Sampling Multiple Nodes in Large Networks: Beyond Random Walks

Omri Ben-Eliezer
Harvard University &
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
omrib@mit.edu

Talya Eden
Boston University &
Massachusetts Institute of Technology
Massachusetts, USA
teden@mit.edu

Joel Oren
Bosch Center for Artificial Intelligence
Haifa, Israel
joel.oren@il.bosch.com

Dimitris Fotakis
National Technical University of Athens
Athens, Greece
fotakis@cs.ntua.gr

ABSTRACT

Sampling random nodes is a fundamental algorithmic primitive in the analysis of massive networks, with many modern graph mining algorithms critically relying on it. We consider the task of generating a large collection of random nodes in the network assuming limited query access (where querying a node reveals its set of neighbors). In current approaches, based on long random walks, the number of queries per sample scales linearly with the mixing time of the network, which can be prohibitive for large real-world networks. We propose a new method for sampling multiple nodes that bypasses the dependence in the mixing time by explicitly searching for less accessible components in the network. We test our approach on a variety of real-world and synthetic networks with up to tens of millions of nodes, demonstrating a query complexity improvement of up to ×20 compared to the state of the art.

CCS CONCEPTS

• **Theory of computation → Design and analysis of algorithms**; **Sketching and sampling**; **Theory and algorithms for application domains**; • **Information systems → Social networks**.

KEYWORDS

Graph and Network Sampling, Node Sampling

ACM Reference Format:
Omri Ben-Eliezer, Talya Eden, Joel Oren, and Dimitris Fotakis. 2022. Sampling Multiple Nodes in Large Networks: Beyond Random Walks. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM '22), February 21–25, 2022, Tempe, AZ, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3488560.3498383

1 INTRODUCTION

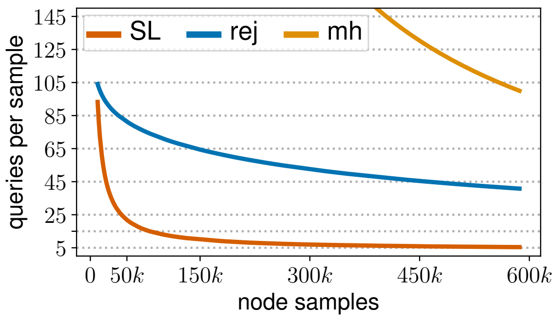
Random sampling of nodes according to a prescribed distribution has been extensively employed in the analysis of modern large-scale networks for more than two decades [22, 32]. Given the massive sizes of modern networks, and the fact that they are typically accessible through node (or edge) queries, random node sampling offers the most natural approach, and sometimes virtually the only approach, to fast and accurate solutions for network analysis tasks. These include, for example, estimation of the order [29], average degree and the degree distribution [11, 15, 59], number of triangles [2], clustering coefficient [51], and betweenness centrality [7], among many others. Moreover, node sampling is a fundamental primitive used by many standard network algorithms for quickly exploring networks, e.g., for detection of frequent subgraph patterns [42] or communities [44, 57], or for mitigating the effect of undesired situations, such as teleport in PageRank [21]. Hence, a significant volume of recent research is devoted to the efficiency of generating random nodes in large social and information networks; see, e.g., [8, 9, 26, 39, 41, 46, 60] and the references therein.

Problem formulation. We consider the task of implementing a *sampling oracle* that allows one to sample multiple nodes in a network according to a prescribed distribution (say, the uniform distribution). The collection of sampled nodes should be independent and identically distributed. Standard algorithms for random node sampling assume query access to the nodes of the network, where querying a node reveals its neighbors. As a starting point, we are given access to a single node from the network, and our goal is to generate a (possibly large) set of samples from the desired distribution using few queries. A typical efficiency measure is the amortized *query complexity* — the total number of node queries divided by the number of sampled nodes. This extends the framework proposed by Chierichetti et al. [8, 9], who studied the query complexity of sampling a *single* node. For simplicity, we focus on the important case of the *uniform distribution*, but our approach can be easily generalized to any natural distribution. We consider the regime where the desired number of node samples N is large.

Random walks and their limitations. Most previous work on node sampling has focused on random-walk-based approaches,

WSDM '22, February 21–25, 2022, Tempe, AZ, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9132-0/22/02.
https://doi.org/10.1145/3488560.3498383

Talya Eden was supported by the NSF Grant CCF-1740751, Eric and Wendy Schmidt Fund for Strategic Innovation, Ben-Gurion University of the Negev and the computer science department of Boston University. Dimitris Fotakis was partially supported by NTUA Basic Research Grant (PEBE 2020) "Algorithm Design through Learning Theory: Learning-Augmented and Data-Driven Online Algorithms - LEADAlgo".
Source code: `https://github.com/omribene/sampling-nodes`.



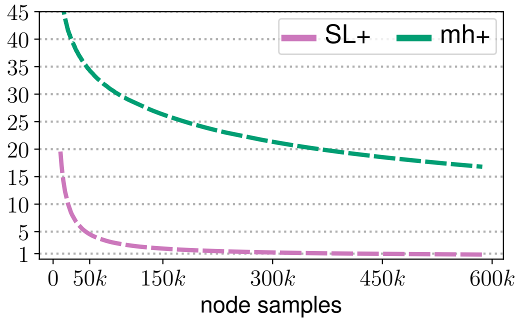


Figure 1: Amortized query complexity per sample in SinaWeibo, a network with 58.6M nodes and 261M edges. Our algorithm for the standard query model, SampLayer, is compared to rejection sampling and Metropolis-Hastings random walks (left). The variant for the stronger query model, SampLayer+, is compared to the stronger variant of MH (right).

which naturally exploit node query access to the network. They are versatile and achieve remarkable efficiency [8, 10, 26, 39]. The random walk starts from a seed node and proceeds from the current node to a random neighbor, until it (almost) converges to its stationary distribution. Then, a random node is selected according to the walk's stationary distribution, which can be appropriately modified if it differs from the desired one [8]. The number of steps before a random walk (almost) converges to the stationary distribution is called the *mixing time*, and usually denoted by t_{mix}.

RW-based approaches are very effective in highly connected networks with good expansion properties, where the mixing time is logarithmic [25]. In most real-world networks, however, the situation is more complicated. It is by now a well-known phenomenon that the mixing time in many real-world social networks is in the order of hundreds or even thousands, much higher than in idealized expander networks [13, 40, 43]. As part of this work, we prove lower bounds for sampling multiple nodes: sampling a collection of N (nearly) uniform and uncorrelated random nodes using a random walk may require $\Omega(N \cdot t_{\text{mix}})$ queries under standard assumptions on the network structure. Given the multiplicative dependence in t_{mix}, this bound becomes prohibitive as N grows larger.

1.1 Our Contribution

In light of the above discussion, we ask the following question:

Can we design a highly query-efficient method for sampling a large number of nodes that does not depend multiplicatively on the network's mixing time?

We answer this in the affirmative by presenting a novel algorithm for sampling nodes with a query complexity that is *up to a factor of* 20 *smaller* than that of state of the art random walk algorithms. To the best of our knowledge, this is the first node sampling method that is not based on long random walks.

Lower bound for random walks. We present an $\Omega(N \cdot t_{\text{mix}})$ lower bound for sampling N uniform and independent nodes from a network using naive random walks (that do not try to learn the network structure). Our lower bound construction is a graph consisting of a large expander-like portion and many small components connected to it by bridges, a structure that is very common among large social networks [38]. The main intuition is that sampling N nodes from the network requires the walk to visit many small communities, and thus cross $\Theta(N)$ bridges, which in expectation takes $\Theta(t_{\text{mix}})$ queries per bridge, and $\Theta(N \cdot t_{\text{mix}})$ in total.

Bypassing the multiplicative dependency. We present a new algorithm for sampling multiple nodes, SampLayer, whose query complexity does not depend multiplicatively on the mixing time. Our algorithm learns a structural decomposition of the network into one highly connected part and many small peripheral components. We also present a stronger variant of our algorithm, SampLayer+, that works in a more expressive query model, where querying a node also reveals the degrees (and not just the identifiers) of its neighbors.[1] Our approach is inspired by the core-periphery perspective on social networks [48]. We start by exploring the graph with a random walk that is biased towards higher degree nodes. With the high degree nodes in hand, we build a data structure that partitions all non-neighbors of these nodes into extremely small components. Then, we use the data structure to quickly reach these components (and subsequently, sample from them); the process involves running a BFS inside the reached component, which due to its tiny size does not require many queries.

