2021 IEEE 37th International Conference on Data Engineering (ICDE) | 978-1-7281-9184-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICDE51399.2021.00083

A Bootstrapping Approach to Optimize Random Walk Based Statistical Estimation over Graphs

Pei Yi*, Hong Xie*, Yongkun Li†, John C.S. Lui‡
*College of Computer Science, Chongqing University
†School of Computer Science & Technology, University of Science & Technology of China
‡Department of Computer Science and Engineering, The Chinese University of Hong Kong
*{201914131045,xiehong2018}@cqu.edu.cn, †ykli@ustc.edu.cn, ‡cslui@cse.cuhk.edu.hk

***Abstract*—Graphs are commonly used in various applications such as online social networks (OSNs), E-commerce systems and social recommender systems. Random walk sampling is often used to conduct statistical estimation over such graphs. This paper develops an algorithmic framework to reduce the mean square error of such statistical estimation. Our algorithmic framework is inspired by that the mean square error can be decomposed into a sum of the bias and variance of the estimator. More specifically, we apply the bootstrapping technique to design a bias reduction algorithm. A new feature of this bias reduction algorithm is that it allows the variance to increase whenever the bias can be further reduced. The increased variance may lead to a large mean square error of the estimator. We use multiple parallel random walks to reduce this variance such that it can be reduced to arbitrarily small by deploying a sufficient number of random walks. Our algorithmic framework enables one to attain different trade-offs between the sample complexity (i.e., number of parallel random walks) and the mean square error of the statistical estimation. Also, the proposed bias reduction algorithm is generic and can be applied to optimize a large class of random walk sampling algorithms. To demonstrate the versatility of the framework, we apply it to optimize the Metropolis random walk and simple random walk sampling. Extensive experiments confirm the effectiveness and efficiency of our proposed algorithmic framework.**

***Index Terms*—Random walk; Bootstrapping; Graph; Statistical estimation;**

I. Introduction

Statistical estimation over graphs is a fundamental task in graph analytic problems [1]–[8]. A number of statistical estimation problems over graphs have been studied ranging from estimating simple statistics such as node degree distribution, vertex label distribution, size estimation [3], [4], [9]–[11], etc., to sophisticated statistics such as classification, ranking and regression [2], [12]–[14]. It is not an easy task to conduct statistical estimation over graphs. First, graphs in applications are usually large in scale. For example, the number of monthly active users of Facebook has reached over two billion [15]. Second, the whole graph is usually not accessible to third-party agents. In many OSNs, only APIs are available for third-party agents to access the graph data. Random walk sampling is a mainstream method to address this challenge [9], [15]–[19]. We use the following simplified example to illustrate random walk based statistical estimation over graphs.

Example 1. *A company needs to make decisions regarding whether to do advertisements over a social network. The social network is characterized by a graph $\mathcal{G} \triangleq (\mathcal{V}, \mathcal{E}, x)$, where $\mathcal{V}$ denotes the user set, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the edge set and the function x prescribes an attribute $x(v) \in \{1, \dots, 10\}$ for vertex $v \in \mathcal{V}$. The attribute $x(v)$ quantifies the degree of proneness of user v on advertisements over social networks. The company wants to know the mean α and standard deviation σ of the degree of proneness over the whole user population:*

$$\alpha = \sum_{v \in \mathcal{V}} \frac{x(v)}{|\mathcal{V}|}, \qquad \sigma = \sqrt{\sum_{v \in \mathcal{V}} \frac{(x(v) - \alpha)^2}{|\mathcal{V}|}}.$$

Suppose we use the Metropolis random walk sampling algorithm [9] to get samples from the graph $\mathcal{G}$ (details in Sec. IV). Suppose we get $L \in \mathbb{N}_+$ samples $U_1, \dots, U_L$, where $U_i \in \mathcal{V}$. Then one can estimate the mean and standard deviation as:

$$\widehat{\alpha} = \sum_{i=1}^{L} \frac{x(U_i)}{L}, \qquad \widehat{\sigma} = \sqrt{\sum_{i=1}^{L} \frac{(x(U_i) - \widehat{\alpha})^2}{L}}.$$

Example 1 illustrates one typical characteristic of random walk based statistical estimation over graphs, i.e., only a finite number (usually small number) of samples are generated to conduct the estimation [8], [20]. One central problem is how to improve the estimation accuracy under this finite sample setting. A number of random walk algorithms were proposed to solve this problem [8]–[10], [15], [21]. In other words, these works improve estimation accuracy via getting "better" samples. This paper aims to improve estimation accuracy from an orthogonal perspective, i.e., we apply bootstrapping techniques to exploit the property of the statistic in estimation to improve estimation accuracy.

Our framework is inspired by bootstrapping techniques and recent graph analytic systems that enable one to run millions of random walks in parallel on consumer-level personal computers [22]–[26]. The accuracy of the estimators in Example 1 can be characterized by bias and variance. To illustrate, consider the estimator $\widehat{\sigma}$. The mean square error of $\widehat{\sigma}$ is denoted by $\texttt{MSE}(\widehat{\sigma})$:

$$\texttt{MSE}(\widehat{\sigma}) \triangleq \mathbb{E}[(\widehat{\sigma} - \sigma)^2] = \texttt{Var}[\widehat{\sigma}] + (\texttt{Bias}(\widehat{\sigma}))^2,$$

where $\texttt{Var}[\widehat{\sigma}]$ and $\texttt{Bias}(\widehat{\sigma})$ are defined as:

$$\texttt{Var}[\widehat{\sigma}] \triangleq \mathbb{E}[(\widehat{\sigma} - \texttt{Mean}(\widehat{\sigma}))^2], \ \texttt{Bias}(\widehat{\sigma}) \triangleq \texttt{Mean}(\widehat{\sigma}) - \sigma,$$

with $\texttt{Mean}(\widehat{\sigma}) \triangleq \mathbb{E}[\widehat{\sigma}]$. This implies that one can reduce the estimation error by reducing the variance or bias. Note that these observations hold for many statistics beyond the mean and standard deviation and hence we consider general statistics. We apply the bootstrapping technique to reduce the bias. Unlike most previous bootstrapping techniques that reduce the bias under the constraint of not changing the variance or only allowing it to increase slightly [27], we allow the variance to increase whenever the bias can be further reduced. Then we use multiple parallel random walks to reduce the variance. One may argue that this rises an issue of scalability. Fortunately, it can be addressed by recent graph analytic systems [22]–[26], which enable running millions of random walks in parallel on consumer-level personal computers. In fact, the variance can be reduced by averaging, i.e.,

$$\texttt{Var}\left[\widehat{\sigma}_{\texttt{mr}}\right] = \frac{\texttt{Var}[\widehat{\sigma}]}{n},$$

where $\widehat{\sigma}_{\texttt{mr}} = (\widehat{\sigma}^{(1)} + \ldots + \widehat{\sigma}^{(n)})/n$ denotes the average of n estimations of σ from n parallel random walks with the same initial point. Note that the bias is unchanged, i.e., $\texttt{Bias}(\widehat{\sigma}_{\texttt{mr}}) = \texttt{Bias}(\widehat{\sigma})$. Namely, one can reduce the variance to arbitrarily small by deploying a sufficient number of random walks. We develop an algorithmic framework to reduce both bias and variance of the estimator. Our contributions are:

- We develop an algorithmic framework to reduce the mean square error of random walk based statistical estimation over graphs. Our algorithmic framework provides a novel combination of *random walk sampling* and *bootstrapping technique*, and it enables one to attain different trade-offs between sample complexity (i.e., number of parallel random walks) and mean square error of the statistical estimation.
- We apply the bootstrapping technique to design a bias reduction algorithm. A new feature of our bias reduction algorithm is that it allows the variance to increase whenever the bias can be further reduced. The increased variance may lead to large mean square error of the estimator. To overcome this problem, we use multiple parallel random walks to reduce this variance, and show that it can be reduced to arbitrarily small by deploying a sufficient number of random walks. Our bias reduction algorithm is generic and can be applied to a large class of random walk sampling algorithms and statistical estimation problems.
- To demonstrate the versatility of our framework, we apply it to optimize the Metropolis random walk sampling and simple random walk sampling. Experiment results on four public datasets show that our algorithmic framework can reduce the bias of both random walks without bias reduction (or with a classical bias reduction method) by as high as around 80% (or 60%). We also achieve similar reduction on mean square error using only 1000 parallel random walks and a larger number of random walks can lead to a larger reduction.

This paper is organized as follows. Section II presents the graph model and the problem formulation. Section III presents a general algorithmic framework, which uses a random walk sampling oracle and a bias reduction oracle to optimize the random walk based statistical estimation over graphs. Section IV present algorithms to implement the random walk sampling oracle. Section V presents an algorithm to implement the bias reduction oracle via bootstrapping. Section VI and VII presents the experimental evaluation of our methods on four real-world datasets in terms of bias and mean square error respectively. Section VIII presents the related work and Section IX concludes.

II. Model & Problem Formulation

We first present the graph model and statistical estimation model. Then we present the problem formulation.

