# Guidelines for Online Network Crawling: A Study of Data Collection Approaches and Network Properties

Katchaguy Areekijseree, Ricky Laishram and Sucheta Soundarajan
Department of Electrical Engineering and Computer Science, Syracuse University
Syracuse, NY
{kareekij,rlaishra,susounda}@syr.edu

## ABSTRACT

Over the past two decades, online social networks have attracted a great deal of attention from researchers. However, before one can gain insight into the properties or structure of a network, one must first collect appropriate data. Data collection poses several challenges, such as API or bandwidth limits, which require the data collector to carefully consider which queries to make. Many online network crawling methods have been proposed, but it is not always clear which method should be used for a given network. In this paper, we perform a detailed, hypothesis-driven analysis of several online crawling algorithms, ranging from classical crawling methods to modern, state-of-the-art algorithms, with respect to the task of collecting as much data (nodes or edges) as possible given a fixed query budget. We show that the performance of these methods depends strongly on the network structure. We identify three relevant network characteristics: community separation, average community size, and average node degree. We present experiments on both real and synthetic networks, and provide guidelines to researchers regarding selection of an appropriate sampling method.

## KEYWORDS

Experiments; Online Sampling Algorithm; Network Crawling; Network Sampling; Complex Networks.

**ACM Reference Format:**
Katchaguy Areekijseree, Ricky Laishram and Sucheta Soundarajan. 2018. Guidelines for Online Network Crawling: A Study of Data Collection Approaches and Network Properties. In *WebSci '18: 10th ACM Conference on Web Science, May 27–30, 2018, Amsterdam, Netherlands.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3201064.3201066

## 1 INTRODUCTION

The study of complex networks has become a critical aspect of research in a wide range of fields. Often, network data is collected from online sources, including online social networking sites (OSNs) such as Facebook or Twitter. Such sites may provide APIs, which are a convenient channel for accessing data. However, the data collection process can require a significant amount of time. Thus, when collecting data, efficiency is extremely important. In this paper, we consider the problem of **network sampling through crawling**, or **online sampling**, in which the only way to obtain more information about the network is to query observed nodes for their neighbors, and thus expand the observed network. In this paper, we use *network sampling* and *network crawling* interchangeably.

In network crawling, there are a number of important goals, such as finding samples that are unbiased with respect to some property, locating 'important' nodes, or finding a sample that preserves information flow patterns. In this work, however, we focus on the efficiency of the crawling algorithm itself (i.e., How quickly nodes and edges can be discovered through the crawling process). Our first considered goal is maximizing the total number of nodes observed, which we refer to as the *node coverage* goal. Second goal is maximizing number of edges observed, which we refer to as the *edge coverage* goal. These two goals have been closely tied to other application-specific crawling goals, including crawling for census-type applications (i.e., collecting node or edge attribute information) [10] and crawling to preserve community structure [15].

While network crawling is a critically important step in a network analysis tasks, it is often difficult for users to select a crawling technique. The literature contains a large number of crawling algorithms, and it is rarely clear which method is best to use for a given network. Existing work in the literature has typically attempted to determine which crawling method is the 'best'. In this paper, rather than arguing for usage of one single crawling algorithm, our goal is to investigate the relationship between network structure and crawler performance. More specifically, we investigate those network structural properties that govern the ability of a crawler being able to move between 'regions' of a graph, and demonstrate that these features have a strong effect on the performance of various crawling methods.

However, the knowledge of how structural properties affect the performance of a crawler might not be immediately helpful, because one does not know ahead of time what the properties of network are! To address this, we consider broad categories of networks, and show that for networks in the same category (which exhibit similar structural properties), the crawling methods tend to have similar performance relative to one another. Based on these properties, we provide general guidelines on selecting a crawling method for a particular network type.

Our work has several novel contributions:

(1) To the best of our knowledge, this is the first work to systematically examine the effect that network structural properties have on the performance of network crawling methods.
(2) We provide an extensive, scientific analysis of the relationship between network structural properties and the performance of popular crawling algorithms. Unlike existing


*WebSci '18, May 27–30, 2018, Amsterdam, Netherlands*

ACM ISBN 978-1-4503-5563-6/18/05…$15.00
https://doi.org/10.1145/3201064.3201066

works on sampling algorithms, which typically perform a high-level comparison of crawling methods, our goal is to understand the relationship between network structure and crawler performance.

(3) We provide a framework for classifying network crawling algorithms, based on how their performance changes as network structure changes, and categorize networks according to their structural properties.

