Sampling Triples from Restricted Networks using MCMC Strategy

Mahmudur Rahman, and Mohammad Al Hasan
Dept. of Computer Science, Indiana University—Purdue University, Indianapolis
{mmrahman, alhasan}@cs.iupui.edu

Rahman, M., & Hasan, M. A. (2014). Sampling Triples from Restricted Networks Using MCMC Strategy. In Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management (pp. 1519–1528). New York, NY, USA: ACM. http://doi.org/10.1145/2661829.2662075

ABSTRACT

In large networks, the connected triples are useful for solving various tasks including link prediction, community detection, and spam filtering. Existing works in this direction concern mostly with the exact or approximate counting of connected triples that are closed (aka, triangles). Evidently, the task of triple sampling has not been explored in depth, although sampling is a more fundamental task than counting, and the former is useful for solving various other tasks, including counting. In recent years, some works on triple sampling have been proposed that are based on direct sampling, solely for the purpose of triangle count approximation. They sample only from a uniform distribution, and are not effective for sampling triples from an arbitrary user-defined distribution. In this work we present two indirect triple sampling methods that are based on Markov Chain Monte Carlo (MCMC) sampling strategy. Both of the above methods are highly efficient (several magnitudes faster on large datasets) compared to a direct sampling-based method, specifically for the task of sampling from a non-uniform probability distribution. Another significant advantage of the proposed methods is that they can sample triples from networks that have restricted access, on which a direct sampling based method is simply not applicable.

1. INTRODUCTION

For a long time, scientists from a wide variety of disciplines, including social sciences,information science and bioinformatics are using networks for modeling complex relations among different entities. They also invented various measures of interest relating to graph topology for understanding the underlying dynamics that drive these relations. For instance, clustering coefficient and transitivity [20] metrics are invented for denoting the clustering tendency of the vertices in a network, centrality indices are used for understanding the diverse roles of the vertices, and modularity is used for discovering communities. Most of the existing methods for computing the above metrics have a super-linear time complexity, which is deemed costly for today's large networks reaching the planetary scale in size. So, there have been growing interests to obtain efficient algorithms for computing these metrics in linear or sub-linear time through sampling based methods.

Triples, which are defined as paths of length two are important building blocks of social networks. Two prominent theories for temporal evolution of social networks are *homophily* and *transitivity*; according to the homophily theory, people tend to choose friends that are similar (to some extent) to themselves and according to the transitivity, people who have common friends tend to become friends themselves [12]. Transitivity accounts for a social network to have a large number of triangles than a random network of similar size. To measure the conformity of a network with the transitivity theory, network metrics, such as, transitivity, or clustering co-efficient are used. They simply capture the tendency of "closing the triangle" by adding the missing edge in an open triple. This tendency is more prevalent inside a network community as the nodes in a community connect more often with other members in the community. Thus, the role of a triple in defining the network structure makes it an important entity while analyzing a social network.

The importance of transitivity and its obvious connection to the triangles in a network also contributed to the regained interest in the seemingly simple task of triangle counting [9]. Besides computing transitivity in social networks, there are other usages of the triangle counting task. In [6], the distribution of triangles is used to uncover hidden thematic structure in the World Wide Web. Bar-Yossef et al. [2] show that triangle count can be used for query plan optimization in databases. In another recent work [3], the authors show that the distribution of local triangle count can be used as features for assessing the content quality in social networks. However, for large networks with millions of vertices and edges, all the exact methods for triangle counting can be deemed as expensive; so, the majority of the recent efforts of triangle counting either adopt a method for approximate counting [19] or design a parallel or distributed framework for solving the counting task in sub-linear time [18].

Surprisingly, a relatively small number of the existing works consider sampling of triples (or triangles); nonetheless, sampling is more fundamental than counting as the earlier can be used for solving various tasks, including counting. Schank et al. [16] proposed one of the earliest works for uniform sampling of triples, which they use for approximating the transitivity index of a network. The same method is also used by another recent work for triangle count approximation [17]. Both of these works adopt a direct sampling strategy for which critical information regarding the network, such as vertex count, and the degree of each of the vertices are required before the sampling can be commenced; unfortunately, for a restricted network such information are not available. Besides, these methods only perform uniform sampling of triples for approximating triangle counting or

transitivity, whereas in this work we are considering sampling triples from an arbitrary distribution.

1.1 Objective and Motivation

Our objective is to obtain an efficient method for sampling triples from an arbitrary (user defined) probability distribution (say, f) defined over the set of triples in a network. The distribution f can be defined implicitly; for instance, one can only define a weighting function $w(\cdot)$ over the set of triples, and f is simply the probability vector obtained from the weights of each of the triples. Any locally computable weight function should be admissible. Such a function can be formed by the topological properties of the vertices in the triples, or in case, the graph contains vertex or edge labels, the weight function can be designed based on the label composition of the vertices or edges of the triples.

To obtain a direct method for the sampling task that is defined in the above paragraph, we first need to compute f (probability mass function) from the weight function, and then obtain the *cmf* (cumulative mass function) of f. Obviously, this requires the knowledge of the entire sampling space (total number of vertices, edges, and triples). For a restricted graph, which can only be crawled by following the edges of the input network, such information is not available, so direct sampling is infeasible for solving the above sampling task on a restricted network.

The motivation for considering a restricted network comes from real-life consideration. Say, an analyst is using a crawler for crawling a Web graph, and he does not have the resources to store the entire graph in memory/disk. Under this setting, he may want to sample a set of triples (from a uniform distribution) alongside crawling so that he can approximate the transitivity of the Web graph. Clearly, without storing the entire network, he has no knowledge of the number of vertices, or edges in this network, let alone the number of triples. Also, for a hidden network, a user may not have access to an arbitrary node in the network for security reason, rather the desired node can only be accessed from another node which is one-hop away from it; such scenarios are common in real-life and are considered in some of the recent works that compute various network properties (degree distribution, average degree) by random walk over real-life networks, such as, Facebook [8].

Even if a network is not restricted, an indirect sampling method can be more desirable than a direct sampling method, both from viability and efficiency consideration. We will show in this paper that an MCMC based method is significantly more efficient than a direct sampling method, for weighted sampling where the probability distribution vector (f) is not readily available, because in such a case, computation of f and *cmf*(f) (cumulative mass function of f) are required for direct sampling (see Section 4.2 for details); this fixed cost can be expensive, as the number of triples in large networks are, typically, in the order of billions.

In this work, we propose two methods for indirect triple sampling using Markov Chain Monte Carlo (MCMC) strategy. MCMC performs a random walk over the sample space such that the desired probability distribution (in this case, f) aligns with the stationary distribution of the random walk. Since MCMC computes the transition probability matrix of the random walk locally (on demand), it does not compute f explicitly; consequently, it does not need any information regarding the size of the sample space. As long as a state of the random walk can be visited from one of the neighboring states, an MCMC-based sampling works, which makes it an ideal candidate for sampling from a restricted network. Also, an MCMC-based method computes the transition probability matrix on-line, so it can accommodate addition or deletion of vertices (or edges) in a dynamic network, even when the sampling process is running.