A. The Graph Model

We consider an undirected graph with a finite set

$$\mathcal{V} \triangleq \{1, \ldots, V\}$$

of vertices, where $V \in \mathbb{N}_+$. Each vertex v can be a user in an OSN, or an item in a social recommender system, etc. Let $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denote the edge set. As the graph $\mathcal{G}$ is undirected, then $(u,v) \in \mathcal{E}$ implies that $(v,u) \in \mathcal{E}$. For example, an edge $(u,v) \in \mathcal{E}$ can indicate the friendship between u and v in a social network. We focus on the case that the graph $\mathcal{G}$ is connected. Let $d(v) \in \mathbb{N}_+$ denote the degree of vertex v, formally

$$d(v) \triangleq |\{u|(v,u) \in \mathcal{E}\}|.$$

Let $\mathcal{N}(v) \subseteq \mathcal{V}$ denote the neighbor set of vertex v, formally

$$\mathcal{N}(v) \triangleq \{u|(v,u) \in \mathcal{E}\}.$$

One can observe that $d(v) = |\mathcal{N}(v)|$.

We consider a real value attribute. In particular, each vertex v is associated with a value $x(v) \in \mathcal{X}$ indicating the attribute, where $\mathcal{X} \in \mathbb{R}$ denotes the value set. For example, the value $x(v)$ can denote the gender of vertex v, then $\mathcal{X} = \{-1(\text{male}), 1(\text{female})\}$. The value $x(v)$ can also denote the degree of vertex v, then $\mathcal{X} = \{1, \ldots, d_{\max}\}$, where $d_{\max} = \max_{v \in \mathcal{V}} d(v)$. Let μ denote a distribution over the attribute value set $\mathcal{X}$, which summarizes the collective attribute value over the whole vertex set. Formally, we define μ as:

$$\mu(y) = \sum_{v \in \mathcal{V}} \frac{\{x(v)=y\}}{|\mathcal{V}|}, \forall y \in \mathcal{X}.$$

Namely, $\mu(y)$ is the fraction of vertices with value $y \in \mathcal{X}$. To simplify notations, we denote the undirected graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, x)$.

B. The Estimator Model

We consider a class of statistics $\theta \in \mathbb{R}$ over the graph $\mathcal{G}$ such that it can be expressed as a function of the distribution μ, i.e., $\theta = T(\mu)$, where T denotes a mapping function:

$$T : \mu \mapsto \mathbb{R}.$$

For example, the mean of value can be modeled as

$$T(\mu) = \sum_{y \in \mathcal{X}} \mu(y) y. \quad (1)$$

The standard deviation of value can be modeled by T as

$$T(\mu) = \sqrt{\sum_{y \in \mathcal{X}} \mu(y)(y - \bar{y})^2}, \quad (2)$$

where $\bar{y} = \sum_{y \in \mathcal{X}} \mu(y) y$. The standard deviation can be generalized to

$$T(\mu) = \left(\sum_{y \in \mathcal{X}} \mu(y) |y - \bar{y}|^c \right)^{1/c},$$

where $c \in \mathbb{R}_+$. We like to remark that many statistical estimation problems over the graph $\mathcal{G}$ produce an estimator being a function of the distribution μ, e.g., maximum likelihood estimation, regression, etc. Namely, the statistic $\theta = T(\mu)$ can model a large class of statistical estimation problems over the graph.

C. Problem Formulation

We consider a large scale graph $\mathcal{G}$ and one has to use random walk sampling to estimate the statistic θ. This setting is adopted in many previous works [8], [15], [28]. We aim to design an algorithm denoted by $\mathbb{A}$ to estimate the statistic θ via samples generated by a random walk sampling algorithm running on the graph $\mathcal{G}$. We consider two metrics in the design of $\mathbb{A}$. The first one is the sample complexity defined as:

$$\texttt{SC}(\mathbb{A}) \triangleq \text{\# of samples required by the algorithm}$$

The second one is the estimation error. We consider the mean square error defined as $\texttt{MSE}(\widehat{\Theta}) \triangleq \mathbb{E}[(\widehat{\Theta} - \theta)^2]$, where $\widehat{\Theta}$ denotes an estimator of θ produced by the algorithm $\mathbb{A}$. The objective is to design an algorithm $\mathbb{A}$ to estimate the statistic θ attaining different trade-offs between the above two metrics.

III. Algorithmic Framework

We present a general algorithmic framework to estimate the statistic θ based on a sampling oracle and a bootstrapping oracle for bias estimation. We also derive the analytical expression for the estimation error of our algorithmic framework as well as its sample complexity. These analytical expressions enable one to trade sample complexity for reducing the estimation error. The detail of the sampling oracle and bootstrapping oracle is deferred to Section IV and V respectively.

A. Design of the Algorithmic Framework

We first define a random walk sampling oracle, which supports *sampling query* and *estimating query*.

Definition 1. *A random walk oracle denoted by* `RWOracle` *is defined as a function such that:*

- *for each query with initial point $U_0 \in \mathcal{V}$ and sample length $L \in \mathbb{N}_+$, it returns L samples denoted by $\boldsymbol{U} \triangleq (U_1, \ldots, U_L)$:*

$$\boldsymbol{U} = \texttt{RWOracle.Sampling}(U_0, L);$$

- *for each query with a sequence of L samples $\boldsymbol{U}$, it returns an estimation of the statistic θ:*

$$\widehat{\Theta}_{\texttt{rw}}(\boldsymbol{U}) = \texttt{RWOracle.Statistics}(\boldsymbol{U}).$$

We defer details of the random walk oracle `RWOracle` to Section IV. Here we focus on applying it to design our algorithmic framework. Note that for the linear statistic, we usually have $\widehat{\Theta}_{\texttt{rw}}(\boldsymbol{U})$ being asymptotically unbiased, i.e., the bias $(\mathbb{E}[\widehat{\Theta}_{\texttt{rw}}(\boldsymbol{U})] - \theta)$ goes to zero when L goes to infinity (more details in Section IV). Unfortunately, in practice, we can only collect a finite number of samples, under which the bias is not negligible. To reduce the bias, let us define an oracle to bootstrap the bias of the estimator $\widehat{\Theta}_{\texttt{rw}}(\boldsymbol{U})$ first.

Definition 2. *The bootstrapping oracle denoted by* `BootBiasOracle` *is defined as an algorithm such that for each query with a sample sequence $\boldsymbol{U}$ and the corresponding random walk oracle* `RWOracle`*, it returns an estimation for the bias of $\widehat{\Theta}_{\texttt{rw}}(\boldsymbol{U})$:*

$$\Delta(\widehat{\Theta}_{\texttt{rw}}(\boldsymbol{U})) = \texttt{BootBiasOracle}(\boldsymbol{U}, \texttt{RWOracle}),$$

where $\Delta(\widehat{\Theta}_{\texttt{rw}}(\boldsymbol{U}))$ denotes the estimated bias of $\widehat{\Theta}_{\texttt{rw}}(\boldsymbol{U})$.

We defer details of `BootBiasOracle` to Sec. V. Here, let us focus on applying it to design our algorithmic framework.

To present our algorithmic framework, we need the following notations. Denote $M \in \mathbb{N}_+$ initial points as:

$$\boldsymbol{U}_0 \triangleq (U_{0,1}, \ldots, U_{0,M}),$$

where $U_{0,M} \in \mathcal{V}$. Denote $N_m \in \mathbb{N}_+$ as the number of parallel random walks associated with initial point $U_{0,m}$. For simplicity, denote

$$\boldsymbol{N} \triangleq (N_1, \ldots, N_M).$$

Denote $L_m \in \mathbb{N}_+$ as the length of each parallel random walk associated with initial point $U_{0,m}$. For simplicity, denote

$$\boldsymbol{L} \triangleq (L_1, \ldots, L_M).$$

Algorithm 1 outlines a parallel algorithmic framework to estimate the statistic θ over the graph $\mathcal{G}$. In step 2 of Algorithm 1, we run $\sum_{m=1}^{M} N_m$ random walks in parallel via the random walk oracle `RWOracle.Sampling`. These random walks are organized into M groups. Group m has N_m parallel random walks and each random walk within this group has the same initial point $U_{0,m}$. After we obtain samples from these $\sum_{m=1}^{M} N_m$ random walks, in step 3 we apply the bootstrapping oracle `BootBiasOracle` to estimate the bias of each random walk sequence. We then use the estimated bias to debias the random walk estimator by deducting it from the estimator. Finally, return the average of debiased estimators as an estimator of the statistic θ.

B. Analysis of the Algorithmic Framework

To analyze the performance of Algorithm 1, we first decompose the mean square error into follows:

$$\texttt{MSE}(\widehat{\Theta}) = \texttt{Var}[\widehat{\Theta}] + [\texttt{Bias}(\widehat{\Theta})]^2,$$

Algorithm 1 Algorithmic Framework

```
1: Input: $\boldsymbol{U}_0, \boldsymbol{N}, \boldsymbol{L}$, Random walk oracle RWOracle, Bootstrapping oracle BootBiasOracle
2: Parallel random walk sampling: run $\sum_{m=1}^{M} N_m$ random walks in parallel to get samples:
       $\boldsymbol{U}_{m,n} \leftarrow$RWOracle.Sampling$(U_{0,m}, L_m)$,
           $\forall m = 1, \dots, M, n = 1, \dots, N_m$
3: Estimate the bias via bootstrapping:
       $\Delta_{m,n} \leftarrow$ BootBiasOracle$(\boldsymbol{U}_{m,n},$ RWOracle$)$,
           $\forall m = 1, \dots, M, n = 1, \dots, N_m$
4: Compute the debiased estimators:
       $\widehat{\Theta}_{m,n} \leftarrow$ RWOracle.Statistics$(\boldsymbol{U}_{m,n}) - \Delta_{m,n}$,
           $\forall m = 1, \dots, M, n = 1, \dots, N_m$
5: Return: $\widehat{\Theta} \leftarrow \frac{1}{M}\sum_{m=1}^{M}\sum_{n=1}^{N_m}\frac{\widehat{\Theta}_{m,n}}{N_m}$
```

where $\texttt{Var}[\widehat{\Theta}]$ and $\texttt{Bias}(\widehat{\Theta})$ are defined as the variance and bias of the estimator $\widehat{\Theta}$:

$$\texttt{Var}[\widehat{\Theta}] \triangleq \mathbb{E}[(\widehat{\Theta} - \texttt{Mean}(\widehat{\Theta}))^2], \quad \texttt{Bias}(\widehat{\Theta}) \triangleq \texttt{Mean}(\widehat{\Theta}) - \theta,$$

with $\texttt{Mean}(\widehat{\Theta}) \triangleq \mathbb{E}[\widehat{\Theta}]$ denoting the mean of the estimator $\widehat{\Theta}$.