The rest of this paper is organized as follows. In the following Section 2, we give an overview of the existing works in this area. In Section 3, we discuss the problem, preliminaries, details of the network crawling methods and the network structural properties of interest. We describe our experiments and discuss about the results in Section 4. We present conclusion in Section 5 and future works in 6.

## 2 RELATED WORK

Work on network sampling can be separated into two main scenarios: *down-sampling* and *sampling through crawling*. When down-sampling, one possesses the complete network dataset, and wishes to scale it down to some desired size (perhaps because the entire dataset is too large to fit into the memory or would take too long to analyze). The good sample network will maintain the relevant properties and characteristics of the original network.

In the crawling scenario, one starts with *no information* about the network other than the knowledge of a single node. One can obtain addition information by iteratively performing queries on observed nodes (e.g., through an API) to collect the information about the unobserved network. In this way, the observed sample is expanded from the single initially observed node. While both of these problems are often broadly referred to as 'sampling', they require intrinsically different approaches. In this work, we consider algorithms that are used in the *crawling scenario*.

Due to the vast amount of literature in this area, we cannot provide a complete discussion of the field. For a more comprehensive discussion, we refer the reader to the excellent survey in [2].

**Algorithms for Network Crawling:** Network crawling has been used extensively to collect data from the Internet and other large networks like the WWW and online social network platforms. One of the largest online social network studies is presented by Mislove, et al. They study four online social network sites (Orkut, Youtube, Live Journal and Flickr) and collect the data by using a BFS crawler [16]. A similar study on other online social networking sites, such as CyWorld and MySpace, is presented by Ahn, et al. in [3]. However, the sampled networks produced by a BFS crawler contains bias. Kurant, et al. suggest a solution for correcting the bias from a BFS crawler [9]. Similarly, Gjoka, et al. propose another approach baseed on Metropolis-Hastings-based Random Walk for crawling an unbiased sample and use it for collecting Facebook network [7].

There has also been a great deal of interest on network crawling for specific applications. Salehi, et al. introduce PageRank-Sampling [18] to preserve community structure. Avrachenkov, et al. present a greedy crawling method, called Maximum Observed Degree (MOD), for maximizing number of nodes in the sample in [4] by querying the node with the highest observed degree in each step. A similar approach is used by the OPIC algorithm, which considers PageRank as a measurement of importance [1]. Results show that both MOD and OPIC significantly outperform other methods.

**Analysis of Sampling Algorithms:** There has also been a significant amount of work comparing the performance of sampling algorithms. In [12], Leskovec and Faloutsos study the characteristics of different down-sampling methods and attempt to find out the best method that produces the smallest bias for all the defined network properties, arguing that Random Walk is the best at preserving network properties. Similarly, Kurant, et al. analyze BFS crawler [8]. The experiment results show that the crawler is biased towards high degree nodes.

Ye, et al. present a large-scale empirical study that compares crawling methods on the basis of performance, sensitivity, and bias, on the tasks of node and edge coverage [19]. Ahmed, et al. provide a detailed framework for classifying sampling algorithms and examine the performance of various algorithms at the task of preserving graph statistics [2]. While these works perform an important evaluation of existing algorithms, they typically conduct a high-level comparison of sampling methods.

In contrast, the purpose of our work is to perform a detailed analysis of *the effects of network structure on the performance of crawling algorithms*, rather than evaluating the performance of algorithms, we seek to answer *how* and *why* algorithms succeed or fail. Thus, we can give some insights and guildlines on how to pick an appropriate method when data collection need to be performed.

**Community Structure:** We draw heavily from the literature on community structure in networks. To characterize the strength of communities in networks, we use the popular modularity metric and the corresponding Louvain method for optimizing modularity [5]. Mixing between communities is an important part of our analysis: for example, Leskovec, et al. suggest that communities are well defined and distinct if they are small, but that large communities tends to mix with one another [13].

Our work differs from existing work in important ways: (1) We present a comprehensive analysis of sampling algorithms for the *crawling scenario* (in contrast to most existing work, which is for down-sampling). (2) Our primary goal is not to determine which existing algorithms are best, but rather to gain insight into the interplay between network structure and crawler performance, and thus understand *why* certain algorithms perform better or worse. (3) We provide guidelines for selecting the appropriate sampling approach if the network category is known.

## 3 NETWORK SAMPLING THROUGH DATA CRAWLING

Suppose that we are collecting data from an online social network through its API, and we only have 24 hours to collect data. The process starts with one known user account. As our first step, we must query that user, and the server responds with a list of neighbors. Then, one of these returned users is selected for the next query, and so on. This process is repeated until 24 hours have passed. The problem is to decide which node to select for each query such that the total number of nodes or edges observed is maximized.