The sampling methods that we propose are called vertex-MCMC, and triple-MCMC: the former is more accurate and the latter is more versatile. Both the methods can sample from an arbitrary distribution, yet vertex-MCMC is particularly suitable (both efficient and accurate) for sampling from a uniform distribution. So, we use it for approximating triangle count in a large network. In experiment section, we show that the performance of vertex-MCMC is almost as good as a direct sampling based method. On the other hand, triple-MCMC method is more suitable for sampling from non-uniform distribution; our experiments with one of the real-life graph show that it is 170 times faster than a direct sampling method with a better sampling quality.

2. RELATED WORKS

The popularity of sampling based techniques has grown in recent years for analyzing large graphs. For example, sampling has been used for finding interesting subgraphs [1], communities [11], and graphlet frequency distribution [15].

Triple sampling is considered in the context of approximate counting of triangles (or computing transitivity) in the following works [16, 5, 17]. Both [16] and [17] obtain uniform sampling of triples using a direct sampling method, which we will discuss in Section 4.2. Buriol et al. [5] propose a collection of streaming algorithms for triple sampling, also with the intention of triangle approximation. One of their methods, named, 3-pass-incident-stream, is conceptually similar to the direct sampling method of [16, 17]. Buriol et al. also consider another 3-pass method for arbitrary edge streaming; it samples triples by first sampling an edge, and then sampling a vertex, both uniformly. A triple that is obtained this way belongs to one of the following sets exclusively: disconnected triples (set T_1), connected open triples (set T_2), or triangles (set T_3). From the size of each of these sets, the authors find an approximation of the triangle count in a graph. To the best of our knowledge, no works exist that consider sampling of triples from a user-defined arbitrary sampling distribution.

A set of recent works [10, 8] considers the task of sampling from restricted networks that can only be crawled. The most notable among these is the work by Leskovec and Faloutsos [10] which used a collection of random walk methods, namely, BFS (breadth-first search), forest-fire, simple random walk (SRW), and snowball sampling for obtaining a representative sample of the restricted network. One can apply the above random walk methods for sampling a representative networks of appropriate size and return all the triples from that network as the sampled triples. However, such a sampling of triples does not guaranty uniform sampling of triples; furthermore, the user has no control over the probability distribution by which the triples would be sampled in the above approach. On the other hand, the MCMC method that we propose in this work can sample from any arbitrary user-defined distribution.

There are other recent works that adopt MCMC sampling strategy. Bhuiyan et al. [15] use it for sampling graphlets,

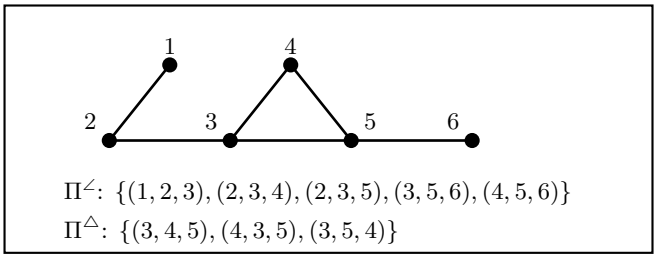

Figure 1: Open and closed triples in a graph

Maiya et al. [11] use it for sampling community structure, and Gjoka et al. [8] use it for finding an approximate degree distribution of Facebook network. However, each of these works have a different objective and they sample from different population. Besides, none of these works samples from an arbitrary user-defined distribution.

3. BACKGROUND

Let $G = (V, E)$ is a graph, where V is the set of vertices and E is the set of edges. Each edge $e \in E$ can be denoted by a pair of vertices (u, v) where, $u, v \in V$. A graph without a self-loop or multi edge is a simple graph. In this work, we consider simple, connected, and undirected graphs. We use n to define the number of vertices in G, $d(v)$ to define the degree of a node v, and d_{max} to denote the maximum degree value for a vertex over the entire graph.

3.1 Triples and Triangles

A triple (u, v, w) at a vertex v is a path of length two for which v is the center vertex. If the other two vertices (u and w) are also connected by an edge, the triple is called a closed triple (triangle), otherwise it is called an open triple. A triangle actually contains three closed triples, one centered on each of its vertices.

We use the symbol Π_v to represent the set of triples that are centered at the vertex v. The set of triples in a graph $G = (V, E)$ is Π, which is the union of the set of triples at each of its node, i.e., $\Pi = \bigcup_{v \in V} \Pi_v$. Based on whether the triple is open or closed (in terms of its induced embedding in the graph G), we can partition the set Π into $\Pi^{\angle}$ (open triples) and $\Pi^{\triangle}$ (closed triples). Note that, each of the nodes of a triangle in a graph G contributes one distinct triple in the set $\Pi^{\triangle}$. To represent the set of open and closed triples centered at a vertex v, we will use $\Pi_v^{\angle}$ and $\Pi_v^{\triangle}$, respectively. If $\delta(G)$ is the number of triangles in the graph G, then

$$\delta(G) = \frac{1}{3}|\Pi^{\triangle}| = \frac{1}{3}\sum_{v \in V}|\Pi_v^{\triangle}| \tag{1}$$

Example: Graph in Figure 1 has eight triples. Five of them are open and the remaining three are closed. Also the graph has exactly one triangle. □

Given a graph $G(V, E)$, the total number of triples in G,

$$|\Pi| = \sum_{v \in V}|\Pi_v| = \sum_{v \in V}\binom{d(v)}{2} \tag{2}$$

3.2 Transitivity

Newman, Watts and Strogatz [14] defined the transitivity of a graph G (say, $\gamma(G)$) as the fraction that represents the number of closed triples divided by the number of all the triples over the entire network.

$$\gamma(G) = \frac{|\Pi^{\triangle}|}{|\Pi|} = \frac{|\Pi^{\triangle}|}{|\Pi^{\angle}| + |\Pi^{\triangle}|} \tag{3}$$

Using Equation 1 and Equation 3, the triangle count ($\delta(G)$) of a network can be obtained from the transitivity of the network as below:

$$\delta(G) = \frac{1}{3} \cdot \gamma(G) \cdot |\Pi| \tag{4}$$

Following Equation 3, the transitivity of a graph, $\gamma(G)$, is the probability that an arbitrary triple in G is closed. This probabilty can be approximated using uniform triple sampler. For this, we sample a set of triples Ω ($\subset \Pi$) from G using a uniform distribution, and count the number of closed tripled in that set (say, $\Omega^{\triangle}$). Then, we define a random variable $\gamma_a(G) = \frac{|\Omega^{\triangle}|}{|\Omega|}$. The following lemma holds:

LEMMA 1. $E[\gamma_a(G)] = \gamma(G)$

PROOF: *form the uniformity assumption,* $E\left[|\Omega^{\triangle}|\right] = \gamma(G) \cdot |\Omega|$. *Then,* $E[\gamma_a(G)] = E\left[\frac{|\Omega^{\triangle}|}{|\Omega|}\right] = \frac{E[|\Omega^{\triangle}|]}{|\Omega|} = \frac{\gamma(G)\cdot|\Omega|}{|\Omega|} = \gamma(G)$. □

Thus, the expectation of the variable $\gamma_a(G)$ provides an unbiased estimate of the transitivity, which can subsequently be used in Equation 4 for finding an approximate triangle count in the graph G.