We theoretically relate the query complexity of SampLayer to several network parameters and show that they are well-behaved in practice. The samples generated by our algorithm are provably independent and nearly uniform.

Improved empirical performance. We compare the amortized query complexity of SampLayer against those of the two most representative and standard random walk-based approaches for node sampling, rejection sampling (REJ) and the Metropolis-Hastings (MH) approach; SampLayer+ is compared against MH+, an analogue of MH in the degree-revealing model. For a complete description of REJ, MH and MH+ with a theoretical and empirical analysis of their query complexity, see the work of Chierichetti et al. [8].

We perform the comparisons on seven real world social and information networks with diverse characteristics, with the largest being SinaWeibo [58], which consists of more than 50M nodes and 250M edges. The results presented in Figures 1 and 8 and in

[1]Such strong queries are supported, e.g., by Twitter API (link1, link2).

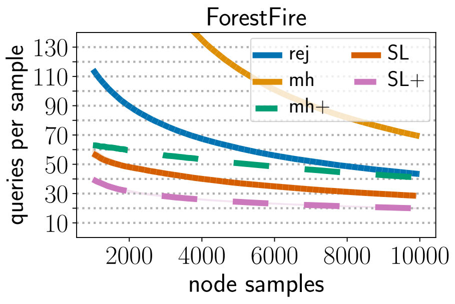


Figure 2: Amortized query complexity per sample in a Forest Fire graph with $1M$ nodes (parameters: $p_f = 0.37$, $p_b = 0.3$).

Section 4.1 show that when N is not extremely small, the query complexity of our algorithms significantly outperforms the random walk-based counterparts across the board. This holds both under the standard query model (i.e., SampLayer vs. REJ and MH) and in the stronger, degree-revealing query model (SampLayer+ vs MH+). In both models, and for all seven networks, we achieve at least 40% and up to 95% reduction in the query complexity. Remarkably, as shown in Figure 1, in some cases SampLayer may achieve a near-optimal query complexity of as little as 5 queries per sample, even when N is less than 1% of the network size. In SampLayer+ this is even more dramatic, essentially achieving one query per sample.

Generative models. One possible explanation to our findings might lie in the seminal work by Leskovec et al. [38]. In one of the most extensive analyses of the community structure in large real-world social and information networks, they examined more than 100 large networks in various domains. They discovered that most of the classical generative models at the time did not capture well the community structure and additional various properties of social networks (e.g., size of communities, their connectivity to the graph, scaling over time, etc). The *Forest Fire* generative model [36, 37] was developed to fill this gap, being more in-line with empirical findings. In this model, which is by now standard and well-investigated, edges are added in a way that creates small, barely connected pieces that are significantly larger and denser than random. We show that our algorithm performs very well on networks generated by this model (a consistent query complexity improvement of 30-50%) even for tiny core sizes; see Figure 2.

1.2 Related Work

Random-walk-based approaches for node sampling have been studied extensively in the last decade. The aforementioned work of Chierichetti et al. [8] is the closest to ours, studying the query complexity of such approaches. The analysis of Iwasaki and Shudo [26] also focuses on the average query complexity of random walks.

Using random node sampling to determine the properties of large-scale networks goes back to the seminal work of Leskovec and Faloutsos [34]. Since then, the performance of node sampling via random walks has been widely considered in the context of network parameter estimation. For example, Katzir et al. [28, 29] use random walks to estimate the network order and the clustering coefficient based on sampling and collision counting. Cooper et al. [10] present a general random-walk-based framework for estimating various network parameters; whereas Ribeiro et al. [46] extend these approaches to directed networks. Ribeiro and Towsley [45] use multidimensional random walks. Eden et al. [16–18] and Tětek and Thorup [55] study the query complexity of generating uniform edges given access to uniform nodes. Bera and Seshadhri [5] devise an accurate sublinear triangle counting algorithm that queries only a small fraction of the graph edges. For several other examples of network estimation works, see [20, 27, 30, 39, 41, 60].

In many of these works, improved query complexity is achieved by relaxing the requirement for independent samples. Understanding how dependencies between sampled nodes affect the outcome, however, inherently requires a complicated analysis tailored to the network-parameter at question. Such an analysis is not required for nodes generated by our approach, which are provably independent. From a technical viewpoint, our adaptive exploration of the network's isolated components bears some similarity to node sampling via deterministic exploration [50] and to *node probing* approaches (e.g., [6, 33, 52, 53]) for network completion [24, 31]. Given access to an incomplete copy of the network, Soundarajan et al. [52, 53] and LaRock et al. [33] discover unobserved parts via adaptive network exploration; see the survey by Eliassi-Rad et al. [19].

Utilizing core-periphery characteristics of networks for algorithmic purposes has received surprisingly little attention. The most relevant work is by Benson and Kleinberg [4], on link prediction. More weakly related is the study of k-cores of social network [1, 14], which aim to capture the subset of all "engaged" nodes by iteratively removing nodes of small degree.

2 LOWER BOUND FOR RANDOM WALKS

In this section, we quickly present the $\Omega(N \cdot t_{\text{mix}})$ lower bound on the number of queries required to sample N (nearly) independent and uniformly distributed nodes using random walks. See proof sketch in the full version [3]. For clarity, we focus our analysis on the most standard random walk, which at any given time proceeds from the current node to one of its neighbors, uniformly at random; such a random walk is used in rejection sampling (REJ). Similar lower bounds hold for other random walk variants that do not learn structural characteristics of the network, including MH and MH+.

Theorem 2.1. *For any n and $\log n \ll t \ll n^{\Theta(1)}$, there exists a graph G on n vertices with mixing time $t_{mix} = \Theta(t)$, that satisfies the following: for any $N \leq n^{\Theta(1)}$, any sampling algorithm based on uniform random walks that outputs a (nearly) uniform collection of N nodes must perform $\Omega(N \cdot t_{mix})$ queries.*

3 ALGORITHM

In order to bypass the multiplicative dependence in the mixing time, one needs to exploit structural characteristics of social networks in some way. One natural property is that the degree-distribution is top-heavy; furthermore, a large fraction of nodes in the network are well-connected to the high-degree nodes, whereas the remaining nodes decompose into small, weakly connected components [38, 48]. Figure 3 demonstrates this in a strong quantitative form. Suppose that we are able to access a collection of, say, the top 1% highest degree nodes in the network, and call these L_0. Denote their

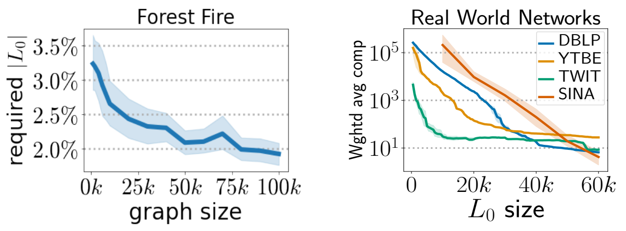


Figure 3: Very small L_0 suffices to shatter $L_{\geq 2}$-components in a Forest Fire graph (left); convergence of $L_{\geq 2}$ component sizes as $|L_0|$ grows, in four real-world networks (right).

neighbors by L_1 and the rest of the network by $L_{\geq 2}$. Does a small L_0 size suffice for $L_{\geq 2}$ to decompose into tiny components?

Our experiments indicate that the answer is positive, even if one instead generates L_0 greedily with our query access, starting from an arbitrary seed vertex. The details are given in the experimental section, but briefly, Figure 3 demonstrates that for both the Forest Fire model with standard parameters and for various real-world social networks with diverse characteristics, a very small L_0 size (ranging between 0.1% and 10% of the graph size, and in most cases about $1-2\%$) suffices for $L_{\geq 2}$ to decompose very effectively.

These results suggest a new approach to quickly reach nodes in the network: In a preprocessing phase, greedily capture L_0 as above, which decomposes the rest of the network into L_1 and $L_{\geq 2}$. L_1-nodes are easy to reach; $L_{\geq 2}$-nodes are reachable by attempting to visit a component from $L_{\geq 2}$ through a walk of length 2 from L_0, and then fully exploring the component via a BFS. Our algorithm, SampLayer, is based on this idea, also running size and reachability estimations to ensure the generated samples are close to uniform.