## 3.1 Problem Definition

Let $G = (V, E)$ be a static unobserved, undirected network. We are given a starting node $n_s \in V$ and a total query budget $b$. To collect data, we may perform a query on an previously-observed node. In this work, we assume that in response to a query, we receive all neighbors of the queried node. In each step, the crawler queries an observed-but-not-queried node (i.e., a node that was observed as a neighbor of a previously-queried node, but has itself not yet been queried). This process is repeated until $b$ queries have been made. The output is a sample graph $S = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, containing all nodes and edges observed. Table 1 shows the notation used in the paper.

**Table 1: Notations and Definitions.**

| Notation | Definition |
|---|---|
| $n_s$ | A seed node from which the crawler begins. |
| $V_o^{'}$ | The set of nodes that have been observed but not queried ('*open*' nodes). |
| $V_c^{'}$ | The set of nodes that have been observed and queried ('*closed*' nodes). |
| $V^{'}$ | The set of all observed nodes, $V^{'} = V_o^{'} \cup V_c^{'}$. |
| $E^{'}$ | The set of all observed edges. |
| $d_v$ | The degree (number of neighbors) of node v. |

We consider two different sampling goals:

**Goal 1: Node Coverage** - Collect a sample graph $S = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$ so that the number of nodes in $V'$ is maximized.

**Goal 2: Edge Coverage** - Collect a sample graph $S = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$ so that the number of edges in $E'$ is maximized.

We selected these goals because they are closely related to many other application-specific goals: for example, the community-based sampling techniques in [15] use the node coverage goal to identify a sample that captures the community structure of the network. The authors in [14] also use this goal to identify a set of most influential nodes across several centrality measures. We do not discount the importance of crawling for other applications (such as obtaining unbiased samples), but these are fundamentally different problems.

Note that a sampling algorithm that is successful with respect to one of these goals may not be successful with respect to the other. For example, suppose an open node (i.e., a node that has been observed through other nodes but not yet queried) has many unobserved edges adjacent to it, but these edges lead to other open nodes. Querying this node would lead to a large increase in the number of observed edges, but not the number of observed nodes.

## 3.2 Online Crawling Methods

In our study, we compare nine popular crawling methods. As mentioned in the previous section, the crawler expands the sample by selecting for query a node that had been previously observed. The details of each algorithm are as follows:

**Random Crawling (Rand):** In each iteration, the crawler randomly selects a node from $V_o^{'}$ for the next query.

**Random Walk (RW):** In each iteration, the crawler transitions to a random neighbor of the node that was just queried. Nodes can be visited multiple times but crawler only performs a new query on node $v \in V_o^{'}$ if it had not been previously queried. The results of `Random Walk` came out on top in [12].

**Breadth-First Search (BFS):** The crawler selects the node that has been in the list of unqueried nodes the longest (i.e., First-In, First-Out). `BFS` is widely used for network sampling because of its simplicity. In addition, the obtained network gives a complete view (all nodes and edges) of a particular area in the graph, which may be useful for network analysis. An example of one of the largest network analysis using BFS is presented in [16].

**Snowball Sampling (SB):** The crawler acts similarly to `BFS`, except that when a node is queried, only $p$ fraction of its neighbors are added to the queue for future queries. Here, we set $p$ to 0.5. Experimental results in [3] show that this approach is capable of discovering hub nodes (i.e., nodes with high degree), which helps the crawler in expanding to unexplored areas in the graph.

**Depth-first Search (DFS):** The crawler acts similarly to `BFS`, except that a node is selected in LIFO fashion (i.e., Last-In, First-Out).

**Maximum Observed Degree (MOD):** This is a greedy algorithm based on the intuition that a node with high observed degree likely has high unobserved degree. For each query, the crawler selects an open node with the highest observed degree [4]. Experimental results in [4] demonstrated that `MOD` substantially outperforms other algorithms at the node coverage task.

**Maximum Observed PageRank (PR):** The crawler acts similarly to `MOD`. The PageRank score of every observed node is calculated when new nodes (or new edges) are added to the sample graph. The node with the highest observed PageRank score is selected for the next query. As argued in [18], this algorithm can capture the community structure of the network.

**Online Page Importance Computation (OPIC):** This is an online algorithm that aims to calculate the nodes importance score without recalculating from the whole sample. The algorithm only updates the scores of the most recently queried node and its neighbors. Initially, each observed node is given an equal amount of *"cash"*. In each step, the crawler selects the node with the most cash and its cash is distributed evenly to its neighbors. Results in [1] show that `OPIC` can compute the importance of nodes as in standard methods, but it is faster.