3.3 Metropolis-Hastings (MH) Algorithm

The main goal of the Metropolis-Hastings algorithm is to draw samples from some distribution $\pi(x)$, called the *target distribution*, where, $\pi(x) = f(x)/K$; here K is a normalizing constant which may not be known and difficult to compute. MH algorithm can be used together with a random walk to perform Markov Chain Monte Carlo (MCMC) sampling. For this, the MH algorithm draws a sequence of samples from the target distribution as follows:
i. It picks an initial state (say, x) satisfying $f(x) > 0$.
ii. From current state x, it samples a point y using a distribution $q(x, y)$, referred as *proposal distribution*.
iii. Then, it calculates the *acceptance probability*,

$$\alpha(x, y) = \min\left(\frac{\pi(y)q(y, x)}{\pi(x)q(x, y)}, 1\right) = \min\left(\frac{f(y)q(y, x)}{f(x)q(x, y)}, 1\right) \tag{5}$$

and accepts the proposal move to y with probability $\alpha(x, y)$. The process continues until the Markov chain reaches to a stationary distribution.

4. METHOD

4.1 Problem Formulation

Assume, Π is the set of triples in a large network G. Now, for a user defined non-negative weight function, $w : \Pi \to \mathbb{R}^+$, we can define a probability distribution over the set of triples (Π) by normalizing the weights, i.e, for a triple $t \in \Pi$, its probability is assigned as $\frac{w(t)}{\sum_{x \in \Pi} w(x)}$. The task of triple sampling is to sample triples from Π using the above probability distribution. We can represent the probability distribution using a probability mass function, f, which simply assigns a probability value to each of the triples in Π. If the weights of all the triples are the same, then the above sampling becomes

a uniform sampling of triples. For triple sampling, we also consider the scenario that the given network is restricted such that it is not explicitly visible, but can be crawled. More formally, in a restricted network, we can perform a random walk over the network, where at any given state of the walk, the currently visiting vertex, along with its adjacency list is visible to us.

In this paper, we propose, explain and compare two MCMC based algorithms for solving the sampling problem that we define in the previous paragraph. The first among these two is vertex-MCMC which we discuss in Section 4.3, and the second among these two is triple-MCMC, which we discuss in Section 4.4. In the following we will discuss a direct sampling approach first to prove that for a restricted graph direct sampling is not feasible.

4.2 Direct Sampling

A direct sampling method for sampling a triple from Π first constructs the probability mass function (f) over the sample space (if not given) using the weight function, and from that it constructs the cumulative mass function (say, F) of f. Then it uses the inverse-transform method to sample an object from the sample space. More formally, if the sampled object is x, then $x = F^{-1}(U)$ where $U \sim Uni(0, 1)$. For computer implementation, we can simply store the function F in a vector of size $|\Pi|$ considering an arbitrary (but constant) ordering of triples, and then choose an index from the vector uniformly using binary search, and return the triple corresponding to that index. For a restricted network, construction of F is impossible, so direct sampling method is not applicable for such a network.

Authors of [16] and [17] use a slightly modified version of direct sampling for sampling triples. Theirs' is a two-step sampling process. The first step samples a vertex v from a multinomial distribution, ζ, which is constructed by summing $f(\cdot)$ of each of the triples at the vertex v. Mathematically, $\zeta(v) = \sum_{t \in \Pi_v} f(t)$. It is easy to see that $\sum_{v \in V} P_\zeta(v) = 1$. The second step samples a triple from the set of triples at vertex v (Π_v) using another multinomial distribution, τ_v. If $t \in \Pi_v$, then $P_{\tau_v}(t) = \frac{f(t)}{\zeta(v)}$. Thus, the probability of sampling a triple, $P(t) = P(t|v) \cdot P(v) = P_{\tau_v}(t) \cdot P_\zeta(v) = \frac{f(t)}{\zeta(v)} \cdot \zeta(v) = f(t)$, as desired. As there are $O(n^2)$ triples in a graph, the cost of construction of *cmf*s of ζ and τ_v's is $O(n^2)$, and the cost of sampling by inverse-transform is logarithm of the sample space (cost of binary search). Overall complexity of sampling k triples is $O\left(n^2 + k(\lg n + \lg d_{max})\right)$, where n is the number of vertices, and d_{max} is the largest degree value for a vertex in the graph. Clearly, such a method is very inefficient.

However, note that the authors of [16] and [17] considered uniform distribution only. For this, $P_\zeta(v) = \frac{|\Pi_v|}{|\Pi|}$. Also, each of the τ_v's is trivially a uniform distribution. So, Z (*cmf* of ζ) can be computed in $O(n)$ time using equation 2 considering that the degree of a vertex is available in $O(1)$ time; we simply need to add the terms $\binom{d(v)}{2}$ in the above equation cumulatively for each of the vertices. Overall complexity of sampling k triples is then $O(n + k \lg n)$. Thus, the direct sampling of triples is efficient for uniform sampling, but not for arbitrary sampling.

Example: For uniform triple sampling of the graph presented in Figure 1, we choose a vertex with the distribution ζ. Under this, the vertex 3 is selected with probability 3/8, as there are three triples for which vertex 3 is the center. If vertex 3 is selected, we randomly choose two vertices from the adjacency list of 3 $(2, 4, 5)$ and construct one of the three possible triples and return. Consequently, the probability of triple $(3, 4, 5)$ being selected is $(3/8 \times 1/3 = 1/8)$, which is equal to $1/|\Pi|$, as desired. □

4.3 Vertex-MCMC for Triple sampling

We have seen in previous section that sampling a triple from an arbitrary distribution requires the construction of the *cmf* of ζ. For a restricted graph this is an infeasible task due to the lack of availability of the required information. Besides, the construction of the *cmf* of ζ takes $O(n^3)$ time, which is a fixed cost that is required to be paid up before the sampling process starts.

Our first indirect method to address the above limitations is to use an MCMC sampling method that does not construct ζ explicitly. We call it vertex-MCMC; the justification of this name will be clear in short time. Vertex-MCMC sampling uses a similar approach as the two-step direct sampling, but unlike the latter, it replaces the first-step (sampling a vertex from ζ) with an indirect sampling via MCMC. The second step of vertex-MCMC sampling remains unchanged from the two-step direct sampling method. More details of vertex-MCMC is given below.

For any MCMC algorithm, we need to define the states, the state transition process, the transition probability matrix, and the desired probability distribution. For vertex-MCMC, the set of states are the vertex-set V and the transition over the states happens along the edges (E). So the MCMC process is simply a random walk on the graph G. However, we want the stationary distribution of this walk to be identical to the desired distribution, which is ζ—identical to the desired distribution of vertices for a two-step sampling. To achieve the desired sampling distribution we will use Metropolis-Hastings (MH) algorithm.