3.1 Algorithm Description

We next describe our algorithm, SampLayer, in detail. The algorithm runs in two phases: a structural learning phase and a sampling phase. In the first phase, the algorithm constructs a data structure providing fast access to nodes that are either very highly-connected (we call these nodes the L_0-layer) or neighbors thereof (the L_1-layer). This exploits the well-known fact that in large social and information networks, typically a large fraction of the nodes are connected to a highly influential core [48]. The node sampling itself takes place in the second phase, which uses the data structure to either sample from the core layers $L_0 \cup L_1$, or to explicitly cross bridges that lead to the small, less connected parts. These are edges from L_1 to nodes outside $L_0 \cup L_1$. We refer to these nodes as the $L_{\geq 2}$ layer. Once it reaches such a small component, the algorithm fully explores the component and uniformly samples a node from within it. Finally, our algorithm uses rejection sampling to ensure that (almost) all nodes are returned with equal probability.

Structural decomposition phase. Starting from an arbitrary node, we aim to capture the highest-degree nodes in the network. This is done by our procedure Generate-L_0 below. We add these nodes greedily, one by one, where intuitively, in every step the newly added node is the one we perceive (according to the information currently available) as the highest-degree one. We refer to this initial collection of high-degree nodes as the base layer, L_0.[2]

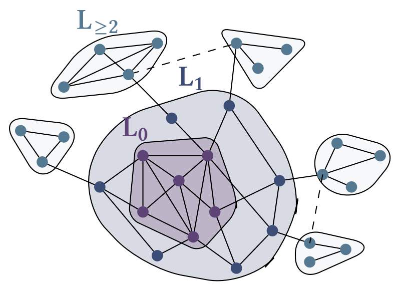


Figure 4: An illustration of a typical network and its layering by our algorithm, SampLayer. The $L_{\geq 2}$-layer components intuitively correspond to small communities that are weakly connected to the rest of the network.

```
Generate-$L_0$
Input: arbitrary vertex $v_0$, number $\ell_0$.
Output: $L_0$ of size $\ell_0$, $L_1$, $\mathcal{D}$.
(1) Query $v_0$ and let $L_0 = \{v_0\}$, $L_1 = N(v_0)$.
(2) Repeat $\ell_0 - 1$ times:
    (a) Pick $u \in L_1$ with maximum number of $L_0$ neighbors (break ties randomly).
    (b) Query $u$ and remove it from $L_1$.
    (c) Add $u$ to $L_0$ and add $N(u) \setminus L_0$ to $L_1$.
(3) Create a data structure $\mathcal{D}$ to sample edges between $L_0$ and $L_1$ uniformly at random.
(4) Return $L_0$, $L_1$ and $\mathcal{D}$.
```

The next layer, L_1, is the set of neighbors of L_0 that are not already in L_0, i.e. $L_1 = \bigcup_{v \in L_0} N(v) \setminus L_0$, where $N(v)$ denotes the set of neighbors of node v. Intuitively, the union of these two layers captures the well-connected or "expanding" part of the network. The neighbors of L_1 are denoted L_2 and the multi-layer consisting of all other nodes in the network is denoted by $L_{>2}$, where we also set $L_{\geq 2} = L_2 \cup L_{>2}$. See Figure 4 for a visualization of the layers and Figure 5 for a visualization of the structural decomposition phase of SampLayer and its variant SampLayer+.

Denote by $G_{\geq 2}$ the subgraph whose node set is $L_{\geq 2}$, and whose edge set includes all edges between L_2 and $L_{>2}$ and all edges between nodes in $L_{>2}$. Crucially, the size ℓ_0 of the generated L_0 should be sufficiently large so that the subgraph $G_{\geq 2}$ will "break" into many small connected components. (Note that we intentionally "ignore" edges between vertices that lie strictly in L_2, to make these components as small as possible.) In Section 4.2, we explore the typical size of $G_{\geq 2}$-components as a function of ℓ_0, and discuss how to determine the "correct" ℓ_0 value for the network at hand.

To complete this phase, we learn various parameters of the layering that are crucial for the sampling phase, including accurate approximations of the size of $L_{\geq 2}$ and the typical reachability of nodes in it. This is done using the procedures Estimate-Periphery-Size and Estimate-Baseline-Reachability, given in Section A.1. Specifically, estimating the size of $L_{\geq 2}$ is done by considering the

[2]We note that one can modify Generate-L_0 by first conducting a random walk using part of the L_0-construction budget, and only then continuing with the above process. While the added randomness could theoretically help escaping situations where the initial node is problematic in some way or there are multiple cores in the graph, in all networks that we tested adding such a random walk did not improve the quality of L_0; in fact, the existing algorithm captured essentially all nodes with very high degrees.

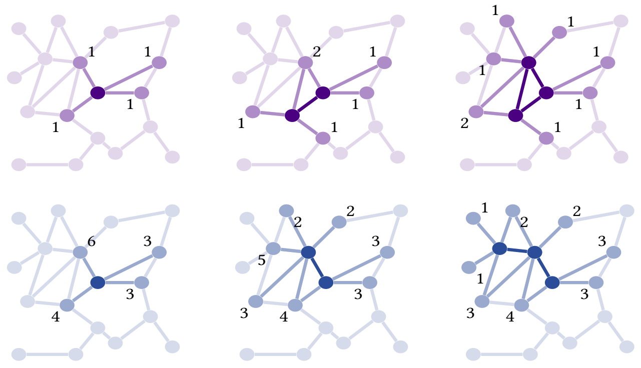


Figure 5: The structural decomposition phase, of generating (from left to right) the base layer L_0 in SampLayer (top, purple) and SampLayer+ (bottom, blue) from an arbitrary starting node. At any given step, the next layer L_1 consists of all neighbors of the L_0 nodes. The value next to each L_1-node indicates its number of neighbors in L_0 (in SampLayer) or its total degree (in SampLayer+).

bipartite graph with L_1 on one side and $L_{\geq 2}$ on the other. By sampling s_1 nodes from L_1 and $s_{\geq 2}$ nodes from $L_{\geq 2}$ (using the procedure Reach-$L_{\geq 2}$), we estimate the average degrees of the nodes of each side of the bipartite graph, from which we estimate $|L_{\geq 2}|$. The reachability distribution is approximated by calculating the reachabilities of the $s_{\geq 2}$ samples from $L_{\geq 2}$. This procedure receives as an input a parameter ε, and returns a "baseline reachability" which is approximately the ε-percentile of $L_{\geq 2}$-nodes in terms of reachability. In Section 4.1 we discuss how to practically choose s_1, $s_{\geq 2}$, and ε.

Sampling phase. Sampling from the core layers L_0 and L_1 is trivial; the challenge is to sample efficiently from $L_{\geq 2}$. Taking advantage of the layering, we sample random nodes in $L_{\geq 2}$ by combining walks of length 2 that start in L_0 and reach $L_{\geq 2}$, with a local BFS step that explores and returns a uniformly selected node in the reached $L_{\geq 2}$ component. The above process generates biased samples, as the vertices in different components have different probabilities to be reached in the initial 2-step walk. Hence, the final step in the sampling procedure is a rejection step, whose role is to unbias the distribution. Here, we compute a suitable *reachability score*, $rs(v)$ for every reached vertex. We then perform a rejection step, where the acceptance probability is inversely proportional to the reachability score of the chosen node. See Figure 6 for the pseudo-code and Figure 7 for an illustration of the sampling process.

Non-uniform distributions. For simplicity, our algorithm is presented for node generation according to the uniform distribution. We note that it can be adapted to generate other desirable distributions. For example, to conduct ℓ_p-sampling, the size estimation procedure should be replaced by a procedure that estimates the sum $\sum_{v \in L_{\geq 2}} (d(v))^p$ (and the corresponding sum for L_1), and the reachability distribution estimation should be adjusted accordingly.