Note that for a given group of random walks with the same initial point $U_{0,m}$, the $\widehat{\Theta}_{m,n}$ across $n = 1, \dots, N_m$ are independent and identically distributed. Thus, we denote the variance and bias of $\widehat{\Theta}_{m,n}$ as

$$\sigma_m^2 \triangleq \texttt{Var}[\widehat{\Theta}_{m,n}], \quad \delta_m \triangleq \texttt{Bias}(\widehat{\Theta}_{m,n}), \quad \forall n = 1, \dots, N_m.$$

Lemma 1. *The variance and bias of $\widehat{\Theta}$ produced by Algorithm 1 can be derived as*

$$\textit{Var}[\widehat{\Theta}] = \frac{1}{M^2}\sum_{m=1}^{M}\frac{\sigma_m^2}{N_m}, \qquad \textit{Bias}(\widehat{\Theta}) = \sum_{m=1}^{M}\frac{\delta_m}{M}.$$

Furthermore, $\lim_{\forall m, N_m \to \infty} \textit{MSE}(\widehat{\Theta}) = [\textit{Bias}(\widehat{\Theta})]^2$.

Due to page limit, all proofs are in our technical report [29]. Lemma 1 states closed-form expressions for the variance and bias of the estimator $\widehat{\Theta}$ outputted by Algorithm 1. One can reduce the variance of $\widehat{\Theta}$ to zero by increasing the number of parallel random walks N_m at each initial point m to infinity. The bias of $\widehat{\Theta}$ is the average of the bias of multiple random walks associated with M initial points. The following lemma derives analytical expression for the sample complexity of Algorithm 1.

Lemma 2. *The sample complexity of Algo. 1 is*

$$\textit{SC}(\mathbb{A}_{\texttt{Algo1}}) = \sum_{m=1}^{M} N_m L_m,$$

where $\mathbb{A}_{\texttt{Algo1}}$ denotes Algo. 1.

Lemma 2 states the sample complexity for Algorithm 1. The sample complexity increases linearly with the sample length L_m associated with each initial point $U_{0,m}$, and also increases linearly with the number of parallel random walks in each initial point. In real-world applications, one usually does not want to have a long sample sequence L_m as it is time-consuming. However, one may want to use a larger number of parallel random walks to decrease the estimation error. Hence, the analytical expressions derived in Lemma 1 and 2 enable one to select the appropriate sample complexity to trade it for improving estimation accuracy.

IV. The Sampling Oracle

Due to page limit, we only demonstrate how to apply the Metropolis random walk [30] to implement a random walk sampling oracle. Note that one can easily extend this random walk sampling oracle to other random walk algorithms, such as simple random walk [31], or its sophisticated variants [15].

The Metropolis random walk is an application of the Metropolis-Hastings algorithm [30] to graph sampling. Under the Metropolis random walk algorithm, in each step, the walker moves to a neighbor of the current vertex with certain probability derived as follows:

$$\mathbb{P}[U_{\ell+1}|U_\ell] = \begin{cases} \frac{1}{d(U_\ell)}\min\left(1, \frac{d(U_\ell)}{d(U_{\ell+1})}\right), & \text{if } U_{\ell+1} \in \mathcal{N}(U_\ell), \\ 1 - \sum_{v \in \mathcal{N}(U_\ell)} \frac{1}{d(U_\ell)}\min\left(1, \frac{d(U_\ell)}{d(v)}\right), & \text{if } U_{\ell+1} = U_\ell, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Note that the graph $\mathcal{G}$ is connected. Thus, the Metropolis random walk has a stationary distribution. Let π denote the stationary distribution of the Metropolis random walk. It can be derived as $\pi(v) = 1/|\mathcal{V}|, \forall v \in \mathcal{V}$. Furthermore, for any function $f : \mathcal{V} \to \mathbb{R}$, it holds

$$\lim_{L \to \infty} \sum_{\ell=0}^{L} \frac{f(U_\ell)}{L} = \sum_{v \in \mathcal{V}} \frac{f(v)}{|\mathcal{V}|}.$$

Namely, one can use the samples from the Metropolis random walk to construct asymptotic unbiased estimators (when the sample length goes to infinity). It is important to remember that when the sample length is finite, the bias is non-zero.

Based on the Metropolis random walk, Algorithm 2 outlines a random walk oracle. Consider the $\texttt{RWOracle.Sampling}(U_0, L)$ function, the algorithm first simulates the Metropolis random walk by $\tilde{L} \in \mathbb{N}_+$ steps, for the purpose of making the random mix. This period is also called the burn-in period. Then the algorithm simulates another L steps and returns them as the samples. The function $\texttt{RWOracle.Estimate}(\boldsymbol{U})$ first estimates the distribution μ using the samples $\boldsymbol{U}$. Then, it uses the estimated distribution to produce an estimator of the statistic θ.

Algorithm 2 `RWOracle` based on Metropolis random walk

```
1: function RWORACLE.SAMPLING($U_0, L$)
2:     for $\ell = 1, \dots, \tilde{L}$ do                ▷ Burn-in period
3:         $U_\ell \leftarrow v$ with prob. $\mathbb{P}[v|U_{\ell-1}]$ derived in Eq. (3)
4:     for $\ell = \tilde{L}+1, \dots, \tilde{L}+L$ do    ▷ sampling period
5:         $U_\ell \leftarrow v$ with prob. $\mathbb{P}[v|U_{\ell-1}]$ derived in Eq. (3)
6:     return $(U_{\tilde{L}+1}, \dots, U_{\tilde{L}+L})$
7: function RWORACLE.ESTIMATE($\boldsymbol{U}$)
8:     Estimate the probability mass:
           $\widehat{\mu}(y) \leftarrow \frac{\sum_{i=1}^{\text{length}(\boldsymbol{U})} \mathbb{1}_{\{x(U_i)=y\}}}{\text{length}(\boldsymbol{U})}, \forall y \in \mathcal{X}$
9:     return $T(\widehat{\mu})$
```

V. Bootstrapping Bias

We first design an oracle to bootstrap the bias via Jackknife. More importantly, we derive sufficient conditions, under which this oracle can reduce the bias of estimator. These conditions reveal a restriction that this oracle may perform poorly when the graph is incomplete. Then we design another oracle to relieve this restriction, and theoretically show that it works even over incomplete graphs. This new oracle is achieved at the cost of increasing the variance, which can be reduced by increasing the number of parallel random walks as shown in Section III.

A. Bootstrapping Bias via Jackknife

We apply the Jackknife [32] to estimate the bias of Algorithm 1, i.e.,

$$\texttt{Bias}(\widehat{\Theta}) = \texttt{Mean}(\widehat{\Theta}) - \theta.$$

Recall the closed-form expression for estimator $\widehat{\Theta}$ in line 5 of Algo. 1, i.e.,

$$\widehat{\Theta} = \frac{1}{M} \sum_{m=1}^{M} \sum_{n=1}^{N_m} \frac{\widehat{\Theta}_{m,n}}{N_m}.$$

By the linearity of expectation, it boils down to estimate the bias for $\widehat{\Theta}_{m,n}$. The $\widehat{\Theta}_{m,n}$ is evaluated from the samples denoted by

$$\boldsymbol{U}_{m,n} \triangleq \left\{ U_{m,n}^{(1)}, \dots, U_{m,n}^{(L_m)} \right\}.$$

For simplicity of notation, let $\boldsymbol{U}_{m,n}^{(-i)}$ denote a vector of samples excluding the i-th sample $U_{m,n}^{(i)}$:

$$\boldsymbol{U}_{m,n}^{(-i)} \triangleq \left[U_{m,n}^{(1)}, \dots, U_{m,n}^{(i-1)}, U_{m,n}^{(i+1)}, \dots, U_{m,n}^{(L_m)} \right].$$

Namely, $\boldsymbol{U}_{m,n}^{(-i)}$ is a $(L_m - 1)$-sized sub-sample from $\boldsymbol{U}_{m,n}$. In total, we have L_m such sub-samples: $\boldsymbol{U}_{m,n}^{(-1)}, \dots, \boldsymbol{U}_{m,n}^{(-L_m)}$.

From the $(L_m - 1)$-sized sub-sample $\boldsymbol{U}_{m,n}^{(-i)}$, we apply the random walk oracle to generate one estimator as:

$$\widehat{\Theta}_{m,n}^{(-i)} = \texttt{RWOracle.Estimate}\left(\boldsymbol{U}_{m,n}^{(-i)}\right).$$

Applying the Jackknife, the estimate of the bias of $\widehat{\Theta}_{m,n}$ is

$$\widehat{\texttt{Bias}}(\widehat{\Theta}_{m,n}) = (L_m - 1) \left(\frac{\sum_{i=1}^{L_m} \widehat{\Theta}_{m,n}^{(-i)}}{L_m} - \widehat{\Theta}_{m,n} \right). \quad (4)$$

Note that $\widehat{\texttt{Bias}}(\widehat{\Theta}_{m,n})$ estimates the bias fully relying on the samples $\boldsymbol{U}_{m,n}$ itself. Based on $\widehat{\texttt{Bias}}(\widehat{\Theta}_{m,n})$, we outline a bootstrapping oracle to estimate the bias in Algorithm 3.