Algorithm 1 triple sampling vertex-MCMC

```
1: procedure TRIPLESAMPLING2(G(V,E), k, {w(i)}_{i∈Π})
                          ▷ Graph G is given as vectors of
   adjacency vector, k is number of triples to be sampled,
   w(·) is user-defined weights of the triples.
2:     S ← φ
3:     u = An arbitrary starting vertex from V.
4:     while |S| ≠ s do
5:         v ← SelectNodeMCMC(u, {w(i)}_{i∈Π})      ▷ see
   Algorithm 2
6:         Select a triple t ∈ Π_v using t ∼ τ_v
7:         S.add(t)
8:     end while
9:     return S                    ▷ Return a set of s triples.
10: end procedure
```

Assume that MCMC random walk of a vertex-MCMC based triple sampler is visiting a vertex v. As was discussed in Section 3.3, MH algorithm uses a proposal distribution (q) to make a trial move; vertex-MCMC chooses q to be uniform over the neighborhood of v, in other word, it chooses one of the vertices (say, u) from the adjacency list of v uniformly. Therefore, the proposal distribution $q(v, u) = 1/d(v)$; here $d(v)$ is the number of nodes adjacent to node v. $q(v, u)$ represents the probability of an adjacency node u to be selected

from current node v. Similarly, $q(u,v) = 1/d(u)$. Now, using Equation 5, the acceptance probability of the proposal move is as shown in Equation 6.

$$\alpha(v,u) = \min\left\{1, \frac{P_\zeta(u)\cdot\frac{1}{d(u}}{P_\zeta(v)\cdot\frac{1}{d(v)}}\right\}$$
$$= \min\left\{1, \frac{\sum\limits_{t\in\Pi_u} f(t)\cdot d(v)}{\sum\limits_{t\in\Pi_v} f(t)\cdot d(u)}\right\} = \min\left\{1, \frac{\sum\limits_{t\in\Pi_u} w(t)\cdot d(v)}{\sum\limits_{t\in\Pi_v} w(t)\cdot d(u)}\right\} \quad (6)$$

Algorithm 2 MCMC node sampling

```
1: procedure SelectNodeMCMC(current, {w(i)}_{i∈Π})
   ▷ current is the currently visiting node, w(·) is user-defined weights of the triples.
2:     W_current = Σ_{x∈Π_current} w(x),        ▷ compute if not available from earlier iterations
3:     next = Uniform from adj(current)  ▷ Proposal step
4:     W_next = Σ_{x∈Π_next} w(x)  ▷ compute if not available from earlier iterations
5:     acceptance ← (W_next * d(current)) / (W_current * d(next))   ▷ See Equation 6
6:     if uniform(0,1) ≤ acceptance then
7:         return next
8:     end if
9:     return current
10: end procedure
```

Algorithm 1 illustrates the vertex-MCMC algorithm. The sampling process starts from an arbitrary seed vertex (Line 3). To sample the next vertex from ζ, the method calls the subroutine shown in Algorithm 2, which is simply an implementation of MH algorithm, where the sample space is the vertex set, and the target distribution is ζ. Once a vertex is selected on Line 5, vertex-MCMC computes the *cmf* of τ_v, and uses the direct sampling method to sample a triple using τ_v (Line 6). The process continues until the desired number of triples are obtained. The complexity of vertex-MCMC for sampling k triples is $O(kd_{max})$, as weight computations, and neighbor selection in Algorithm 2 can be performed in $O(d_{max})$ time.

Why vertex-MCMC method works for a restricted graph? The answer to this question is that the transition decisions of the random walk of vertex-MCMC are made using information that is locally available. More precisely, the transition decision of vertex-MCMC's random walk is made in Line 6 of Algorithm 2 using information computed in Line 2 and Line 4; these lines compute the sum of weights associated to the triples of *current* and *next* nodes, this computation can be accomplished within the scope of a restricted graph. Besides, vertex-MCMC method avoids the construction of the *cmf* of ζ vector, which makes it computationally more efficient than a direct sampling based method, the latter has a quadratic complexity with respect to the number of vertices in the network. Vertex-MCMC is also suitable for the case of a dynamic network.

LEMMA 2. *The random walk over the graph $G(V,E)$ in Algorithm 1 converges to a stationary distribution which is equal to ζ.*

PROOF: *To achieve a unique stationary distribution, a random walk needs to be ergodic which can be proved by showing that the walk is finite, irreducible and aperiodic [13]. The state space of the random walk in Algorithm 1 is finite with size $|V|$. We also assume that the input graph G is connected, so in this random walk any state u is reachable from any state v with a positive probability and vice versa, so the random walk is irreducible. Finally the walk can be guaranteed to be aperiodic by allocating a self-loop probability at every node* [1]. *This prove that the random walk achieves a unique stationary distribution. Now, consider two adjacent vertices $u, v \in V$ in graph G. Using Equation 6, the transition probability from v to u, $P_{vu} = 1/d(v)\cdot \min\left\{1, \frac{P_\zeta(u)\cdot d(v)}{P_\zeta(v)\cdot d(u)}\right\}$ which is equal to $\min\left\{\frac{1}{d(v)}, \frac{P_\zeta(u)}{d(u)\cdot P_\zeta(v)}\right\}$. Assuming that the stationary distribution probability of the node v is $\pi(v) = P_\zeta(v)$, then $\pi(v)\cdot P_{vu} = \min\left\{\frac{P_\zeta(v)}{d(v)}, \frac{P_\zeta(u)}{d(u)}\right\}$. Similarity, the transition probability from u to v is, $P_{uv} = \min\left\{\frac{1}{d(u)}, \frac{P_\zeta(v)}{d(v)\cdot P_\zeta(u)}\right\}$. Using $\pi(u) = P_\zeta(u)$ yields $\pi(u)\cdot P_{uv} = \min\left\{\frac{P_\zeta(u)}{d(u)}, \frac{P_\zeta(v)}{d(v)}\right\}$. Clearly we have, $\pi(v)\cdot P_{vu} = \pi(u)\cdot P_{uv}$. Thus the detailed balanced condition holds using ζ as the stationary distribution. Since, detailed balanced condition is a sufficient condition for an ergodic Markov chain to achieve the given stationary distribution, the random walk in Algorithm 1 achieves the stationary distribution ζ over the vertices of G.* □

We proved that the MCMC sampling of Algorithm 1 chooses a vertex v from the ζ distribution on convergence. Then the Algorithm samples a triple t centered at v using τ_v distribution (On Line 6). Thus using the correctness of the two-step direct sampling method, the sampled triple t is obtained from the desired distribution f (which is proportional to the weight $w(t)$). We will discuss the convergence of random walk in more details in Section 4.5.

Uniform sampling using vertex-MCMC:, $P_\zeta(v) = \binom{d(v)}{2} = \frac{d(v)(d(v)-1)}{2}$, The Equation 6 then changes as follows:

$$\alpha(v,u) = \min\left\{\frac{d(u)(d(u)-1)/2\cdot 1/d(u)}{d(v)(d(v)-1)/2\cdot 1/d(v)}\right\} = \min\left\{\frac{d(u)-1}{d(v)-1}\right\} \quad (7)$$

We do not show the pseudo-code of uniform triple sampling using vertex-MCMC. But, it is easy to obtain by making minor changes in Algorithm 1 and Algorithm 2. In Line 2 and Line 4 of Algorithm 2, we can compute the vertex weights in $O(1)$ time using the degree value of the corresponding vertex; the acceptance probability (Line 5) changes as shown in Equation 7. Finally, the Line 6 in Algorithm 1 requires to sample a triple from a uniform distribution instead of τ_v, which also takes $O(1)$ time. Due to the above changes, vertex-MCMC is faster in uniform sampling setting than weighted sampling setting. However, the theoretical complexity of the uniform triple sampling is still $O(kd_{max})$; although the weight computation cost is constant, we still need to find a neighbor of the currently visiting vertex, which in the worst case can take $O(d_{max})$ time.

[1]This is required only from a theoretical standpoint; in our experiment we do not allocate any self-loop probability explicitly.