3.2 Convergence to Uniformity

Our main theorem states that samples generated by our algorithm converge to (near-)uniformity. The proof builds in part on the fact

```
Sample
Input: $\bar{\ell}_{\geq 2}$ - size estimate for $L_{\geq 2}$. $rs_0$ - baseline reachability.
(Both computed in the preprocessing step)
Output: An almost uniform node in the network.
(1) Choose a layer $L_0, L_1$ or $L_{\geq 2}$ with probability proportional to their sizes $|L_0|, |L_1|, \bar{\ell}_{\geq 2}$.
(2) If $L_0$ or $L_1$ are chosen then sample a uniform node in $L_0$ or $L_1$, respectively.
(3) If $L_{\geq 2}$ is chosen, then repeatedly do:
    (a) Invoke Reach-$L_{\geq 2}$ and let $u$ and $rs(u)$ be the returned node and its reachability.
    (b) With probability $\min(\frac{rs_0}{rs(u)}, 1)$ return $u$. If not returned, repeat loop.
```

```
Reach-$L_{\geq 2}$
Input: The data structure $\mathcal{D}$.
Output: vertex $v \in L_{\geq 2}$, its component and reach. score.
(1) While no vertex $w$ chosen:
    (a) Use $\mathcal{D}$ to sample a uniform edge between $L_0$ and $L_1$. Let $u$ denote its $L_1$ endpoint.
    (b) Query $u$, and if it has neighbors in $L_{\geq 2}$, choose one of them, $w$, uniformly at random.
(2) Perform a local BFS of the component $C$ of $w$ in $G_{\geq 2}$.
(3) Choose a vertex $v$ in $C$ uniformly at random.
(4) Invoke Comp-Reachability to compute the reachability of $C$, $rs(C)$.
(5) Return $v$, and its reachability score, $rs(v) = rs(C)$.
```

```
Comp-Reachability
Input: An (already queried) component $C$ of $G_{\geq 2}$.
Output: The reachability score of $C$.
(1) $\forall v \in C \cap L_2$:
    (a) Query all $u \in N(v) \cap L_1$. For each such $u$:
        (i) Let $d^-(u) = |N(u) \cap L_0|$, and $d^+(u) = |N(u) \cap L_2|$.
        (ii) $\forall u \in N(v) \cap L_1$ set $rs(u) = \frac{d^-(u)}{d^+(u)}$.
        (iii) Set $rs(v) = \sum_u rs(u)$.
(2) Return $rs(C) = \frac{1}{|C|} \sum_{v \in C} rs(v)$.
```

Figure 6: The sampling procedures.

that our algorithm can estimate the size of $L_{\geq 2}$ given sufficient effort in the preprocessing phase. Proofs are given in the full version [3]. For experiments on the size estimation procedures, see Section A.4.

Theorem 3.1. *If our size estimation for $L_{\geq 2}$ is in $(1 \pm o(1))|L_{\geq 2}|$, and if the baseline reachability rs_0 used in our algorithm is the $o(1)$-percentile in the reachability distribution, then the output node distribution of* Sample *is $o(1)$-close to uniform in total variation distance.*

We stress that even in the case that the L_0 generation process is unsuccessful (in a sense that it does not break the $L_{\geq 2}$ vertices into small components), it *always* holds that our algorithm returns a close to uniform vertex, provided that the size and reachability estimates are correct. That is, the correctness of our algorithm holds for *any* given L_0 (with high probability), and only the query complexity of subsequent sampling might be negatively affected, e.g., due to a high expected component size value.

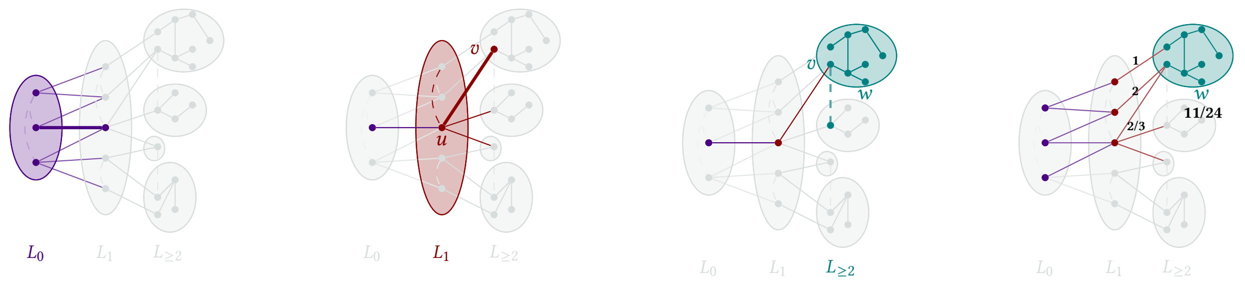


Figure 7: Sampling a node from $L_{\geq 2}$ in SampLayer. We start by picking a uniform edge L_0 and L_1, let u denote its L_1-endpoint. We next traverse a random edge from u to $v \in L_2$, if one exists. We then fully explore the $L_{\geq 2}$-component C containing v, choosing a uniformly random node $w \in C$. A final rejection step estimates how likely it is for the process to end at w.

3.3 Query Complexity

In this section, we analyze the query complexity of the sampling phase of our approach. We show here that the query complexity of sampling nodes using SampLayer is bounded as a function of several parameters related to the layered structure we maintain. The starting point of our analysis is immediately after the preprocessing phase is completed. In particular, L_0 and L_1 are already known, as well as a good estimate of the size of $L_{\geq 2}$. In addition, we have the ability to sample uniformly random edges between L_0 and L_1 without making any queries. We make the following assumptions.

- **Reachability distribution.** We assume that the reachabilities of nodes in L_2 are relatively balanced: the reachability score $rs(v)$ of every $v \in L_2$ satisfies $rs_0 \leq rs(v) \leq c \cdot rs_0$, where rs_0 is viewed as the "base reachability", and $c > 1$ is not large. We empirically verify this in Section 4.2.
- **Entry points.** Let α denote the fraction of edges e between L_0 and L_1, for which the L_1-endpoint of e has neighbors in L_2. Then α is precisely the probability that a single attempt at reaching $L_{\geq 2}$ succeeds (without taking the rejection step into account). In practice, α is known to be well-behaved [48], as most bridges to $L_{\geq 2}$ occur at higher-degree nodes of L_1.
- **Component sizes.** Set $w = \mathbb{E}[\,|CC(v)|\,]$, where $v \in L_{\geq 2}$ is (distributed as) the result of a single run of our procedure Reach-$L_{\geq 2}$, and $CC(v)$ is the $L_{\geq 2}$-component in which v resides. Intuitively, w measures the sizes of components that we reach, and we empirically validate that it is typically small on both synthetic and real-world networks, see Section 4.2.
- **Degrees of component nodes.** We assume that for all components C of $L_{\geq 2}$, the number of bridges from C to the rest of the network is at most $d \cdot |C|$, for a small integer d. This is in line with the well-observed fact [38, 48] that peripheral components are weakly connected to the network.

Our experiments verify that the parameters discussed here are indeed well-behaved when the size ℓ_0 of L_0 is chosen correctly – see Section 4.2 for more details. We bound the expected query complexity of our sampling algorithm as a function of the above parameters. Crucially, this implies that, once the preprocessing phase is complete, the query complexity does not directly depend on the network size or on the mixing time of long random walks. Due to space constraints, the proof appears in the full version [3].

Theorem 3.2. *The expected query complexity of sampling a single node using SampLayer is* $O\left(c \cdot \left(\frac{1}{\alpha} + wd\right)\right)$.

4 EMPIRICAL RESULTS

In this section, we describe several experiments we conducted, comparing our algorithms to previous approaches which are all based on random walks (Section 4.1), and explaining the query-efficiency of our methods (Section 4.2).

4.1 Evaluation of Query Complexity

The main experiment computes the amortized number of queries per sample of our algorithm, and compares it with the corresponding query complexity of existing RW-based approaches. In the standard query model, we compare our algorithm SampLayer with two random walk-based algorithms, Rejection sampling (REJ) and Metropolis-Hastings (MH). In the stronger query model, we compare SampLayer+ to Metropolis-Hastings "plus" (MH+). The methods REJ, MH, and MH+ were all described in detail by Chierichetti et al. [8]. In REJ, the algorithm performs a standard (unbiased) random walk, where nodes are subject to rejection sampling according to their degree; in MH the neighbor transition probabilities are controlled by the neighbors' degrees. MH+ is the same as MH, but assumes the stronger query model, where a node query also reveals the degrees of its neighbors. RW-based algorithms are most commonly used to sample multiple nodes by performing a long random walk, and sampling a new node once every fixed interval to allow for re-mixing. Indeed, as discussed in Section 2, to ensure that the

Dataset	n	m	d_{avg}	L_0 size	
				SL	SL+
Epinions [47]	76K	509K	13.4	3K	1K
Slashdot [38]	82K	948K	23.1	3K	2K
DBLP [56]	317K	1.05M	6.62	30K	20K
Twitter-Higgs [12]	457K	14.9M	65.1	25K	10K
Forest Fire [36, 37]	1M	6.75M	13.5	10K	10K
Youtube [56]	1.1M	2.99M	5.27	30K	10K
Pokec [54]	1.6M	30.6M	37.5	200K	100K
SinaWeibo [58]	58.7M	261M	8.91	500K	100K

Table 1: The list of networks we considered with numbers of nodes (n), edges (m), their average degrees (d_{avg}), and L_0 sizes we selected for SampLayer and SampLayer+.

node samples will be uniform and independent, the interval length must allow the walk to mix between subsequent samples.