Algorithm 3 `BootBiasOracle`($\boldsymbol{U}_{m,n}$, `RWOracle`) via Jackknife

```
1: for $i = 1, \dots, L$ do
2:     $\boldsymbol{U}_{m,n}^{(-i)} \leftarrow [U_{m,n}^{(1)}, \dots, U_{m,n}^{(i-1)}, U_{m,n}^{(i+1)}, \dots, U_{m,n}^{(L_m)}]$
3:     $\widehat{\Theta}_{m,n}^{(-i)} \leftarrow$ RWOracle.Estimate$\left(\boldsymbol{U}_{m,n}^{(-i)}\right)$
4: $\widehat{\Theta}_{m,n} \leftarrow$ RWOracle.Estimate$(\boldsymbol{U}_{m,n})$
5: Estimate the bias
       $\widehat{\texttt{Bias}}(\widehat{\Theta}_{m,n}) \leftarrow (L_m - 1)\left(\frac{\sum_{i=1}^{L_m} \widehat{\Theta}_{m,n}^{(-i)}}{L_m} - \widehat{\Theta}_{m,n}\right)$
6: return $\widehat{\texttt{Bias}}(\widehat{\Theta}_{m,n})$
```

To analyze the theoretical guarantee of Algorithm 3, we next define a class of expandable T.

Definition 3. *Let $X_1, \dots, X_n$ denote n independent and identically distributed samples from the distribution μ. Let $T(\mu_n)$ denote an estimator of the statistic θ, where $\mu_n(y) = \sum_{i=1}^{n} \mathbb{1}_{\{X_i=y\}}/n$. The T is expandable if it satisfies*

$$\mathbb{E}[T(\mu_n)] = \theta + \sum_{j=1}^{\infty} \frac{a_j(\mu)}{n^j},$$

where $a_j(\mu) \in \mathbb{R}, \forall j = 1, \dots, \infty$ is independent of n.

Most statistics are expandable [32], i.e., mean, variance, most maximum likelihood estimators, etc. For example, when the T is the mean, i.e., derived in Equation 1, we have $\mathbb{E}[T(\mu_n)] = \theta$. When T is the variance, i.e., derived in Equation (2), we have

$$\mathbb{E}[T(\mu_n)] = \theta - \frac{\theta}{n}.$$

The following theorem states theoretical guarantee for Algorithm 3 under expandable statistic.

Theorem 1. *Suppose $\mathcal{G}$ is complete and the T is expandable with $a_1(\mu) > 0$. Algo. 3 corrects the bias from $\texttt{Bias}(\widehat{\Theta}_{m,n}) = O(1/L_m)$ to $\texttt{Bias}(\widehat{\Theta}_{m,n}^{JK}) = O(1/L_m^2)$, where*

$$\widehat{\Theta}_{m,n}^{JK} \triangleq \widehat{\Theta}_{m,n} - \widehat{\texttt{Bias}}(\widehat{\Theta}_{m,n})$$

denotes the corrected estimator.

Theorem 1 states sufficient conditions under which Algorithm 3 corrects the bias of estimator $\widehat{\Theta}_{m,n}$ from $O(1/L_m)$ to $O(1/L_m^2)$. One sufficient condition is that $\mathcal{G}$ has to be a complete graph. However, in real-world applications, graphs are usually incomplete. In this case, we may not have theoretical guarantee for Algorithm 3. The following section explores how to relieve this restriction.

B. Improve Accuracy via Sub-sample Selection

One restriction of Algorithm 3 is that some of the $(L_m - 1)$-sized sub-samples of $\boldsymbol{U}_{m,n}$ are not valid random walk sequences when the graph is incomplete. When the graph is incomplete, it may happen that $\boldsymbol{U}_{m,n}^{(-i)}$ is not a sample sequence generated by the random walk sampling algorithm. In particular, according to the transition probability in Equation (3), the walker can not move from vertex $U_{m,n}^{(i-1)}$ to $U_{m,n}^{(i+1)}$. Formally, we define valid and invalid sub-sample in the following definition.

Definition 4. *A sub-sample $\boldsymbol{U}_{m,n}^{(-i)}$ is valid under the Metropolis random walk if $\mathbb{P}[U_{m,n}^{(i+1)}|U_{m,n}^{(i-1)}] > 0$ for $i \geq 2$ and $U_{m,n}^{(i+1)} = U_{m,n}^{(i-1)}$ for $i = 1$, otherwise it is invalid.*

Invalid sub-samples may lead to the bias estimation (i.e., Equation (4)) being inaccurate. To relieve this problem, we use a greedy approach, i.e., we filter out all invalid sub-samples. Let $\mathcal{I}_{m,n}$ denote the index of all the valid sub-samples of $\boldsymbol{U}_{m,n}$. Then we estimate the bias as

$$\widehat{\mathtt{Bias}}_{\mathtt{VS}}(\widehat{\Theta}_{m,n})=(L_m - 1)\left(\frac{\sum_{i\in\mathcal{I}_{m,n}} \widehat{\Theta}_{m,n}^{(-i)}}{|\mathcal{I}_{m,n}|} - \widehat{\Theta}_{m,n}\right). \quad (5)$$

One can observe that we take the average over $|\mathcal{I}_{m,n}|$ samples in (5), i.e., $\sum_{i\in\mathcal{I}_{m,n}} \widehat{\Theta}_{m,n}^{(-i)}/|\mathcal{I}_{m,n}|$, while we take the average of L_m sub-samples in Equation (4), i.e., $\sum_{i=1}^{L_m} \widehat{\Theta}_{m,n}^{(-i)}/L_m$. Taking the average over fewer sub-samples leads to a larger variance of the bias estimation $\widehat{\mathtt{Bias}}_{\mathtt{VS}}(\widehat{\Theta}_{m,n})$ than that of $\widehat{\mathtt{Bias}}(\widehat{\Theta}_{m,n})$ with all the sub-samples. Namely, our approach of filtering out all invalid sub-samples is at the cost of increasing the variance. As we have shown in Section III, one can increase the number of parallel random walks to reduce the variance. Based on the new bias estimator $\widehat{\mathtt{Bias}}_{\mathtt{VS}}(\widehat{\Theta}_{m,n})$, we implement an oracle to bootstrap bias in Algorithm 4.

Algorithm 4 `BootBiasOracle`($\boldsymbol{U}_{m,n}$, `RWOracle`) with valid sub-sample selection

```
1: Compute the index set $\mathcal{I}_{m,n}$ for valid sub-samples of $\boldsymbol{U}_{m,n}$
2: for $i \in \mathcal{I}_{m,n}$ do
3:     $\boldsymbol{U}_{m,n}^{(-i)} \leftarrow [U_{m,n}^{(1)}, \dots, U_{m,n}^{(i-1)}, U_{m,n}^{(i+1)}, \dots, U_{m,n}^{(L_m)}]$
4:     $\widehat{\Theta}_{m,n}^{(-i)} \leftarrow \texttt{RWOracle.Estimate}\left(\boldsymbol{U}_{m,n}^{(-i)}\right)$
5: $\widehat{\Theta}_{m,n} \leftarrow \texttt{RWOracle.Estimate}\left(\boldsymbol{U}_{m,n}\right)$
6: Estimate the bias
       $\widehat{\mathtt{Bias}}_{\mathtt{VS}}(\widehat{\Theta}_{m,n}) \leftarrow (L_m - 1)\left(\frac{\sum_{i\in\mathcal{I}_{m,n}} \widehat{\Theta}_{m,n}^{(-i)}}{|\mathcal{I}_{m,n}|} - \widehat{\Theta}_{m,n}\right)$
7: return $\widehat{\mathtt{Bias}}_{\mathtt{VS}}(\widehat{\Theta}_{m,n})$
```

Due to page limit, we present theoretical guarantees for Algorithm 4 in our technical report [29].

VI. Experiment I: the Bias

We conduct experiments on four real-world datasets published on SNAP[1] to evaluate our algorithmic framework on bias reduction. Experiment results further confirm the superior performance of our algorithm in reducing the bias.

A. Experiment Setting

Dataset. Table I summarizes four datasets published on SNAP. We have two reasons in selecting them: (1) each node is associated with an attribute, i.e., community; (2) they are from four diverse applications. For each dataset, if it is a directed graph, we add reciprocal edges to make it an undirected one. In each dataset we consider two types of attributes: (1) the community associated with a node; (2) the degree of a node.