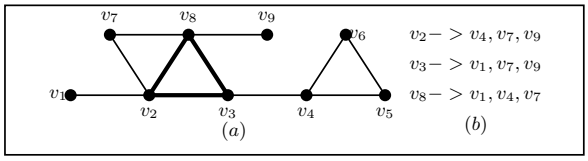

Figure 2: Induced triple and neighbors

4.4 MCMC walk over Triples

Our second indirect sampling method is named triple-MCMC, which performs MCMC walk over the triples. Triple-MCMC avoids computing *cmf* for both the distributions (ζ and $\{\tau_v\}_{v \in V}$). In fact, triple-MCMC is completely oblivious about the total number of triples in the graph. The set of states for this sampling algorithm is the set of all the triples, Π. Thus the random walk proceeds over the set of triples along a neighborhood graph which is defined below.

The neighbor of a triple is another triple with two common vertices. Thus, the triple-MCMC sampling obtains a sequence of dependent samples, where a sampled triple shares two vertices with the previous sampled triple. To compute the neighbor-set of a triple t, we need to find the other triples that can be obtained by replacing exactly one of the vertices of t.

Example: Suppose we are performing an MCMC walk on the graph shown in Figure 2 (a). Let $\langle v_2, v_3, v_8 \rangle$ be the currently visiting triple (triangle that is shown in bold line). In Figure 2 (b) we show the information of all its neighbors. The list labeled by v_2 contains the vertices that can be used to replace vertex v_2 to get a valid neighboring triple and similarly for the list labeled by v_3 and v_8. If the MCMC random walk chooses to go to the neighboring triple by replacing the vertex v_8 with v_1, the next sampled triple becomes a path $\langle v_1, v_2, v_3 \rangle$. On the other hand, if the vertex v_3 is replaced by v_7, we get the closed triple $\langle v_2, v_7, v_8 \rangle$ where the center of the triple is taken as v_7, which is the lastly added vertex of the triple. The transition between triples happens only between the neighboring triples. For example, the transition probability between $\langle v_2, v_3, v_8 \rangle$, and $\langle v_4, v_5, v_6 \rangle$ is zero, as they are not neighbors of each other according to our neighborhood definition. □

Let's assume that the random walk of triple-MCMC is visiting a triple t. For proposal distribution (say q), we choose one of the triples from t's neighborhood (say, s) uniformly. So, $q(t, s) = \frac{1}{|\mathcal{N}(t)|}$. Here, $\mathcal{N}(t)$ is the set of neighbors of triple t. Using Equation 5, the acceptance probability of the proposal move is obtained as below:

$$\alpha(t, s) = \min\left\{1, \frac{f(s) \cdot \frac{1}{|\mathcal{N}|(s)|}}{f(t) \cdot \frac{1}{|\mathcal{N}(t)|}}\right\} = \min\left\{1, \frac{w(s) \cdot |\mathcal{N}(t)|}{w(t) \cdot |\mathcal{N}(s)|}\right\} \tag{8}$$

We show a pseudo-code in Algorithm 3. Since, the random walk is performed over the triple space, we initialize the walk with an arbitrary triple t_p, any path of length 2 suffices (Line 3). Now, for the currently visiting triple, t_p, we want to find the next triple using MH algorithm. For this, we first find all the neighboring triples of t_p (Line 5). Reader may review the Figure 2 to refresh the notion of the neighboring triples. This computation requires finding unions or intersections of the adjacency lists of the current triple's vertices, and its complexity is $O(d_{max})$. Then a neighboring triple (say, t_q) is selected uniformly from all the neighbors, and accepted with the probability computed on Line 8. If the move is rejected, the currently sampled triple is sampled again. The process continues until k samples are obtained. The overall cost of obtaining k samples is $O(kd_{max})$.

LEMMA 3. *The random walk over the triples of $G(\Pi)$ in Algorithm 3 converges to a stationary distribution with is proportional to $w(t)$*

PROOF: *The proof is almost identical to the proof of Lemma 2. We first show that the walk is ergodic. The state space Π is finite, because the number of triples is finite. We also assume that the input graph G is connected, so in this random walk any triple y is reachable from another triple x with a positive probability and vice versa, so the random walk is irreducible. Finally the walk can be made aperiodic by allocating a self-loop probability at every node. Thus the random walk reaches a stationary distribution. Similar to the proof of Lemma 2, we can show that the random walk satisfies the detailed balanced condition for the stationary distribution f (proportional to w). Since the detailed balance condition is sufficient for an ergodic random walk to reach the given stationary distribution, the lemma is proved.* □

Algorithm 3 triple sampling triple-MCMC

```
1: procedure TRIPLESAMPLING3(G, k, {w(i)}_{i∈Π})
                          ▷ Graph G is given as vectors of
   adjacency vector, k is number of triples to be sampled,
   w(·) is user-defined weights of the triples.
2:     S ← φ
3:     t_p ← a random triple ▷ A length 2 path is sufficient
4:     while |S| ≠ s do
5:         t_q ← RandomNeighborTriple(t_p)
6:         n_p ← NeighborCount(t_p)
7:         n_q ← NeighborCount(t_q)
8:         acceptance ← (w(t_q)*n_p) / (w(t_p)*n_q)
9:         if uniform(0,1) ≤ acceptance then
10:            S.add(t_q)
11:            t_p ← t_q
12:        else
13:            S.add(t_p)
14:        end if
15:    end while
16:    return S                  ▷ Return a set of s triples.
17: end procedure
```

4.4.1 Uniform triple sampling

For uniform sampling, the weight of an open triple is 1 and the weight of a closed triple is 3 (a triangle represents three triples), i.e., for uniform sampling in Equation 8, we set $w(s)$ and $w(t)$ to be 1 or 3 depending on whether the corresponding triple is open or close. MH algorithm guarantees that the Algorithm 3 using the above acceptance probability yields a uniform triple sampler.

4.5 Convergence Analysis

Convergence analysis is important for any MCMC sampling based method because through such analysis we can estimate the mixing time (number of walks to converge to the stationary distribution overcoming the influence of the starting state) of a Markov chain. Mixing time depends on (i) the neighborhood structure of the space on which the

Name	Nodes	Edges	Triple Count ($\|\Pi\|$)
ca-Hepth	8,638	24,806	297,397
ca-Grqc	4,158	13,422	227,919
ca-Cond	21,363	91,286	1,959,920

Table 1: Small real-life Networks used in sampling quality experiments.

walk is performed, and (ii) the desired target distribution. A method to measure the convergence rate is to find the *spectral gap* of the transition probability matrix P. P has n real eigenvalues $1 = \lambda_0 > \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{n-1} \geq -1$. Then, the *spectral gap* is defined as $\lambda = 1 - \max\{\lambda_1, |\lambda_{n-1}|\}$. Since the absolute values of all the eigenvalues are less than one (based on Perron-Frobenius theorem) with the largest eigenvalue λ_0 be exactly one, the spectral gap is always between 0 and 1. The higher the spectral gap, the faster the convergence [4]. For triple sampling in a restricted graph, the entire transition matrix P is not available, so it is infeasible to measure the spectral gap.