**Volatile Multi-armed Bandit (VMAB):** VMAB is a reinforcement learning algorithm that balances *exploration* and *exploitation*. This approach is used for finding target nodes on a network in [6]. The UCB1 algorithm is used for selecting an arm. Each arm represents a set of unqueried nodes with equivalent structural properties (we use 'common neighbors', as described [6], for the implementation in our experiment). In [6], `VMAB` modifies UCB1 to handle non-stationary bandits, because arms can appear, disappear or merge (because nodes are added to the sample which affect the change of structural properties). In each iteration, the crawler selects the arm that maximizes $\bar{W}_i + C_p \cdot \sqrt{\frac{2 \cdot ln(n - z_i)}{T_i(n)}}$ and randomly picks a node from this arm to query.

### 3.3 The Effects of Network Structure on Algorithm Performance

The purpose of our study is to examine the effects of network structural properties on the relative performance of network crawlers. It is known that crawler performance may vary substantially by network [19]. This variance must be due to differences in structural properties of the underlying networks: but what are those properties, and how do they affect the comparative performance of crawling methods?

*3.3.1 Structural Properties of Interest.* Our hypothesis is that *the performance of crawling methods is mostly affected by the ease with which a crawler can move between regions of the graph.* If it is difficult to move between regions of the graph, and the crawler gets stuck in one general area, then it will eventually start seeing the same nodes and edges over and over again. We thus consider three structural features:[1]

**Community Separation:** We consider a community to be a subgraph with high internal density and relatively few edges to the rest of the graph. To determine how well-separated communities are, we use modularity $Q$, as defined by Newman [17], which is defined as follows:

$$Q = \frac{1}{2m} \sum_{vm} \left[ A_{vw} - \frac{d_v d_w}{2m} \right] \delta(c_v, c_w),$$

where $A$, $m$ and $d_i$ are the adjacency matrix, total edges, and degree of node $i$, respectively. $\delta(c_v, c_w)$ is a delta function which returns one when node $v$ and $w$ are in the same community. Otherwise, it returns zero. The higher the modularity, the better the separation between communities, and so a crawler is more likely to get trapped in a region. We find communities using the Louvain method [5].

**Average Degree:** If the average node degree is high relative to community size, then a node is more likely to have neighbors outside of its own community, making it easier for a crawler to move between regions. It is defined as

$$d_{avg} = \frac{\sum_{v \in V} d_v}{m}.$$

**Average Community Size:** As described above, community size is relevant in conjunction with average degree.

$$CS_{avg} = \frac{\sum_{c_i \in C} |c_i|}{|C|},$$

where $C$ is a set of communities, $c_i$ is the set of nodes in community $i$ and $|\cdot|$ refers to a cardinality of a set.

*3.3.2 Properties of Real Networks.* As we will see, the properties described above have a large effect on crawler performance: but if one begins without any knowledge of the network, how can one take advantage of our results to select an algorithm? As it is well-known, networks of the same type tend to have similar structural properties. For example, social networks tend to have more near-cliques than citation networks, which are more likely to have many long chain-like structures. Unfortunately, while it is not possible for a single algorithm to be the best on every network, we are able to produce general guidelines for selecting a crawling algorithm for a particular type of network. These results are presented in sections 4.1.2 and 4.2.

[1] We explored other structural properties, such as clustering coefficient, but these three emerged as having the greatest effect on crawler performance.

## 4 EXPERIMENTAL STUDIES

We first perform a series of controlled experiments on synthetic networks, in which we methodically modify structural properties of the network and observe the effect of these variations on the performance of the crawling algorithms. We then validate our observations with real world networks. Using these results, we group the crawling algorithms into three distinct classes. We additionally group the network datasets into categories based on domain. Within each category of network, crawling algorithms show consistent performance, allowing a user to select the method that is best for a particular category.

### 4.1 Effects of Network Properties

As mentioned in the earlier section, we consider three network structural properties: the community separation , the average degree, and the average community size.

*4.1.1 Synthetic networks.* We use the LFR network model [11] to generate synthetic networks. This model is usually used as a benchmark networks for evaluating different community detection algorithms. The model allows us to control average node degree, community size, and community mixing. We set each network to have 5000 nodes and a maximum node degree of 300. We vary the community mixing parameter $\mu$ (the fraction of edges crossing between communities, ranging from 0 to 1), average node degree, and community size.