Setting for our algorithm. We examine seven online social and information networks of varying sizes and characteristics, taken from widely used network repositories [35, 49]. We also examine our algorithm on a network generated by the Forest Fire model with parameters $p_f = 0.37$ and $p_b = 0.3$, which are standard for this model [36]. The networks, along with their basic properties, are described in Table 1. For each network, we performed a small grid search to obtain a reasonable value for the input parameter ℓ_0 (the target size of L_0) in our algorithm. The values we used for each network are given in Table 1. For the other two input parameters, s_1 and $s_{\geq 2}$, we observed that choices of 3,000 and 200 respectively are generally sufficient for SampLayer on the first seven networks (for Epinions and Slashdot, we picked $s_1 = 1{,}000$). In SampLayer+, values of $s_1 = 1{,}000$ and $s_{\geq 2} = 100$ are generally sufficient for the seven smaller networks. Separately, for SinaWeibo we picked larger values, of $s_1 = 30k$ and $s_{\geq 2} = 3k$ for both SampLayer and SampLayer+, since the network is substantially larger.

We ran each of our algorithms SampLayer and SampLayer+ for 5-10 times on each of the eight networks; the amortized query complexity we calculated is the average over these runs. As part of our pipeline, we verified the quality of our solution by configuring the algorithm's parameters so as to ensure that the samples generated by our algorithm are close to uniform. Specifically, we fixed a small empirical threshold t (0.01 in most cases) and parameters s_1 and $s_{\geq 2}$ as above, while varying the value of the error parameter ε in our algorithm. For each choice of ε, we ran the following for 10 times: we sampled n nodes using our algorithm (parameterized by ε), where n is the graph size. In each of the runs, we calculated the empirical distance to uniformity; if the average empirical distance over the 10 runs is more than t away from the expected value for a true uniform distribution, ε is discarded. Thus, our final choice of ε ensures near-uniformity of the output samples.

Setting for random walks. As mentioned, the standard approach to sampling multiple nodes using a random walk is by running a single long walk and extracting samples from it in fixed intervals. We examined this approach in two phases: Determining a good choice for the interval length, and evaluating the query complexity.

We judiciously set interval lengths that allow for proper mixing. This is explained in detail in Appendix A.3, but briefly, we generated a large number of short random walks from the same starting point and evaluated at what point in time these walks mix. To this end, we computed in each time step, for all walks simultaneously, the empirical distance to uniformity (using the same value of the empirical threshold t as in our algorithm) or the number of collisions, which is also an indicator of distance to uniformity [23]. To ensure the variance is controlled, we ran this procedure from 3-5 different starting points for each of the algorithms REJ, MH, MH+.

To compute the amortized query complexity, we ran the random walk algorithms on each of the networks, while keeping track of the cumulative number of queries. Then, we computed the mean number of queries per sample as the walk progressed.

Main results. Figure 8 depicts the comparison results for the six smaller real-world networks. Figure 1 and 2 show the results for SinaWeibo, and for a Forest Fire generated network, respectively. The number of node samples in each of the first six networks, as well as in the FF one, is between 0.1% and 1% of the total number of nodes. While the results at the lower end, 0.1%, show the relatively steep initial price of the structural learning phase of our algorithm, the higher end of our sample size clarifies the stark differences in performance between the methods. In SinaWeibo, due to its sheer size, we considered a wider interval, from $10k$ samples (less than 0.02% of the nodes) to about $600k$ samples (1%).

As is evident in the plots, SampLayer and SampLayer+ obtained significantly improved results compared to their RW-based counterparts. In all cases, and throughout the runs (as more samples are gathered), SampLayer demonstrated a query complexity that in all cases offers query complexity savings of at least 40%, and often much more, compared to both REJ and MH. Moreover, REJ consistently required fewer queries on average than its counterpart MH. This is in line with previous results from [8]. In the more powerful query model, SampLayer+ also gave at least 50% (and almost always better) improvement over its random walk analog MH+. The most dramatic improvement was for SinaWeibo, the largest network, where SampLayer and SampLayer+ yielded reductions reaching 90% and 95% in the query complexity, respectively, compared to their random walk counterparts. Curiously, as shown in Figure 1, the query complexity of SampLayer+ in SinaWeibo was in some cases less than one query per sample. While this may seem counter-intuitive at first, we note that node samples from L_1 are generated by our algorithm without any query cost. One feature of SinaWeibo that we observed is that a large majority of the nodes in the network are located in L_1, even for small L_0 sizes. Thus, many samples do not induce any query-cost.

Interestingly, our algorithm was challenged by the DBLP network, which required a costly L_0-construction stage, resulting in an initial disadvantage. However, as is clear in the figure, our algorithm recovers at samples of at least 1,000 nodes, to quickly reach consistent improvement of about 40% compared to REJ. We believe that these difficulties stem from the fact that in DBLP, a collaboration network, nodes have a weaker tendency to connect to very high degree nodes than in most social or information networks.

4.2 Other Experiments

Structural layering parameters. In Section 3.3 we have seen that two factors mostly control the query complexity of our sampling phase: the typical size of $L_{\geq 2}$-components, and the reachability distribution of nodes in $L_{\geq 2}$. We empirically demonstrate that for the "right" size of L_0, these two factors are well-behaved, explaining the improved query complexity of our method.

Define μ as the average component size of a node in $L_{\geq 2}$. This is a weighted (biased) average, giving more weight to the larger components. To see why this weighted average is of interest, recall our definition of w from Section 3.3: the expected size of a component reached by the procedure Reach-$L_{\geq 2}$. Under the assumption that all node reachabilities in $L_{\geq 2}$ are of the same order, it holds that $w = \Theta(\mu)$. To analyze the typical size of reached components in $L_{\geq 2}$, we computed the value of μ as a function of the L_0-size. This was done five times for each network, and is demonstrated for DBLP, Youtube, Twitter-Higgs, and SinaWeibo in Figure 3 (right).

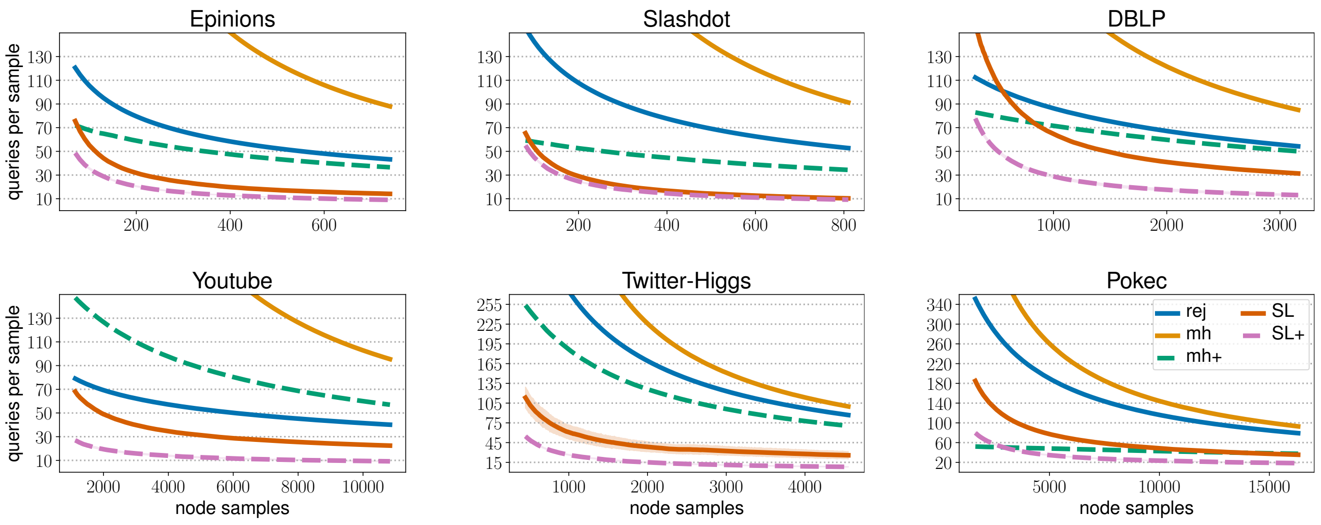


Figure 8: Amortized query complexity of our methods compared to random-walk-based node sampling methods. In the standard query model, SampLayer is compared against Rejection Sampling (REJ) and Metropolis Hastings (MH), and in the stronger query model, SampLayer+ is compared against Metropolis Hastings "plus" (MH+). For each of the networks, the number of node samples ranges between 0.1% and 1% of the network's order. See Section 4 and Table 1 for details.