However, if the objective of MCMC sampling is to measure a metric over the sampling population, some experimental methods are available to study the convergence of the chain solely based on the convergence of the metric value. One such method is called *Geweke diagnostics* [7]. Both vertex-MCMC, and triple-MCMC sampling can be used to approximate $\gamma(G)$, the transitivity of the graph G, so this metric can be analyzed for convergence study using Geweke diagnostics. It works as follows: we consider X to be a single sequence of triple samples. Also let, $X_i = 1$ if ith sample of the sequence is a triangle and $X_i = 0$ otherwise. Then the expected value of the random variables in the sequence X gives as unbiased estimate of transitivity, i.e., $E[X] = \gamma(G)$.

Geweke considers two subsequences of samples, X_a form the beginning part of X (typically first 10%) and X_b from the last part of X (typically last 50%). From these two subsequence he computes z-statistic: $z = \frac{E[X_a]-E[X_b]}{\sqrt{Var(X_a)+Var(X_b)}}$. X_a ans X_b goes further apart as the number of samples is increased. Consequently, the correlation between the subsequences decreases. After convergence, there is no correlation between X_a and X_b, and z becomes normally distributed with mean 0 ($E[X_a] = E[X_b] = \gamma(G)$, so, $E[X_a] - E[X_b] = 0$) and variance 1. The number of iterations that it takes for the z-score to fall between [-1, 1] is considered the mixing time. However, one should run the experiment for at least a few distinct walks, and declare convergence when z-scores from all the walks fall within the $[-1, 1]$ range. In Section 5.4 we will show that only a few hundred walks are sufficient to achieve convergence even on a graph having more than a million of vertices.

5. EXPERIMENTS AND RESULTS

In our experiments we first show the performance of uniform triple sampling. Then we show the performance of triple sampling from a nonuniform distribution. Finally, we demonstrate that, uniform sampling of triples can be applied for approximating triangle count, which provides an application driven method for measuring sampling effectiveness.

5.1 Datasets

All the graphs [2] listed in Table 1 and 2, are undirected, unweighted, simple and connected. We pre-process them to ensure these properties. The specification of the graphs (vertex count and edge count) may not match with the source, as in source, for some networks an undirected edge is represented by two directed edges in opposite directions; in our representation, for such edges we discard one edge of the edge-pairs. Additionally, we ensure that the graph is connected as MCMC algorithms perform a random walk over the graph. However, for all the graphs that we use, the largest connected component of a graph retains more than 90% of the edges.

[2]obtained from `http://snap.stanford.edu,http://socialnetworks.mpi-sws.org/,http://www.cise.ufl.edu/research/sparse`

Name	Nodes	Edges	Triangle Count
AS-Skitter	1,694,616	11,094,209	28,769,842
flickr	1,624,992	15,476,835	548,646,525
livejournal	5,189,809	48,688,097	310,784,143
orkut	3,072,441	117,185,083	627,584,181
Soc-LiveJournal	4,843,953	42,845,684	285,688,896
Wikipedia 2005/11	1,596,970	18,539,720	44,667,088
Wikipedia 2006/9	2,935,762	35,046,792	84,018,181
Wikipedia 2006/11	3,099,074	37,042,065	88,823,813
Wikipedia 2007/2	3,512,462	42,374,383	102,434,914

Table 2: Large real-life Networks used in approximate triangle count experiments.

5.2 Uniform sampling performance

Our first experiment compares the performance of uniform triple sampling of vertex-MCMC with that of the direct sampling method discussed in Section 4.2 (We skip triple-MCMC for this experiment as It can be easily demonstrated that for uniform sampling, vertex-MCMC is more efficient than triple-MCMC). For this comparison we simulate both the methods on an input graph by sampling triples (with replacement); then we study the statistics of *sample count*, a random number representing the frequency at which a triple is sampled. For each of input graphs, we run the sampler for $|\Pi| \times i$ iterations, where Π is the set of distinct triples in that graph. By definition, sample count follows a binomial distribution $\mathcal{B}(k, m, p)$, where $m = |\Pi|.i$ and $p = \frac{1}{|\Pi|}$. For this distribution, the median sample count will be identical to the mean, which is $m.p = |\Pi|.i.\frac{1}{|\Pi|} = i$ and the variance is $m * p(1-p) = \frac{i(|\Pi|-1)}{|\Pi|} \approx i$. Since m is large, this binomial distribution resembles a normal distribution. We run this experiments using i=50 on smaller graphs that are listed in Table 1. Performing this experiment on large graphs is infeasible, because for this experiment we need to store the visit count of all the triples in the memory, and the number of such triples is in the order of billions for large graphs.

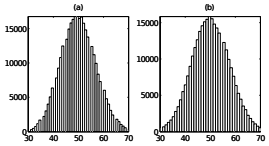

Figure 3: Frequency histogram of the visit counts on ca-Hepth network using (a) Direct triple sampling (b) vertex-MCMC triple sampling.

Graph	Direct	Vertex -MCMC
ca-Hepth	49.93	59.37
ca-Grqc	49.96	58.15
ca-Cond	50.09	52.05

Table 3: Comparison of variances among Direct sampling, and Vertex-MCMC sampling (Median is 50 for all the cases).

In Figure 3, we show the frequency histogram of sample counts for the ca-Hepth network. In this plot, x-axis shows different sample count values, and y-axis represents the number of distinct triples that achieves that value for its sample count. The shape of the histogram is a perfect normal graph, which is expected from an ideal iid distribution. Besides, the median value for both the cases is also 50, as expected (note that i=50). The histograms of other networks are almost identical, hence are not shown. We report the variance of sample counts in Table 3 for both the sampling algorithms on all 3 small networks. For an ideal case, the variance value is around 50 (which is equal to i). As we can see in this table, for Ca-Hepth network, using vertex-MCMC uniform sampler, the variance is 59.37, but the same is 49.93 for a direct triple sampler. For all the networks, the direct method's performance is almost identical to an ideal case; for vertex-MCMC method, variance of sample count is slightly bigger, yet very close to its true value. Note that, the higher value of variance for vertex-MCMC can be attributed to the nature of indirect sampling where a sample is constrained to be within the neighbourhood of the previous sample.

5.3 Non-uniform sampling performance

In this experiment we verify the quality of sampling when the triples to be sampled follow a non-uniform distribution. We compare Direct sampling with both vertex-MCMC and triple-MCMC algorithms. In this experiment we use the dataset listed in Table 1 for the reason discussed in Section 5.2. Here, our objective is to sample triple t in proportion to $w(t)$. For this experiment, we consider $w(t) = |\mathcal{N}(t)|$, i.e., a triple is sampled with the probability proportional to the size of its neighbourhood. One motivation of choosing such a sampling distribution can be to sample triples from a community or a dense region of a graph; in such a neighbourhood, a triple will be surrounded by many triples, so $|\mathcal{N}(t)|$ will be high for a triple t in a dense neighbourhood. For the above choice of target distribution, the acceptance probability of vertex-MCMC is, $\alpha(v,u)$ is $\min\left\{1, \frac{\sum_{t\in\Pi_u}\mathcal{N}(t)\cdot d(v)}{\sum_{t\in\Pi_v}\mathcal{N}(t)\cdot d(u)}\right\}$ and the same for the triple-MCMC, $\alpha(t,s)$ is 1.