TABLE I
Overall statistics of four datasets

	# of nodes	# of edges	# of communities
com-Amazon	334,863	925,872	75,149
wiki-topcats	1,791,489	28,511,807	17,364
com-Orkut	3,072,441	117,185,083	6,288,363
com-LiveJournal	3,997,962	34,681,189	287,512

Statistical estimation model. Consider the degree as the attribute of nodes. We have $\mathcal{X} = \{1, \dots, d_{\max}\}$ and $x(v)$ denotes the degree of node v. Furthermore, $\mu(y)$ denotes the fraction of nodes with degree $y \in \mathcal{X}$. We aim to estimate the standard deviation of degree:

$$T(\mu) = \sqrt{\sum_{y\in\mathcal{X}} \mu(y)(y - \bar{y})^2}, \quad (6)$$

where $\bar{y} = \sum_{y\in\mathcal{X}} \mu(y)y$ denotes the average degree. Consider the community as the attribute of nodes. We first rank the community ID, then based on the ranked list we divide the community ID into $K \in \mathbb{N}_+$ groups such that each group contains the same number of community ID. A node has attribute $k = 1, \dots, K$, if it belongs to a community with ID in group k. Thus, we have $\mathcal{X} = \{1, \dots, K\}$. We further set $\mu(k)$ as the fraction of nodes with community ID belonging to group k. Note that $\sum_{k=1}^{K} \mu(k) \neq 1$, as a node may belong to multiple communities or a node may not belong to any communities. Note that our framework applies to the case that $\sum_{k=1}^{K} \mu(k) \neq 1$. We estimate the variation of $\mu(k)$ as:

$$T(\mu)=\left(\sum_{k=1}^{K} \frac{1}{K}\left|\mu(k) - \frac{\mu(1) + \ldots + \mu(K)}{K}\right|^c\right)^{1/c}, \quad (7)$$

where $c \in \mathbb{R}_+$. When $c > 2$, the above statistic corresponds to generalized standard deviation.

Baseline & Parameter setting. To demonstrate the versatility of our framework, we apply it to reduce the bias of Metropolis random walk and simple random walk. When Metropolis random walk [30] serves as the baseline, we compare: (1) **MR**, which is a variant of Algorithm 2 without bias reduction, i.e., it is the Metropolis random walk; (2) **JKM**, which is a variant

[1]http://snap.stanford.edu/data/index.html

of Algorithm 2 and it uses Jackknife, i.e., Algorithm 3, to reduce the bias of Metropolis random walk; (3) **VSM**, which uses our bootstrapping oracle with valid sub-sample selection, i.e., Algorithm 4, to reduce bias of Metropolis random walk. When simple random walk serves as the baseline, we compare: (1) **SRW**, which extends Algorithm 2 to simple random walk, i.e., it is the simple random walk algorithm; (2) **JKS**, which extends Algorithm 3 to simple random walk, i.e., it uses Jackknife to reduce the bias of simple random walk; (3) **VSS**, which extends Algorithm 4 to simple random walk, i.e., it uses our bootstrapping oracle with valid sub-sample selection to reduce the bias of simple random walk.

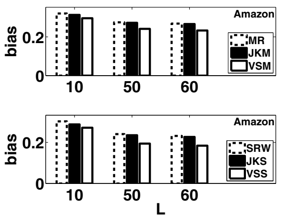
Note that we need to select M initial point for each of the above six algorithms. We rank nodes based on the ID and then select ranked $(|\mathcal{V}|\frac{1}{M})$-th, ..., $(|\mathcal{V}|\frac{M}{M})$-th nodes as M initial points. Random walks in each group have the same length, i.e., $L_1{=}\ldots{=}L_M{=}L$. We also run the same number of parallel random walks on each initial point, i.e., $N_1{=}\ldots = N_M = N$. We set the burn-in period to $\tilde{L}{=}0$. The reason is to give stronger confidence on the performance of our algorithm, because if our algorithm performs well when $\tilde{L}{=}0$, then it will perform better when $\tilde{L}{>}0$ as implied by our technical report [29]. Due to similar reasons, we select only one valid sub-sample, i.e., $\boldsymbol{U}_{m,n}^{(-L_m)}$. Unless we state explicitly, we consider the following default parameters, i.e., $M{=}10, L{=}50, N{=}10^5, c{=}2$ and $K{=}5$ to compute the bias. Note that running this large number of random walks on each initial point is to ensure an accurate estimation of the bias via the Monter Carlo method.

B. Impact of Sample Length

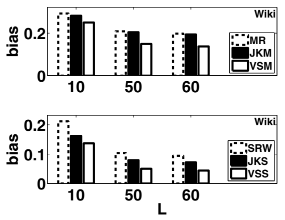
We vary the sample length L from 10 to 60, while set the other parameters as their default values. Consider the case that community serves as node attribute. Figure 1 shows the bias of six algorithms (i.e., **MR, JKM**, **VSM**, **SRW, JKS** and **VSS**) evaluated on four datasets in Table I, where the statistic under estimation is derived in Equation (7). One can observe that the bias of all these six algorithms decrease when the sample length L varies from 10 to 60. Namely, the bias decrease in sample length. Among **MR**, **JKM** and **VSM**, our **VSM** has the smallest bias followed by the **JKM**. Using a sample length of $L = 50$, our **VSM** reduces the bias of **MR** (or **JKM**) by as high as 40% (30%). Our bootstrapping method **VSS** reduces the bias of **SRW** (or **JKS**) by as high as 50% (30%).

Consider the case that node degree serves as node attribute. Figure 2 shows the bias of estimating the statistic stated in Equation (6). One can observe that among **MR**, **JKM** and **VSM**, our **VSM** has the smallest bias. Using a sample length of $L{=}50$, **VSM** reduces the bias of **MR** (or **JKM**) by as high as 80% (50%). Furthermore, our bootstrapping method **VSS** reduces the bias of **SRW** (or **JKS**) by as high as 70% (60%).

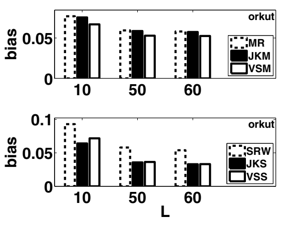
Lessons learned: Under different sample lengths, our bootstrapping method reduces the bias of both Metropolis random walk and simple random walk significantly and it can also reduce the bias of the Jackknife method.



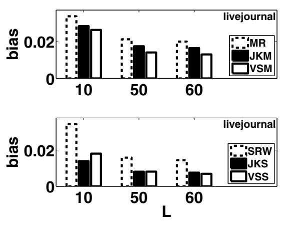
(a) Amazon



(b) Wiki

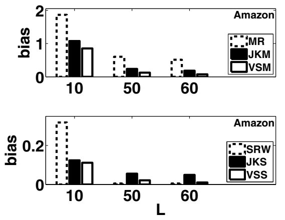


(c) orkut

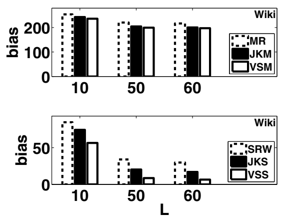


(d) livejournal

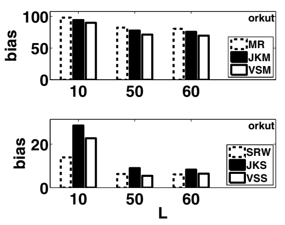
Fig. 1. Impact of sample length L on bias with Metropolis random walk (MR) and simple random walk (SRW) as baselines. [**community as attribute**]



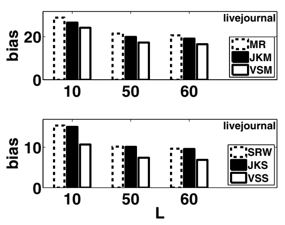
(a) Amazon



(b) Wiki



(c) orkut



(d) livejournal

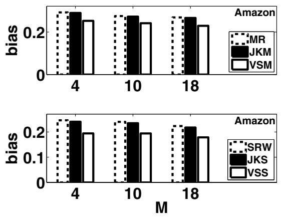
Fig. 2. Impact of sample length L on bias with Metropolis random walk (MR) and simple random walk (SRW) as baselines. [**degree as attribute**]

C. Impact of Initial Points

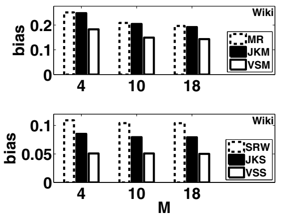
We vary the number of initial points M from 4 to 18. We fix the total number of random walks to be $MN = 10^6$. Consider the case that community servers as node attribute. Figure 3 shows the bias of estimating the statistic derived in Equation (7). One can observe that our bootstrapping algorithm **VSM** can reduce the bias of Metropolis random walk **MR** by as high as 40%. This reduction ratio varies slightly as the number of initial points varies from $M = 4$ to $M = 18$. Furthermore, our bootstrapping method **VSM** reduces the bias of Jackknife method **JKM** by as high as 30%. Furthermore, our bootstrapping method **VSS** can reduce the bias of simple random walk **SRW** by as high as 50%. This reduction ratio varies slightly when the number of initial points varies from $M = 4$ to $M = 18$. When simple random walk serves as the baseline, our bootstrapping method **VSS** reduces the bias of Jackknife method **JKS** by as high as 30%.

Consider the case that node degree serves as node attribute. Figure 4 shows the bias of estimating the statistic derived in Equation (6). One can observe that our bootstrapping method still reduces the bias of both Metropolis random walk and simple random walk significantly (as high as 80%) when the number of initial points varies from $M = 4$ to $M = 18$. Furthermore, our bootstrapping method reduces more bias than the Jackknife method in most cases when the number of initial points varies from $M = 4$ to $M = 18$.

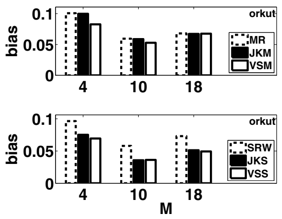
Lessons learned. Under different number of initial points, our bootstrapping method reduces the bias of both Metropolis random walk and simple random walk significantly (i.e., as high as 80%). It can also reduce the bias of the Jackknife method significantly in most cases , i.e., as high as 60%.