Interestingly, the weighted average μ seems to decay exponentially as a function of the L_0-size (note that the y-axis here is log-scaled), until it converges to a constant. To the best of our knowledge, this exponential decay was not previously observed in the literature, and we believe it warrants further research; as the results show, the rate of decay differs majorly between different networks, and explains the choices of L_0-size that we made for the different networks: ideally, one would like L_0 to be small, while inducing a small enough μ-value (say, 10 or 20).

In Figure 3 (left), we investigated what minimal size of L_0 is required in a Forest Fire graph in order to satisfy $\mu \leq 10$. The experiment was run for different values of the graph size n, while fixing the FF parameters $p_f = 0.37$ and $p_b = 0.3$ as before. The results show that the required $|L_0|$ is clearly sublinear in n: they decay from about 3.3% for $n = 1K$ to about 2% for $n = 100K$.

Other experiments. The experiments related to the reachability distribution and the size estimation are discussed in Section A.4.

5 CONCLUSIONS AND OPEN QUESTIONS

We have presented an algorithm that supports query-efficient sampling of multiple nodes in a large social network, exhibiting the efficiency of our algorithm compared to the state of the art on a variety of datasets, that include up to tens of millions of nodes. We gave theoretical bounds on our algorithm's complexity in terms of several graph parameters. We then empirically confirmed that, in all the social networks we have examined, these graph parameters are well-behaved. One major concern is the question of generality. That is, what is the scope of networks for which our methods are suitable. As stated in the experimental section, our algorithms gave better or comparable results on all social networks we have tested, and the good performance on the Forest Fire model, a realistic generative model, further hints to wide applicability.

Our algorithm has some disadvantages compared to random walks. First, it is substantially more complicated, and its analysis does not directly depend on one structural parameter of the network (such as the mixing time for random walks). Second, it inherently relies on a data structure which has a high memory consumption, and may not always be suitable in situations that require bounded-memory algorithms. Third, it depends on the size of L_0, which differs between different networks, so it requires fine tuning according to the given network. Finally, it exploits unique properties of social networks. These properties do not hold for bounded degree graphs, or for, say, real-world road networks. For such graphs, we expect our algorithm to perform worse than random walks.

Having said all that, our work demonstrates that in various real and synthetic social networks, significant query complexity savings can be obtained using structural decompositions and careful preprocessing. We do not believe that our algorithm should replace RW-based approaches. Indeed, these algorithms are the gold standard for node sampling and provide an exceptionally versatile primitive in network analysis. Rather, one can think of our algorithms as useful alternatives when multiple node samples are required and where the networks at question have suitable community structure. It would be very interesting to further explore approaches that combine the query-efficiency of our methods with the flexibility of random walks. For example, how can our core-periphery based data structure be used to accelerate the computation of important network parameters, or the detection of community structure?

ACKNOWLEDGMENTS

The authors wish to thank Suman K. Bera and C. Seshadhri for many helpful pointers and suggestions; Flavio Chierichetti and Ravi Kumar for valuable feedback; and the anonymous reviewers, for their insightful comments.

REFERENCES

[1] José Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. K-core decomposition of internet graphs: hierarchies, self-similarity and measurement biases. *Networks & Heterogeneous Media*, 3(2):371–393, 2008.

[2] L. Becchetti, P. Boldi, Carlos Castillo, and Aristides Gionis. Efficient algorithms for large-scale local triangle counting. *ACM Trans. Knowl. Discov. Data*, 4(3):13:1–13:28, 2010.

[3] Omri Ben-Eliezer, Talya Eden, Joel Oren, and Dimitris Fotakis. Sampling multiple nodes in large networks: Beyond random walks. *CoRR*, abs/2110.13324, 2021.

[4] A. R. Benson and J. M. Kleinberg. Link prediction in networks with core-fringe data. In *Proc. World Wide Web Conference (WWW 2019)*, pages 94–104, 2019.

[5] Suman K. Bera and C. Seshadhri. How to count triangles, without seeing the whole graph. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 306–316, 2020.

[6] C.A. Bliss, C.M. Danforth, and P.S. Dodds. Estimation of global network statistics from incomplete data. *PLoS ONE*, 9, 2014.

[7] M. Borassi and E. Natale. KADABRA is an adaptive algorithm for betweenness via random approximation. *ACM J. Experimental Algorithmics*, 24(1):1.2:1–1.2:35, 2019.

[8] F. Chierichetti, A. Dasgupta, R. Kumar, S. Lattanzi, and T. Sarlós. On sampling nodes in a network. In *Proceedings of the 25th International World Wide Web Conference (WWW 2016)*, pages 471–481, 2016.

[9] F. Chierichetti and S. Haddadan. On the complexity of sampling vertices uniformly from a graph. In *45th ICALP*, pages 149:1–149:13, 2018.

[10] C. Cooper, T. Radzik, and Y. Siantos. Estimating network parameters using random walks. *Social Netw. Analys. Mining*, 4(1):168, 2014.

[11] A. Dasgupta, R. Kumar, and T. Sarlós. On estimating the average degree. In *Proceedings of the 23rd International World Wide Web Conference (WWW 2014)*, pages 795–806. ACM, 2014.

[12] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi. The anatomy of a scientific rumor. *Scientific reports*, 3:2980, 10 2013.

[13] Matteo Dell'Amico and Yves Roudier. A measurement of mixing time in social networks. In *Proceedings of the 5th International Workshop on Security and Trust Management, Saint Malo, France*, page 72, 2009.

[14] S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes. k-core organization of complex networks. *Phys. Rev. Lett.*, 96:040601, 2006.

[15] T. Eden, S. Jain, A. Pinar, D. Ron, and C. Seshadhri. Provable and practical approximations for the degree distribution using sublinear graph samples. In *Proc. 2018 World Wide Web Conference WWW*, pages 449–458, 2018.

[16] T. Eden, D. Ron, and C. Seshadhri. On approximating the number of k-cliques in sublinear time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 722–734. ACM, 2018.

[17] T. Eden and W. Rosenbaum. On sampling edges almost uniformly. In *1st Symposium on Simplicity in Algorithms, SOSA*, pages 7:1–7:9, 2018.

[18] Talya Eden, Saleet Mossel, and Ronitt Rubinfeld. Sampling multiple edges efficiently. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 51:1–51:15, 2021.

[19] T. Eliassi-Rad, R.S. Caceres, and T. LaRock. Incompleteness in networks: Biases, skewed results, and some solutions. In *SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 3217–3218. ACM, 2019.

[20] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in Facebook: A case study of unbiased sampling of OSNs. In *Proc. IEEE INFOCOM*, 2010.

[21] D. F. Gleich. Pagerank beyond the web. *SIAM Review*, 57(3):321–363, 2015.

[22] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

[23] Oded Goldreich and Dana Ron. On testing expansion in bounded-degree graphs. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 68–75. Springer, 2011.

[24] S. Hanneke and E.P. Xing. Network completion and survey sampling. In *Proc. 12th Conf. on Artificial Intelligence and Statistics (AISTATS)*, pages 209–215, 2009.

[25] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43(4):439–561, 2006.

[26] K. Iwasaki and K. Shudo. Comparing graph sampling methods based on the number of queries. In *11th International Conference on Social Computing and Networking*, SocialCom '18, pages 1136–1143, 2018.

[27] 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. 3rd ACM Intl. Workshop on MobiArch*, HotPlanet '11, pages 11–16, 2011.

[28] L. Katzir and S. J. Hardiman. Estimating clustering coefficients and size of social networks via random walk. *ACM Trans. Web*, 9(4):19:1–19:20, 2015.

[29] L. Katzir, E. Liberty, O. Somekh, and I. A. Cosma. Estimating sizes of social networks via biased sampling. *Internet Mathematics*, 10(3-4):335–359, 2014.

[30] Liran Katzir, Clara Shikhelman, and Eylon Yogev. Interactive proofs for social graphs. In *Advances in Cryptology – CRYPTO 2020*, pages 574–601, 2020.

[31] M. Kim and J. Leskovec. The network completion problem: Inferring missing nodes and edges in networks. In *Proc. of the 11th SIAM International Conference on Data Mining, (SDM 2011)*, pages 47–58. SIAM / Omnipress, 2011.

[32] J. M. Kleinberg. Detecting a network failure. *Internet Math.*, 1(1):37–55, 2003.