For a network with $|\Pi|$ triples, the desired distribution over Π can be expressed as a vector f of size $|\Pi|$, here, $f(t) = \frac{w(t)}{W}$. For each of the graphs, we run each sampler for $|\Pi|\cdot i$ times (we choose $i = 10$ for our experiment). The distribution f can be approximated by the sample frequency of each of the triples. Therefore, $\widehat{f}(t) = \frac{count(t)}{|\Pi|\cdot i}$; here, $count(t)$ is the number of times the triple t was sampled and $\widehat{f}$ is the approximation of f that is obtained by the sampling algorithm. The performance of a sampling method can be measured by the correlation between f and $\widehat{f}$. Table 4 shows that all the three methods achieves excellent value for the correlation (more than 0.85). Interestingly, direct method sometimes perform worse than the other methods, our investigation shows that this is because of the precision issue of the floating-point while handling very small probabilities. More precisely, Ca-Hepth network has 227,919 triples, and the cumulative mass probabilities (which sums to 1) is stored in a vector of that size; in this vector the difference between successive cells are sometimes as small as 10^{-8}, and to perform well a uniform random number generator's precision has to be good for that many decimal points, which apparently is not true for existing random number generators. In all our experiments we use Boost random number generator library that has much better performance that those available in standard C++ library. In Figure 4, we compare f vs $\widehat{f}$ distributions for all the three methods using scatter plots for one of the graphs (ca-Grqc). The superiority of triple-MCMC over other sampling methods is easily visible in this figure.

Graph	Direct	MCMC	
		vertex	triple
ca-Hepth	0.86	0.86	0.87
ca-Grqc	0.87	0.87	0.92
ca-Cond	0.93	0.93	0.94

Table 4: Correlation between target distribution and achieved distribution by different sampling algorithms.

Graph	Direct		Vertex-MCMC		Triple-MCMC	
	Time(s)		Time(s)		Time(s)	
	/1k	/10k	/1k	/10k	/1k	/10k
ca-Hepth	5.07	27.46	3.87	27.66	0.03	0.33
ca-Grqc	10	78.83	8.83	80.51	0.05	0.47
ca-Cond	145.5	1225.05	82.65	1075.79	0.08	0.81

Table 5: Execution times of the algorithms for sampling 1k triples and 10k triples.

Table 5 shows the execution time for sampling 1k and 10k triples from the networks using different sampling algorithms. As the table shows, Triple-MCMC is much better than Vertex-MCMC and the direct method. For example, Triple-MCMC takes only 0.08 second to sample 1k triples from ca-Cond network, whereas Direct method and Vertex-MCMC takes 145.5 and 82.65 seconds respectively. This is because, triple-MCMC does not need to compute the *cmf* ζ and τ_v explicitly. Computing ζ is a fixed cost for the direct sampling method. Vertex-MCMC distribute this fixed cost over the iterations because it computes the ζ of each vertex only on demand. On the other hand, triple-MCMC computes $w(t)$ of a specific triple only on demand. If we take more samples, the difference between the direct sampling and vertex-MCMC slowly diminishes, as with many iterations, both the methods can amortize the fixed cost over those iterations. Here, it should be noted that, *cmf* τ_v is not explicitly stored in memory. Storing τ_v will require memory in the order of $O(|\Pi|)$, which is same as enumerating the whole set of triples. And if enumeration is possible, then we do not need to sample triples in the first place.

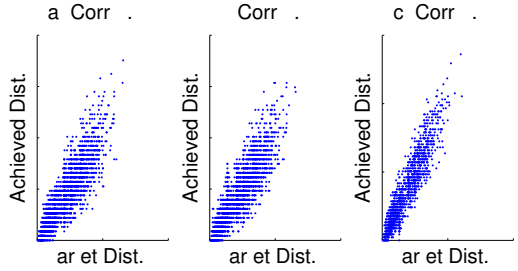

Figure 4: Target distribution vs achieved distribution plot for ca-Grqc network (a) Direct triple sampling (Corr. 0.87) (b) Vertex-MCMC triple sampling (Corr. 0.87) (c) triple-MCMC triple sampling (Corr. 0.92).

Graph	mixing time	
	vertex-MCMC	triple-MCMC
AS-Skitter	496	3001
flickr	114	230
livejournal	186	311
orkut	149	2276
Soc-LiveJournal	<100	327
Wikipedia 2005/11	834	3628
Wikipedia 2006/9	773	3202
Wikipedia 2006/11	1233	3270
Wikipedia 2007/2	543	3827

Table 6: Mixing time to achieve convergence according to Geweke diagnostics

5.4 Convergence experiments

In this experiment, we use Geweke diagnostics for studying the mixing time of vertex-MCMC and triple-MCMC methods while sampling from a uniform distribution. This experiment is not shown for user-defined distribution, because mixing time is different for different choices of distribution.

For this experiment, we use ten distinct walk sequences for each dataset. For each sequence, we compute z-score starting from 100'th walk and declare convergence when all ten z-scores fall in $[-1, 1]$ range. In Table 6 (column 2 and 3) we report the mixing time that we compute from Geweke diagnostics. For all the graphs that we use, the mixing time is only a few hundreds for vertex-MCMC and a few thousands for triple-MCMC, which is very good considering the size of these networks. The triple-MCMC takes more time to mix as the neighborhood graph on which this method performs the random walk is much larger than the same for the vertex-MCMC's walk.

5.5 Approximate triangle counting

In this experiment, we compare the performance of vertex-MCMC algorithm with direct triple sampling [16, 17], and one of the sampling method by Buriol et al. [5] [3]. Note that, recent works [17] have shown that the direct sampling method is the best among the existing methods for approximate triangle counting; so we do not include other triangle counting methods such as DOULION [19] in this experiment. However, we do include Buriol et al.'s method in this comparison, because it is also a triple sampling method like vertex-MCMC. We exclude triple-MCMC in this comparison as when performing uniform sampling its execution time is not competitive with these methods (although it is the best choice for weighted sampling). The task assigned to each sampling method is to approximate the triangle count by sampling triples from uniform distribution. Here, we use the sampled set of triples to approximate triangle count using the idea that was explained in Section 3.1. For our experiment, we use 9 large real-life networks; name and statistics of these networks are shown in Table 2.

The performance of approximate triangle counting is measured by two metrics: execution time and accuracy. Typically, the method that wins in accuracy loses in running time. So, to make the comparison easy, we compare the accuracy (plotted on y axis) of different methods against different running time (plotted on x-axis). However, do note that for a given value for time, the number of triples that the methods sample differ. We show the results in Figure 5(a-c). We fitted the data with Bezier curve to show the trend of the algorithms. All the accuracy and execution times are average value that are computed from 10 runs of the algorithms. We can see that the direct sampling method performs the best, even for a small number of samples, and its performance improvement remains almost flat as the sample count increases; on the other hand, vertex-MCMC improves sharply as the number of samples increases, and for some graphs its performance even surpasses the performance of direct sampling. So, vertex-MCMC is particularly suitable for large graphs, where a sampling method can afford to take many sample, and yet can be competitive with an exact algorithm. For instance, in wiki20060925 graph, both direct sampling and vertex-MCMC obtain 90% counting accuracy for a 6 seconds execution time, but the exact method that uses an efficient edge iterator algorithm takes 67 seconds to execute. The charts in this figure also confirm that Buriol et el.'s method is not competitive with either of these methods.

The accuracy of a direct sampling-based method is better than that of a vertex-MCMC based method. This is because the latter performs indirect sampling in which a pair of consecutive samples are dependent. So, it's result has high variances and it requires more samples in order to ensure uniformity of sampled set of triples. However, MCMC based methods can work perfectly on restricted or dynamic networks, whereas direct sampling based methods are not applicable to those.