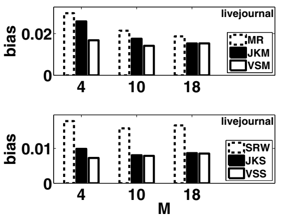
(a) Amazon



(b) Wiki



(c) orkut

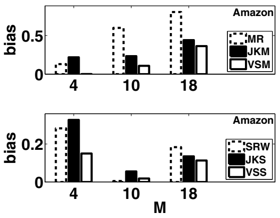


(d) livejournal

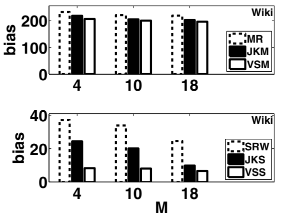
Fig. 3. Impact of M on bias with Metropolis random walk (MR) and simple random walk (SRW) as baselines. [**community as attribute**].

D. Impact of Statistical Estimation Model

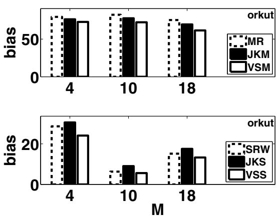
To study the impact of statistical estimation model on bias reduction, we vary the parameter c in the statistical estimation model derived in Equation (7) from 1 to 3. Figure 5 shows the bias of estimating the statistic derived in Equation (7). One can observe that our bootstrapping algorithm **VSM** can reduce the bias of Metropolis random walk **MR** by as high as 30%. This reduction ratio varies slightly as the parameter of the statistical estimation model varies from $c = 1$ to $c = 3$. When Metropolis random walk serves as the baseline, our bootstrapping method **VSM** reduces the bias of Jackknife method **JKM** by as high as 30%. Furthermore, our bootstrapping method **VSS** can reduce



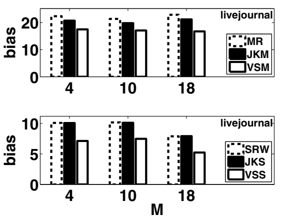
(a) Amazon



(b) Wiki



(c) orkut

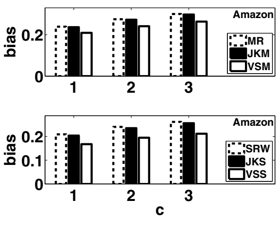


(d) livejournal

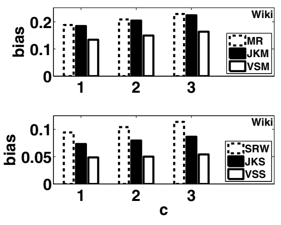
Fig. 4. Impact of M on bias with Metropolis random walk (MR) and simple random walk (SRW) as baselines. [**degree as attribute**].

the bias of simple random walk **SRW** by as high as 50% and this reduction ratio varies slightly when c varies from 1 to 3. When simple random walk serves as the baseline, our bootstrapping method **VSS** can reduce the bias of the Jackknife method **JKS** by as high as 40%.

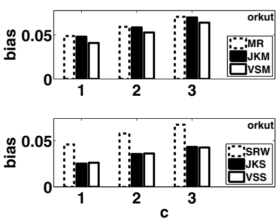
Lessons learned. Under different statistical estimation models, our bootstrapping method reduces the bias of both Metropolis random walk and simple random walk significantly (by as high as 50%). It can also reduce the bias of Jackknife method by as high as 40%.



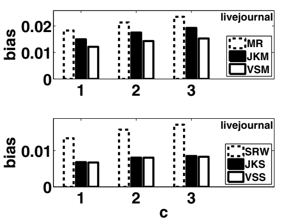
(a) Amazon



(b) Wiki



(c) orkut



(d) livejournal

Fig. 5. Impact of statistical estimation model c on bias with Metropolis random walk (MR) and simple random walk (SRW) as baselines. [**community as attribute**]

VII. Experiment II: the Mean Square Error

In this section, we conduct experiments to show how to use our bootstrapping algorithm to attain different trade-offs between the sample complexity and mean square error of an estimator. Experiment results further show that our bootstrapping method can reduce the mean square error of an estimator significantly by 1000 random walks.

A. Experiment Setting

We consider the same experiment setting as Section VI, except that we study the mean square error of each algorithm when the total number of parallel random walks is small. We use Monte method to estimate the mean square error of each algorithm. In particular, we repeat each algorithm for 1000 times, and use the average of the outputs in these 1000 times to estimate the mean square error. Following previous works [9], [10], [33], [34], we consider the relative mean square error (RMSE), i.e.,

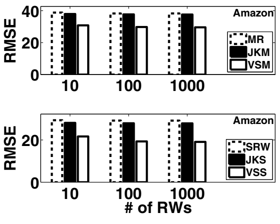
$$\texttt{RMSE}(\widehat{\Theta}) = \frac{\texttt{MSE}(\widehat{\Theta})}{\theta^2},$$

to eliminate the scale bias. Due to page limit, we only consider the case that community serves as attribute. For the case of node degree serving as attribute, one can expect similar results, because the trade-off between the sample complexity and mean square error is governed by the bias reduction and these two cases have similar bias reduction.

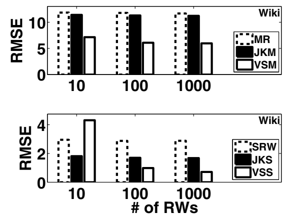
B. Impact of Number of Random Walkers

Note that the number of random walks equals MN. We set all the parameters except N as their default values stated in Section VI. We vary N from 1 to 100 such that the number of random walks MN varies from 10 to 1000. Figure 6 shows the RMSE of six algorithms described in Section VI-A evaluated on four datasets in Table I, where the statistic in estimation is derived in Equation (7). One can observe that when the number of random walks is small, i.e., $MN = 10$, our **VSM** may have a larger RMSE than **MR** and **JKM** in some cases. This is because our **VSM** has a larger variance than **MR** and **JKM** and when the total number of random walks is small, the variance dominates. When the number of random walks is large, i.e., $MN = 1000$, our **VSM** reduces the RMSE of **MR** (**JKM**) by as high as 50% (40%). This is because when the total number of random walks is large, the bias dominates. Furthermore, when the number of random walks is small, i.e., $MN = 10$, our **VSS** may have a larger RMSE than **SRW** and **JKS** in some cases. When the number of random walks is large, i.e., $MN = 1000$, our **VSS** reduces the RMSE of **SRW** (**JKS**) by as high as 70% (50%).

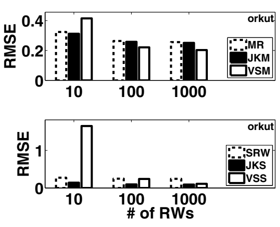
Lessons learned. Our bootstrapping method reduces the RMSE of Metropolis random walk and simple random walk significantly (i.e., as high as 70%) by no more than one thousand random walks and it can also reduce the bias of Jackknife method by as high as 50%.



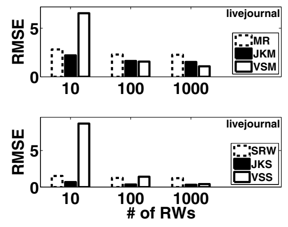
(a) Amazon



(b) Wiki



(c) orkut



(d) livejournal

Fig. 6. Impact of number of random walks on the RMSE with Metropolis random walk (MR) and simple random walk (SRW) as baselines. [**community as attribute**]

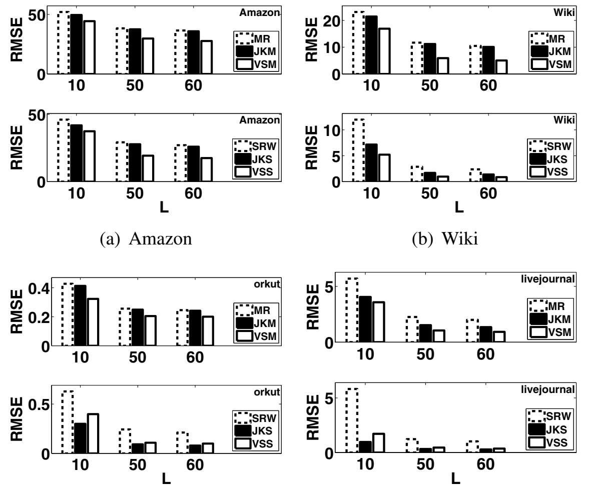
C. Impact of Sample Length

We vary the sample length L from 10 to 60. We set the number of random walks to be 1000. All the other parameters are set as default values. Figure 7 shows the RMSE of six algorithms described in Section VI-A under the statistic derived in Equation (7). One can observe that the RMSE of these six algorithms decreases as the sample length L increases. Furthermore, our **VSM** reduces the RMSE of **MR** and **JKM** by as high as 50%. Lastly, our **VSS** reduces the RMSE of **SRW** (**JKS**) by as high as 60% (40%).

Lessons learned. Under different sample lengths and one thousand random walks, our bootstrapping method reduces the RMSE of Metropolis random walk and simple random walk significantly (i.e., as high as 60%), and it can also reduce the RMSE of Jackknife method by as high as 50%.

