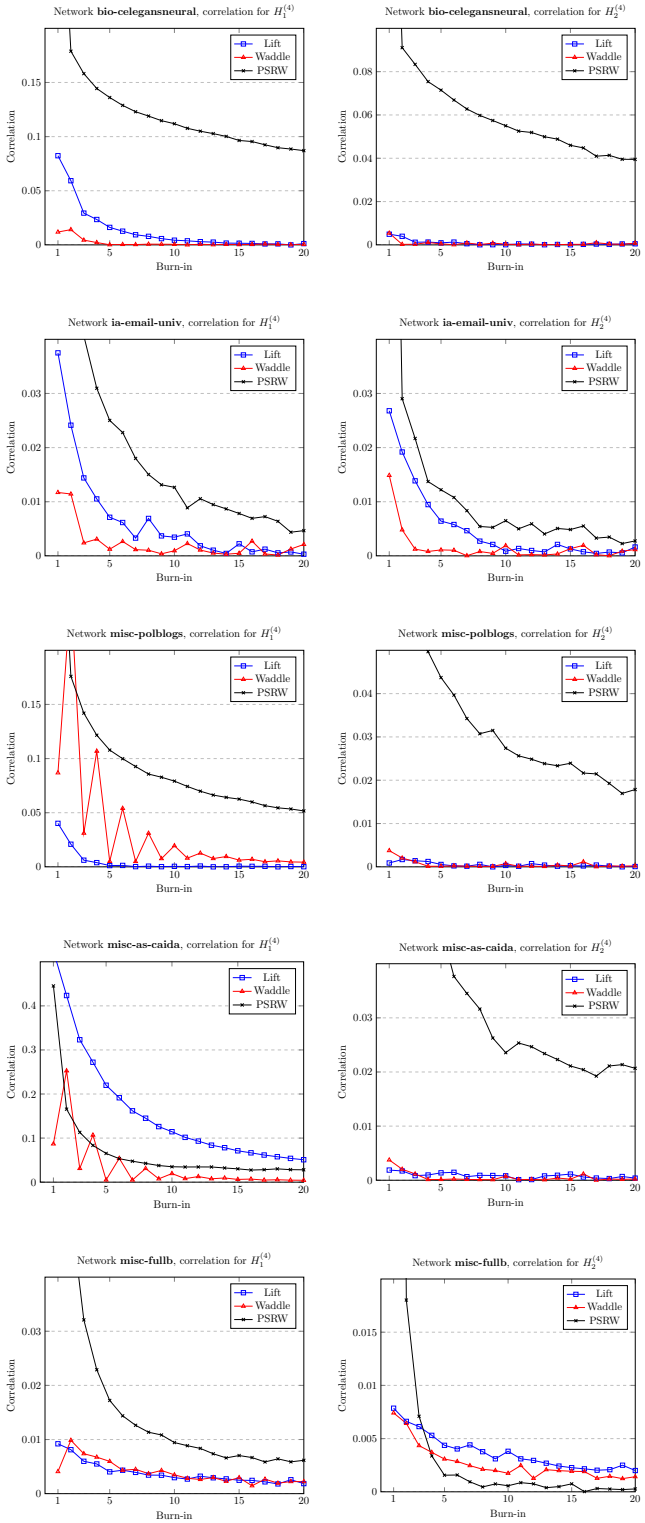


**Figure 4: Correlation of $\phi_i$ and $\phi_{i+1}$ depending on the intermediate burn-in time, $h$, between samples for graphlets $H_1^{(4)}$ and $H_2^{(4)}$.**