5.6 Comparison with different graph sampling technique

In this section, we compare our triple sampling method with various network sampling methods that are proposed in Leskovec et al. [10]. These methods create a small network from a large network through node (or edge) sampling with an objective that the sampled network would preserve various properties of the large network. For the purpose of this experiment, we select the transitivity ($\gamma(G)$) property, i.e., we investigate whether the exact transitivity value computed from the sampled network using the above methods is a good approximation of the actual transitivity value of the original network. For node based sampling, we sample a graph by sampling 5% of the nodes and then take the graph induced by the sampled node set. Similarly for edge based sampling, we sample 5% of the existing edges. The detailed description of all the network sampling methods can be found in [10].

The result is shown in Table 7 using average error metric, which is obtained by averaging over 10 executions. As we can see, none of the network sampling methods (9 methods, from RN to FF) is able to preserve the transitivity metric of the large network for an acceptable accuracy. The best among those is the RN (random node sampling) with a 6.8% error in the flicker dataset. On the other hand, vertex-MCMC achieves much better accuracy by sampling only $5\% * |V|$ number of triples from each of the graphs (see the last column in Table 7).

6. CONCLUSION

In this work, we propose two MCMC based algorithms for sampling triples form a large network. We show experimental results that demonstrate that both the algorithms

[3]this method is actually proposed for arbitrary edge stream setting, but for fair comparison we implement it as a non-stream in memory method.

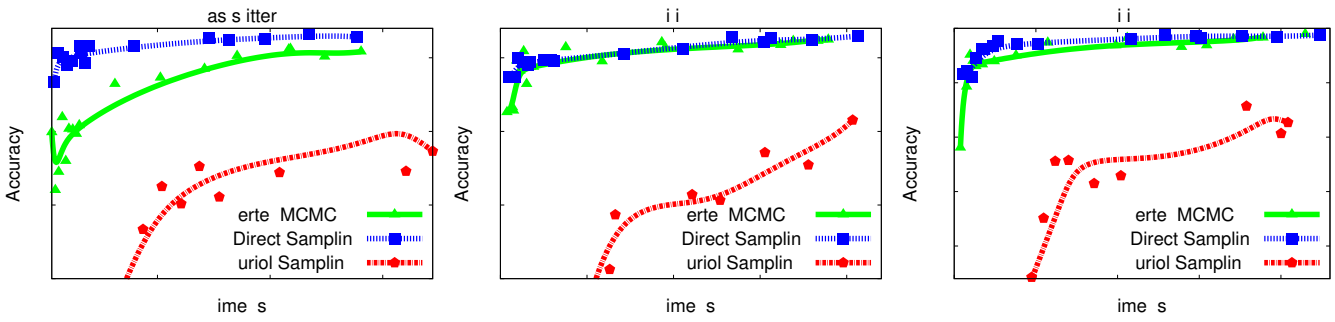

Figure 5: Comparison (of running time and approximation accuracy of transitivity or triangle counting) among the sampling algorithms (a) as-skitter (b) wiki20060925 (c) wiki20061104. Exact triangle counting times are 3.8s, 66.6s and 72.57s respectively. Results on other graphs are similar, hence not shown.

Graph $G(V,E)$	5% of		Avg Error (%)									
	V	E	RN	RDN	RE	RNE	HYB	RNN	RW	RJ	FF	Vertex-MCMC
as-skitter	84731	554710	33	848	94	96	97	93	77	75	267	12
flicker	81250	773842	6.8	59	94	98	99	92	93	97	27	1.28
orkut	153622	5859254	64	43	98	96	97	97	69	96	59	4.49
Soc-LiveJournal	242198	2142284	15	100	95	96	97	93	73	91	60	12.6
wikipedia 2006/11	154954	1852103	63	517	94	96	97	87	47	91	168	5.26
wikipedia 2007/2	175623	2118719	51	513	95	96	97	88	58	91	145	6.68

Table 7: Average error of various algorithms proposed by Leskovec et al. 5% of nodes ($0.05 * |V|$) are sampled for node based algorithms and 5% of edges ($0.05 * |E|$) are sampled for edge based algorithms. To be fair to leskovec et al, Number of triples sampled for Vertex-MCMC is also equals $5\% * |V|$. Each result is average over 10 executions.

achieve excellent performance while sampling triples from a large network using a given distribution. Direct sampling method's performance is almost identical to an ideal sampler, but it is costly, specifically while sampling from a weighted distribution. On the other hand, the MCMC sampling methods that we propose is faster as it does not compute the *cmf* of the desired distribution. More importantly, MCMC sampling methods can sample triples from networks that are restricted or dynamic, for which direct sampling methods fail.

7. REFERENCES

[1] M. Al Hasan and M. J. Zaki. Output space sampling for graph patterns. *Proc. of VLDB Endowment*, 2(1):730–741, 2009.

[2] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. of SODA*, pages 623–632, 2002.

[3] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proc. of 4th ACM SIGKDD*, pages 6–24.

[4] S. Boyd, P. Diaconis, and X. Lin. Fastest mixing markov chain on a graph. *SIAM Review*, 46(4):667–689, 2004.

[5] L. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler. Counting triangles in data streams. In *Proc. of PODS*, pages 253–262, 2006.

[6] J.-P. Eckmann and E. Moses. Curvature of co-links uncovers hidden thematic layers in the world wide web. *PNAS*, 99(9):5825–5829, April 2002.

[7] J. Geweke. Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments. In *Bayesian Statistics*, pages 169–193, 1992.

[8] M. Gjoka, M. Kurant, C. Butts, and A. Markopoulou. Walking in Facebook: A Case Study of Unbiased Sampling of OSNs. In *INFOCOM, Proc. of IEEE*, pages 1–9, 2010.

[9] M. Kolountzakis, G. Miller, R. Peng, and C. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Math.*, 8(1-2):161–185, 2012.

[10] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *Proc. of the 12th ACM SIGKDD*, pages 631–636, 2006.

[11] A. S. Maiya and T. Y. Berger-Wolf. Sampling community structure. In *Proc. of WWW*, pages 701–710, 2010.

[12] M. McPherson, L. Smith-Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27(1):415–444, 2001.

[13] R. Motwani and P. Raghavan. *Randomize Algorithms*. Cambridge University Press, 1995.

[14] M. E. J. Newman, D. J. Watts, and S. H. Strogatz. Random graph models of social networks. *PNAS*, 99(Suppl 1):2566–2572, 2002.

[15] M. Rahman, M. Bhuiyan, M. Rahman, and M. Hasan. Guise: a uniform sampler for constructing frequency histogram of graphlets. *Knowledge and Info. Systems*, pages 1–26, 2013.

[16] T. Schank and D. Wagner. Approximating clustering-coefficient and transitivity. *J. of Graph Algorithms and Applications*, 9(2):265–275, 2005.

[17] C. Seshadhri, A. Pinar, and T. G. Kolda. Triadic measures on graphs: The power of wedge sampling. In *Proc. of Siam Data Mining*, 2013.

[18] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proc. WWW*, pages 607–614, 2011.

[19] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos. Doulion: Counting triangles in massive graphs with a coin. In *Proc. of KDD*, 2009.

[20] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.