D. Impact of Initial Points

We consider the same setting as Section VII-C, except we set the sample length L to be 50 and vary the number of initial points M from 4 to 18. Figure 8 shows the RMSE of six algorithms described in Section VI-A under the statistic derived in Equation (7). One can observe that our bootstrapping algorithm **VSM** can reduce the RMSE of Metropolis random walk **MR** by as high as 50%. This reduction ratio varies slightly as we vary the number of initial points from $M = 4$ to $M = 18$. Our bootstrapping method **VSM** reduces the RMSE of Jackknife method **JKM** by as high as 40%. Furthermore, our bootstrapping method **VSS** can reduce the RMSE of simple random walk **SRW** by as high as 70% and this reduction ratio varies slightly when the number of initial points varies from $M = 4$ to $M = 18$. Lastly, when simple



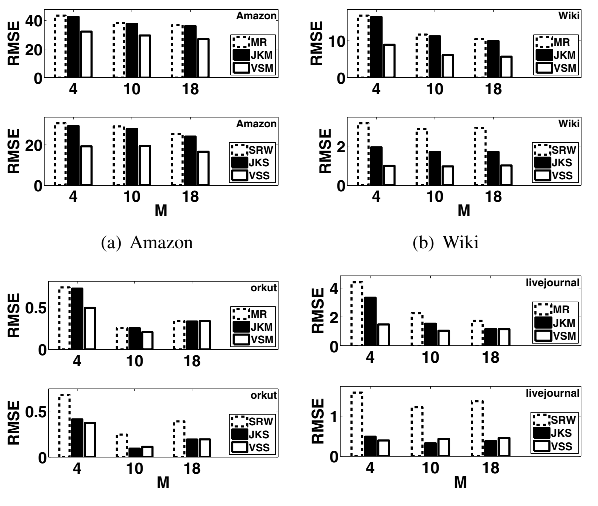
(a) Amazon (b) Wiki

(c) orkut (d) livejournal

Fig. 7. Impact of walk length L on the RMSE with Metropolis random walk (MR) and simple random walk (SRW) as baselines. [**community as attribute**]

random walk serves as the baseline, our **VSS** reduces the RMSE of Jackknife method **JKS** by as high as 50%.

Lessons learned. Under different number of initial points and one thousand random walks, our bootstrapping method reduces the RMSE of both Metropolis random walk and simple random walk significantly (i.e., as high as 70%) and it can also reduce the RMSE of Jackknife by as high as 50%.



(a) Amazon (b) Wiki

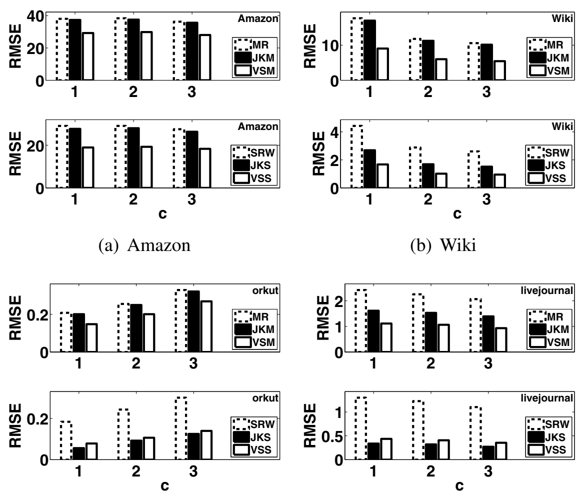
(c) orkut (d) livejournal

Fig. 8. Impact of number of initial points M on the RMSE with Metropolis random walk (MR) and simple random walk (SRW) as baselines. [**community as attribute**]

E. Impact of Statistical Estimation Model

We vary the parameter c of Equation (7) from 1 to 3. All the other parameters are set the same as Section VII-C, except the sample length L is set to be 50. Figure 9 shows the RMSE of six algorithms under the statistic derived in Equation (7). One can observe that our bootstrapping algorithm **VSM** can reduce the RMSE of Metropolis random walk **MR** by as high as 50%. This reduction ratio varies slightly as we vary the parameter of the statistical estimation model from $c = 1$ to $c = 3$. Our bootstrapping method **VSM** reduces the RMSE of Jackknife method **JKM** by as high as 50%. Furthermore, our bootstrapping method **VSS** can reduce the RMSE of simple random walk **SRW** by as high as 70% and this reduction ratio varies slightly when c varies from 1 to 3. Lastly, when simple random walk serves as the baseline, our bootstrapping method **VSS** reduces the RMSE of the Jackknife method **JKS** by as high as 40%.

Lessons learned. Under different statistical estimation models and one thousand random walks, our bootstrapping method reduces the RMSE of both Metropolis random walk and simple random walk significantly (by as high as 70%) and it can also reduce the RMSE of Jackknife method by as high as 40%.



(a) Amazon (b) Wiki

(c) orkut (d) livejournal

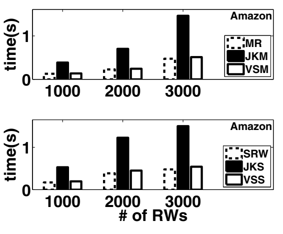
Fig. 9. Impact of statistical estimation model c on the RMSE with Metropolis random walk (MR) and simple random walk (SRW) as baselines. [**community as attribute**]

F. Scalability Analysis

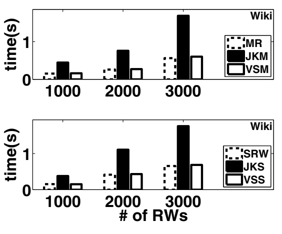
Note that the above reduction of RMSE is achieved by 1000 parallel random walks. To study the scalability of our framework, we vary the total number of parallel random walks MN from 1000 to 3000 (fix $M = 10$). All the other parameters are set as their default values stated in Section VI. We first study the worst case running time of our algorithm on a workstation (two Inter Xeon Platinum 8275 processors, each processor has 24 cores, and 192GB memory). In particular, we run all these random walks serially and study the running time. Figure 10 shows the running time of six algorithms, i.e., **MR, JKM**, **VSM**, **SRW, JKS** and **VSS**, evaluated on four datasets in Table I. One can observe that the running time of our **VSM** (**VSS**) is almost the same as that of **MR** (**SRW**). The running time of **JKM** (**JKS**) is at least two times of that of our **VSM** (**VSS**). This implies that our bootstrapping method only adds a negligible burden on the running time of baseline random walk algorithms, i.e., **MR** and **SRW**. Furthermore, even when $MN = 3000$ the running time of our **VSM** (**VSS**) is around 0.5 seconds.

If we apply parallel tricks to optimize our algorithm, the running time of our algorithm will be much shorter. In fact, a number of graph analytic systems were developed recently, which enable one to run millions of random walks in parallel on consumer-level personal computers [22]–[26]. Namely, these systems support the scalability of our work. Our work is orthogonal to them, so we do not go into details on them.

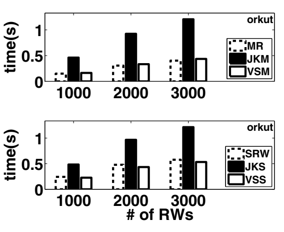
Lessons learned. Our framework adds a negligible burden on the running time of baseline random walk algorithms and it only require one thousand random walks. A number of recent graph analytic systems enable one to run millions of random walks in parallel on consumer-level personal computers. Thus, our framework has good foundation of scalability.



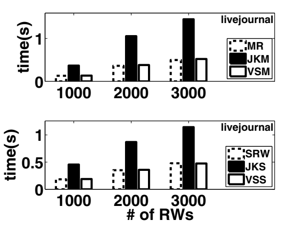
(a) Amazon



(b) Wiki



(c) orkut



(d) livejournal

Fig. 10. Impact of number of random walks on the time consumption with Metropolis random walk (MR) and simple random walk (SRW) as baselines. [**community as attribute**]

VIII. **Related Work**

To the best of our knowledge, this is the first paper applying bootstrapping techniques to optimize random walk based statistical estimation over attribute graphs. We discuss previous with respect to *random walk* and *bootstrapping technique*.

Random walk. Random walk sampling is a mainstream method to generate representation samples from large scale graphs [16]. Two fundamental random walk sampling algorithms are (1) the simple random walk [31], and (2) the Metropolis random walk [30]. A number of variants of random walk sampling algorithms were proposed to improve the estimation accuracy. Rasti *et. al.* [35] proposed an algorithm which incorporate respondent-driven sampling into Metropolis random walk. Ribeiro *et. al.* [33] developed a coordinated multidimensional random walk sampling algorithm. Kurant *et. al.* [9] proposed a stratified weighted random walk algorithm. Jin *et. al.* [21] and Xu *et. al.* [34] proposed random walk sampling algorithms with jumps. Lu *et. al.* [3] developed an algorithm to approximate the bias of estimating the population size via random walk. Lee *et. al.* [10] proposed two algorithms to reduce the asymptotic variance of estimators: (1) non-backtracking random walk and (2) random walk with delayed acceptance. Li *et. al.* [20] further extended the delayed acceptance method to Metropolis random walk, which is shown to further reduce the asymptotic variance. Lu *et. al.* [36] showed that the harmonic mean estimator for average degree can reduce the estimation variance significantly. Zhou *et. al.* [8], [28], [37] proposed history dependent random walk sampling algorithms, which are shown to have a fast convergence speed. Li *et. al.* [15] further improved them by considering the walking history and next-hop candidates. Essentially, all these algorithms improve estimation accuracy via designing random walk strategies, i.e., they focus on generating samples. These algorithms only use simple average to do the estimation. Our framework is orthogonal to their works and complements them. First, we focus on how to utilize samples to produce accurate estimation. In particular, we achieve this by applying bootstrapping techniques to design a bias reduction algorithm. Second, our framework utilizes recent graph processing systems which can run millions of random walks on a consumer-level personal computer. Third, our framework is generic and it can be applied to improve the estimation accuracy of any random walk strategies.