[33] T. LaRock, T. Sakharov, S. Bhadra, and T. Eliassi-Rad. Reducing network incompleteness through online learning: A feasibility study. In *Proc. of the 14th International Workshop on Mining and Learning with Graphs (MLG)*. ACM, 2018.

[34] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 631–636, 2006.

[35] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, 2014.

[36] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. 11th ACM SIGKDD conference on Knowledge discovery in data mining*, pages 177–187, 2005.

[37] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2–es, 2007.

[38] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[39] R. Li, J. X. Yu, L. Qin, R. Mao, and T. Jin. On random walk based graph sampling. In *IEEE 31st Int. Conf. Data Engineering*, pages 927–938, 2015.

[40] A. Mohaisen, A. Yun, and Y. Kim. Measuring the mixing time of social graphs. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 383–389, 2010.

[41] A. Nazi, Z. Zhou, S. Thirumuruganathan, N. Zhang, and G. Das. Walk, not wait: Faster sampling over online social networks. *Proc. VLDB.*, 8(6):678–689, 2015.

[42] Giulia Preti, Gianmarco De Francisci Morales, and Matteo Riondato. Maniacs: Approximate mining of frequent subgraph patterns through sampling. In *Proc. 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1348–1358, 2021.

[43] Yi Qi, Wanyue Xu, Liwang Zhu, and Zhongzhi Zhang. Real-world networks are not always fast mixing. *The Computer Journal*, 2020.

[44] M. Rahmani, A. Beckus, A. Karimian, and G. K. Atia. Scalable and robust community detection with randomized sketching. *IEEE Transactions on Signal Processing*, 68:962–977, 2020.

[45] B. F. Ribeiro and D. F. Towsley. Estimating and sampling graphs with multidimensional random walks. In *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010*, pages 390–403. ACM, 2010.

[46] B. F. Ribeiro, P. Wang, F. Murai, and D. Towsley. Sampling directed graphs with random walks. In *Proc. IEEE INFOCOM*, pages 1692–1700, 2012.

[47] M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. In *ISWC*, pages 351–368, 2003.

[48] M. Rombach, M. Porter, J. Fowler, and P. Mucha. Core-periphery structure in networks. *SIAM Journal on Applied Mathematics*, 74(1):167–190, 2014.

[49] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proc. 29th AAAI Conf. Artificial Intelligence*, 2015.

[50] N. Salamanos, E. Voudigari, and E. J. Yannakoudakis. Deterministic graph exploration for efficient graph sampling. *Social Network Analysis and Mining*, 7(1):24, 2017.

[51] C. Seshadhri, Ali Pinar, and Tamara G. Kolda. Wedge sampling for computing clustering coefficients and triangle counts on large graphs. *Statistical Analysis and Data Mining*, 7(4):294–307, 2014.

[52] S. Soundarajan, T. Eliassi-Rad, B. Gallagher, and A. Pinar. MaxOutProbe: An algorithm for increasing the size of partially observed networks. In *Workshop on Networks in the Social and Information Sciences, 29th Conference on Neural Information Processing Systems (NIPS 2015)*, 2015.

[53] S. Soundarajan, T. Eliassi-Rad, B. Gallagher, and A. Pinar. ϵ-WGX: adaptive edge probing for enhancing incomplete networks. In *Proc. of the 2017 ACM on Web Science Conference (WebSci 2017)*, pages 161–170. ACM, 2017.

[54] L. Takac and M. Zabovsky. Data analysis in public social networks. In *International Workshop Present Day Trends of Innovations*, 2012.

[55] Jakub Tětek and Mikkel Thorup. Sampling and counting edges via vertex accesses. *CoRR*, abs/2107.03821, 2021.

[56] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

[57] Se-Young Yun and Alexandre Proutiere. Community detection via random and adaptive sampling. In *Conference on learning theory*, pages 138–175. PMLR, 2014.

[58] Kai Zhang, Qian Yu, Kai Lei, and Kuai Xu. Characterizing tweeting behaviors of sina weibo users via public data streaming. In *Web-Age Information Management*, pages 294–297. Springer, 2014.

[59] Y. Zhang, E. D. Kolaczyk, and B. D. Spencer. Estimating network degree distributions under sampling: An inverse problem, with applications to monitoring social media networks. *Annals of Applied Statistics*, 9(1):166–199, 2015.

[60] Z. Zhou, N. Zhang, Z. Gong, and G. Das. Faster random walks by rewiring online social networks on-the-fly. *ACM Trans. Database Syst.*, 40(4):26:1–26:36, 2016.

A APPENDICES

A.1 Remaining procedures of SampLayer

We provide in Figure 9 the pseudocode of two structural analysis procedures in SampLayer: Estimate-Baseline-Reachability, for computing an estimate for the baseline reachability score; and Estimate-Periphery-Size, for estimating the size of $L_{\geq 2}$.

```
Estimate-Baseline-Reachability
Input: reachabilities $r_1 \leq \ldots \leq r_m$, param. $\varepsilon > 0$. Output: Unbiased $\varepsilon$-quantile for the reachabilities.
(1) $\forall i \in [m]$ set $w_i = \frac{r_1}{r_i}$. Let $cw_i = \frac{\sum_{j \in [i]} w_j}{\sum_{j \in [|S|]} w_j}$.
(2) Return minimum $r_i$ for which $cw_i \geq \varepsilon$.
```

```
Estimate-Periphery-Size
Input: $s_1$ and $s_{\geq 2}$, numbers of samples to take from $L_1$ and $L_{\geq 2}$, respectively.
Output: $\overline{\ell}_{\geq 2}$ - the estimated size of $L_{\geq 2}$.
(1) Query $s_1$ nodes from $L_1$ uniformly at random. Denote by $S_1$ these queried nodes.
(2) $\forall v \in S_1$, compute $d^+(v) = N(v) \cap L_{\geq 2}$.
(3) Let $d^+_{avg}(S_1) = \frac{1}{|S_1|} \sum_{v \in S_1} d^+(v)$.
(4) Invoke Reach-$L_{\geq 2}$ for $s_{\geq 2}$ times and let $S_2$ denote the set of returned nodes.
(5) $\forall v \in S_2$, invoke Comp-Reachability$(v)$ to compute the reachability score $rs(v)$.
(6) $\forall v \in S_2$, compute $d^-(v) = |N(v) \cap L_1|$.
(7) Let $trs(S_2) = \sum_{v \in S_2} 1/rs(v)$ and define $d^-_{avg}(S_2) = \frac{1}{trs(S_2)} \sum_{v \in S_2} d^-(v)/rs(v)$.
(8) Return $\overline{n}_{\geq 2} = |L_1| \cdot d^+_{avg}(S_1) \ / \ d^-_{avg}(S_2)$.
```

Figure 9: Missing procedures of SampLayer.

A.2 Description of SampLayer+

In Figure 10 we present the psuedocode of the procedures in SampLayer+ whose implementation differs from that in SampLayer. There are three such procedures: for generating L_0, reaching $L_{\geq 2}$, and computing the reachability of a node. All other procedures are as in SampLayer.

SampLayer+ takes advantage of the stronger query model (which also reveals the degrees of the neighbors) in two ways. First, in the L_0-generation phase, at each round we pick a neighbor with absolute maximum degree, rather than a "perceived" maximum degree as in SampLayer. This results in SampLayer+ generally adding higher-degree nodes to L_0 compared to SampLayer.

The second modification is during the sampling phase, and involves the reachability computation. We utilize the stronger query model of SampLayer+ to reach $L_{\geq 2}$ in ways that reduces the query-cost of rejection. Here, we have a way to guarantee that the random edge entering L_2 in our reaching attempt is uniform among all edges between L_1 and L_2; in SampLayer, we do not have such a guarantee. Thus, the reachability of a component is simply the number of edges entering it from L_1 divided by the component size. This makes the reachability both easier to compute (requires less queries) and more evenly distributed than in SampLayer.

A.3 Interval Length for Mixing

In the query complexity evaluation for random walks, Section 4.1, we mention that the walks are sampled every fixed interval, where the interval length should allow for mixing. Here we detail how the interval length is judiciously chosen.