Our work is orthogonal to works on parallel random walk algorithms and systems [22]–[26] in the sense that it is independent of the design of parallel random walk algorithms, i.e., it can be applied to a broad class of parallel random walk algorithms.

Bootstrapping. Bootstrapping is a technique for statistical estimation. There are a variety of bootstrapping techniques such as Efron bootstrap and Jackknife for different applications or settings [27], [32], [38]–[40]. Model-based bootstrapping techniques require a large number of samples to accurately reconstruct the model. Thus it is not suitable for random walk sampling applications because the sample size is usually small. Model-free bootstrapping techniques such as Jackknife have the issue of invalid sub-samples in random walk sampling applications. Our work proposes a new variant of the Jackknife method with sub-sample selection which is fine tuned for the random walk sampling algorithms. In our framework we allow the variance to increase whenever the bias can be further reduced. We overcome the increased variance by applying multiple parallel random walks.

IX. Conclusion

This paper develops an algorithmic framework to improve the accuracy of random walk based statistical estimation over graphs. We apply the bootstrapping technique to design a bias reduction algorithm. Our bias reduction algorithm has a new feature that it allows the variance to increase whenever the bias can be further reduced. The increased variance may lead to large error of the estimator. We use multiple parallel random walks to reduce this variance, and it can be reduced to be arbitrarily small by a sufficiently large number of random walks. Our algorithmic framework enables one to attain different trade-offs between the sample complexity and the error of statistical estimation. Also, our bias reduction algorithm is generic and can be applied to optimize a large class of random walk sampling algorithms. To demonstrate the versatility of our framework, we apply it to optimize the Metropolis and simple random walk sampling. Extensive experiments on four public datasets confirm the effectiveness and efficiency of our algorithmic framework.

X. Acknowledgment

The work of John C.S. Lui is supported in part by the GRF 14200420. The work of Hong Xie is supported in part by the National Nature Science Foundation of China (61902042), Chongqing Natural Science Foundation (cstc2020jcyj-msxmX0652), Chongqing High-Technology Innovation and Application Development Funds (cstc2019jscx-msxmX0422, cstc2019jscx-fxyd0385), and the Fundamental Research Funds for the Central Universities (2020CDJ-LHZZ-057). Hong Xie is the corresponding author.

References

[1] M. Kim and J. Leskovec, "Modeling social networks with node attributes using the multiplicative attribute graph model," in *Proc. of UAI*, 2011.

[2] G. B. Giannakis, Y. Shen, and G. V. Karanikolas, "Topology identification and learning over graphs: Accounting for nonlinearities and dynamics," *Proc. of IEEE*, vol. 106, no. 5, pp. 787–807, 2018.

[3] J. Lu and D. Li, "Bias correction in a small sample from big data," *IEEE TKDE*, vol. 25, no. 11, pp. 2658–2663, 2012.

[4] K. Nakajima and K. Shudo, "Estimating properties of social networks via random walk considering private nodes," in *Proc. of ACM SIGKDD*, 2020.

[5] X. Yang, H. Steck, Y. Guo, and Y. Liu, "On top-k recommendation using social networks," in *Proc. of ACM RecSys*, 2012.

[6] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng, "A model-based approach to attributed graph clustering," in *Proc. of ACM SIGMOD*, 2012.

[7] S. Zhang, J. Yang, and V. Cheedella, "Monkey: Approximate graph mining based on spanning trees," in *Proc. of IEEE ICDE*, 2007.

[8] Z. Zhou, N. Zhang, Z. Gong, and G. Das, "Faster random walks by rewiring online social networks on-the-fly," *ACM TODS*, vol. 40, no. 4, pp. 1–36, 2016.

[9] M. Kurant, M. Gjoka, C. T. Butts, and A. Markopoulou, "Walking on a graph with a magnifying glass: stratified sampling via weighted random walks," in *Proc. of ACM SIGMETRICS*, 2011.

[10] C.-H. Lee, X. Xu, and D. Y. Eun, "Beyond random walk and metropolis-hastings samplers: why you should not backtrack for unbiased graph sampling," *ACM SIGMETRICS Performance evaluation review*, vol. 40, no. 1, pp. 319–330, 2012.

[11] X. Xu, C.-H. Lee *et al.*, "Challenging the limits: Sampling online social networks with cost constraints," in *Proc. of IEEE INFOCOM*, 2017.

[12] S. Agarwal, "Ranking on graph data," in *Proc. of ICML*, 2006.

[13] C. E. Priebe, D. L. Sussman, M. Tang, and J. T. Vogelstein, "Statistical inference on errorfully observed graphs," *J. Comput. Graph. Stat*, vol. 24, no. 4, pp. 930–953, 2015.

[14] D. Zhou and B. Schölkopf, "A regularization framework for learning from graph data," in *ICML 2004 Workshop on Statistical Relational Learning and Its Connections to Other Fields (SRL 2004)*, 2004, pp. 132–137.

[15] Y. Li, Z. Wu, S. Lin, H. Xie, M. Lv, Y. Xu, and J. C. Lui, "Walking with perception: Efficient random walk sampling via common neighbor awareness," in *Proc. of IEEE ICDE*, 2019.

[16] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou, "Walking in facebook: A case study of unbiased sampling of osns," in *Proc. of IEEE Infocom*, 2010.

[17] X. Chen, Y. Li, P. Wang, and J. C. Lui, "A general framework for estimating graphlet statistics via random walk," *Proc. of VLDB*, 2016.

[18] P. Wang, J. Zhao, X. Zhang, Z. Li, J. Cheng, J. C. Lui, D. Towsley, J. Tao, and X. Guan, "Moss-5: A fast method of approximating counts of 5-node graphlets in large graphs," *IEEE TKDE*, vol. 30, no. 1, pp. 73–86, 2017.

[19] J. Zhao, P. Wang, J. C. Lui, D. Towsley, and X. Guan, "Sampling online social networks by random walk with indirect jumps," *Data Mining and Knowledge Discovery*, vol. 33, no. 1, pp. 24–57, 2019.

[20] R.-H. Li, J. X. Yu, L. Qin, R. Mao, and T. Jin, "On random walk based graph sampling," in *Proc. of IEEE ICDE*, 2015.

[21] L. Jin, Y. Chen, P. Hui, C. Ding, T. Wang, A. V. Vasilakos, B. Deng, and X. Li, "Albatross sampling: robust and effective hybrid vertex sampling for social graphs," in *Proc. of ACM MobiArch*, 2011.

[22] A. Kyrola, "Drunkardmob: billions of random walks on just a pc," in *Proc. of ACM RecSys*, 2013.

[23] K. Vora, G. Xu, and R. Gupta, "Load the edges you need: A generic i/o optimization for disk-based graph processing," in *Proc. of USENIX ATC*, 2016.

[24] H. Liu and H. H. Huang, "Graphene: Fine-grained {IO} management for graph computing," in *Proc. of USENIX FAST*, 2017.

[25] K. Yang, M. Zhang, K. Chen, X. Ma, Y. Bai, and Y. Jiang, "Knightking: a fast distributed graph random walk engine," in *Proc. of ACM SOSP*, 2019.

[26] R. Wang, Y. Li, H. Xie, Y. Xu, and J. C. Lui, "Graphwalker: An i/o-efficient and resource-friendly graph analytic system for fast and scalable random walks," in *Proc. of USENIX ATC*, 2020.

[27] B. Efron and R. J. Tibshirani, *An introduction to the bootstrap*. CRC press, 1994.

[28] Z. Zhou, "Faster sampling over online social networks," Ph.D. dissertation, The George Washington University, 2015.

[29] P. Yi, H. Xie, Y. Li, and J. C. Lui, *A Bootstrapping Approach to Optimize Random Walk Based Statistical Estimation over Graphs: Full Version*, 2020, https://1drv.ms/b/s!AkqQNKuLPUbEii8EbThlZQyHpwid?e=IJiRsh.

[30] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, 1953.

[31] L. Lovász *et al.*, "Random walks on graphs: A survey," *Combinatorics, Paul erdos is eighty*, vol. 2, no. 1, pp. 1–46, 1993.

[32] B. Efron, *The jackknife, the bootstrap, and other resampling plans*. Siam, 1982, vol. 38.

[33] B. Ribeiro and D. Towsley, "Estimating and sampling graphs with multidimensional random walks," in *Proc. of ACM IMC*, 2010.

[34] X. Xu, C.-H. Lee *et al.*, "A general framework of hybrid graph sampling for complex network analysis," in *Proc. of IEEE Infocom*, 2014.

[35] A. H. Rasti, M. Torkjazi, R. Rejaie, N. Duffield, W. Willinger, and D. Stutzbach, "Respondent-driven sampling for characterizing unstructured overlays," in *Proc. of IEEE INFOCOM*, 2009.

[36] J. Lu and H. Wang, "Variance reduction in large graph sampling," *Information Processing & Management*, vol. 50, no. 3, pp. 476–491, 2014.

[37] Z. Zhou, N. Zhang, and G. Das, "Leveraging history for faster sampling of online social networks," *Proc. of VLDB*, 2015.

[38] A. C. Davison and D. V. Hinkley, *Bootstrap methods and their application*. Cambridge university press, 1997, vol. 1.

[39] J. Shao and D. Tu, *The jackknife and bootstrap*. Springer Science & Business Media, 2012.

[40] S. N. Lahiri, *Resampling methods for dependent data*. Springer Science & Business Media, 2013.