Consider a collection of k random walks $W_1, \ldots, W_k$ with the same starting point, v. How can we determine how much steps of a random walk are required to mix? One natural way to do so is by considering the t'th nodes in each random walk, $W_1(t), \ldots, W_k(t)$, for different choices of t. We would like to set the interval length to the smallest t for which $W_1(t), \ldots, W_k(t)$ are sufficiently uniform. This is a standard statistical estimation task. The most standard approach to solving it is by calculating the *empirical distance* of the observed nodes $W_1(t), \ldots, W_k(t)$ to the uniform distribution over all nodes. For all networks except for SinaWeibo, we ran $k = n$ random walks, where n is the number of nodes in the network. One can show that for random variables $X_1, \ldots, X_n \in [n]$ that are generated uniformly at random, the empirical distance to uniformity is concentrated around $1/e$. We thus picked t to be the smallest for which the empirical distance to uniformity of $W_1(t), \ldots, W_n(t)$ is no more than some small parameter ζ (on real-valued networks we picked $\zeta = 0.01$, and for forest fire $\zeta = 0.03$) away from the ideal $1/e$.

To make for a fair comparison for our algorithm, we calculated what choice of the proximity parameter ε yields the same empirical

```
Generate-$L_0$
Input: arbitrary node $v_0$, number $\ell_0$.
Output: $L_0$ of size $\ell_0$, $L_1$, $\mathcal{D}^+$.
(1) Query $v_0$ and let $L_0 = \{v_0\}$, $L_1 = N(v_0)$.
(2) Repeat $\ell_0 - 1$ times: pick $u \in L_1$ with maximum degree (break ties randomly); query $u$; remove $u$ from $L_1$; add $u$ to $L_0$; and add $N(u) \setminus L_0$ to $L_1$.
(3) Create a data structure $\mathcal{D}^+$ that allows to sample a node in $L_1$ with probability proportional to its number of neighbors in $L_1 \cup L_2$.
```

```
Comp-Reachability
Input: An (already visited) component $C$ of $G_{\geq 2}$.
Output: The reachability score of $C$.
(1) For every $v \in C$, set $rs'(v) = |N(v) \cap L_1|$.
(2) Return $rs(C) = \frac{1}{|C|} \sum_{v \in C} rs'(v)$.
```

```
Reach-$L_{\geq 2}$
Input: The data structure $\mathcal{D}^+$.
Output: A node in $L_{\geq 2}$, and its component and reachability score.
(1) While true:
    (a) Use $\mathcal{D}^+$ to sample a node $u \in L_1$ according to the distribution prescribed by $\mathcal{D}^+$.
    (b) Query $u$, compute $N(u) \cap L_0$, and pick a uniform neighbor $w$ in $N(u) \setminus L_0 = N(u) \cap (L_1 \cup L_2)$.
    (c) If $w \in L_2$, exit loop. Otherwise, repeat loop.
(2) Perform BFS in $L_{\geq 2}$ to find the component $C$ of $w$. Invoke Comp-Reachability to compute $rs(C)$.
(3) Return $w$,$C$, and the reachability $rs(C)$.
```

Figure 10: Pseudo-code for SampLayer+.

distance of ζ from $1/e$. To do so we ran a grid search over various small values of ε, where for each ε we averaged over 10 experiments of computing the empirical distance from uniformity of a set of n outputs of our algorithm (with values of the other parameters, $\ell_0, s_1, s_{\geq 2}$, as mentioned above).

For SinaWeibo, the empirical distance based evaluation is computationally infeasible, since it essentially requires $k = \tilde{\Omega}(n)$. Instead we used a more efficient estimate, based on the number of collisions in $W_1(t), \ldots, W_k(t)$. Distinguishing the uniform distribution from one that is ε-far in variation distance requires only $O(\sqrt{n}/\varepsilon^2)$ different walks [23]. Here we chose t to be the smallest for which the number of collisions among the different walks is less than three times the standard deviation of the number of collisions expected from the uniform distribution.

A.4 Size and Reachability Experiments

Reachability distribution. In our theoretical analysis, we claim that if the reachabilities of nodes in $L_{\geq 2}$ are all roughly of the same order, then the rejection step of our algorithm is not too costly. Here, we demonstrate that this assumption on the reachability distribution indeed approximately holds in practice. Specifically, Figure 11 presents the reachability distribution of the $L_{\geq 2}$-nodes in DBLP for $|L_0|$ of $30k$ (the distributions for other networks are similar). For clarity, we discarded the top 3% reachabilities, which form a thin upper tail, and only show the lower 97% here. As can be seen, indeed most reachabilities are roughly of the same order: almost all $L_{\geq 2}$-nodes in the experiment have reachability up to 1, where a majority of them are between roughly 0.05 and 0.2.

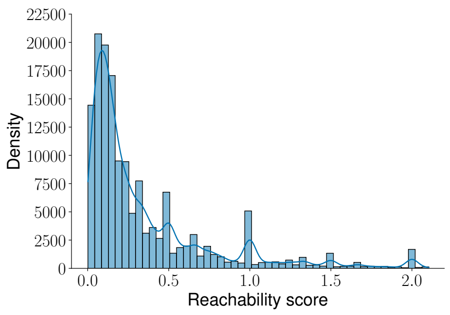


Figure 11: Reachability distribution in $L_{\geq 2}$ for DBLP.

Size estimation. As mentioned, our algorithm needs to compute an accurate size estimate for $L_{\geq 2}$ as part of its preprocessing. We next empirically demonstrate the quick convergence of our size estimate as a function of the numbers of nodes from L_1 and $L_{\geq 2}$ we visit during the preprocessing. Recall the size estimation procedure ESTIMATE-PERIPHERY-SIZE described in Section 3.1 (see also Section A.1). Here we demonstrate the quick rate of convergence of this procedure, validating our choices of the parameters s_1 and $s_{\geq 2}$.

The size of $L_{\geq 2}$ satisfies the following: $|L_{\geq 2}| = |L_1| \cdot d_1^+/d_2^-$, where d_1^+ is the average, over all nodes $v \in L_1$, of the number of neighbors of v in L_2; and d_2^- is the symmetric quantity, i.e. the (unweighted) average over all nodes in $L_{\geq 2}$ of their number of neighbors in L_1. ESTIMATE-PERIPHERY-SIZE estimates d_1^+ and d_2^- by taking s_1 samples from L_1 and $s_{\geq 2}$ sampled nodes from $L_{\geq 2}$ (using REACH-$L_{\geq 2}$, without rejection; as these are biased, we use weighted averaging). It then uses these estimates to approximate $|L_{\geq 2}|$. In the current experiment, we separately check how the chosen values of s_1 and $s_{\geq 2}$ affect the size estimate of $L_{\geq 2}$. We run two experiments, each of them five times for each of the networks; see the results for DBLP in Figure 12 as a representative example. First, we fix the value of d_2^-, and measure the error in the estimate $\bar{\ell}_{\geq 2}$ when d_1^+ is computed as the average out-degree over s_1 samples. The second experiment is similar, except that d_1^+ and d_2^- switch roles: d_1^+ is fixed to its actual value, whereas we compute an estimate of d_2^- from $s_{\geq 2}$ nodes in $L_{\geq 2}$ obtained through our algorithm (without the rejection step), where each reached node is assigned a weight that is inversely proportional to its reachability. As Figure 12 shows, it suffices to take s_1 of order a few thousands (left) and $s_{\geq 2}$ of order a few hundreds (right) to obtain a small error in the size estimation.

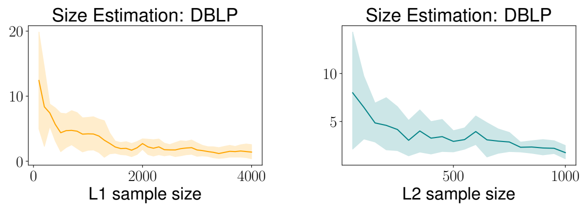


Figure 12: Error (%) of the size estimate obtained by SAMPLAYER for DBLP during our structural decomposition phase, as a function of the number s_1 of nodes queried from L_1 (left) and the number $s_{\geq 2}$ of reached nodes from L_2 (right).

A.5 Source Code

The source code for our algorithm can be found at:

`https://github.com/omribene/sampling-nodes`

For reference, we also provide the source code for the random walk algorithms to which we compared our method. See the `README.md` file for usage instructions.

A.6 System Specifications and Running Time

We ran all experiments on a 20-core CPU configuration: Intel(R) Core(TM) i9-9820X CPU @ 3.30GHz; RAM: 128GB. The most time-consuming experiments were those involving the comparison with random walks, specifically the experiments determining the correct interval size for the random walks. In SinaWeibo, our largest graph, the running time for generating 1M random walks of proper length from one starting node was about three days. For all experiments involving our algorithm, the running time was at most a few hours (and usually up to a few minutes for the smaller networks).