Note that community mixing $\mu$ and modularity $Q$ are inversely related, meaning that networks with high community mixing will have low modularity. Here, we consider values of $\mu$ from 0.1 to 0.9, average degrees from 15 to 200, and average community sizes from 100 to 2500. Higher values of $\mu$ indicate fuzzier community borders.

For each set of parameters, we generate 10 networks. We generate multiple graphs with same parameters instead of running multiple experiments on a single generated graph, because we want to reduce the error or bias that might come from the generated graphs. We consider budgets of up to 1000 queries, representing sampling up to 20% of the nodes in the network.

Based on our experimental results, we are able to categorize the nine crawling algorithms into three groups (G1 - G3). The algorithms in each groups are as follows:

- G1: **Node Importance-based algorithms:** `Maximum observed degree`, `OPIC` and `Maximum observed PageRank`
- G2: **Random Walk**
- G3: **Graph Traversal-based algorithms:** `BFS`, `DFS`, `SB`, `Random`, and `VMAB`

A summary of how structural properties affect each group is shown in Table 2. We plot results for methods from each group in Figures 2-4. For brevity, we show only results for the best method in each group: `MOD`, `Random Walk` and `BFS`. Results are as follows:

***Node Coverage***: Figure 2 depicts results for the node coverage task as average degree and average community size are varied, with community mixing fixed at 0.1 (i.e., few connections between

**Table 2: Categorization and summary of the performances of sampling algorithms.**

| Coverage | Property | G1: Node Importance-Based | G2: Random Walk | G3: Graph Traversal-Based |
|---|---|---|---|---|
| Node | **Community Separation** | Excellent performance when community overlap is high (i.e. low $Q$ or high $\mu$). | Stable | Stable |
| | **Average community size** | Strong performance when communities are large if $\mu$ is low. Community size does not matter if $\mu$ is high. | | |
| | **Average degree** | Strong performance when average degree is extremely low (<10) even if $\mu$ is low. Otherwise, stable | | Performance improvement when average degree increases. |
| Edge | **Community Separation** | Excellent performance when community overlap is high (i.e. low $Q$ or high $\mu$). | Stable | Stable |
| | **Average community size** | Strong performance when communities are large if $\mu$ is low. Community size does not matter if $\mu$ is high. | | |
| | **Average degree** | Strong performance when $d_{avg}$ is low (<10) even if $\mu$ is low. Otherwise, performance drops when $d_{avg}$ increases. | Performance drops when $d_{avg}$ increases. | |
| **Best Method in Group** | | **MOD** | **RW** | **BFS** |

communities). Plots further to the right have larger average community sizes ($CS_{avg}$ ranges from 100 to 2500), and plots higher on the y-axis have higher average degrees ($d_{avg}$ ranging from 7 to 100). Within each plot, the x-axis shows the fraction of nodes queried, and the y-axis shows the fraction of nodes observed in the sample. Figure 1 illustrates the results when community mixing is varied, and average degree and average community size are fixed at 15 and 300, respectively. Figure 3 shows results when average degree is varied, when average community size is fixed at 300 and community mixing at 0.6. For brevity, we cannot show results for all parameter settings, but the depicted results are representative of the full set of results. We make several important observations from this set of plots:

**G1 - Node Importance-based methods:** These methods greedily pick a node with high centrality (degree or PageRank), and tend to behave similarly. This is not surprising, as there is a high correlation between the various centrality measures. The performances of methods in G1 significantly improve as the size of the community is increased. If community mixing is low (fewer connections between communities), and communities are small, G1 methods perform poorly, as they tend to get 'stuck' in a region: if there are few connections between communities, these methods have difficulty transitioning to new communities. They thus exhibit diminishing marginal returns: while no method will query the same node multiple times, they tend to query nodes with similar neighborhoods, resulting in redundant information. One interesting observation is that when the average degree is extremely low (last row on Figure 2), G1 methods perform worse than both G2 and G3 methods for low query budgets, but the performance rapidly increases and outperforms the other methods when budget increases. We observed the same behavior for every generated network with the same parameter setting that has average degree less than 10.

As illustrated in Figure 1, G1 methods perform better when $\mu$ increases. We can see similar results as shown in Figure 3. When community mixing is high (there are many connections between communities), G1 methods are consistently the best, as a node outside of the crawler's current community may have a high observed degree, so the crawler can escape easily.

**G2 - Random Walk:** As we can clearly see from Figure 2 and 3, the Random Walk crawler is a very stable algorithm. Its performance seems to be unaffected by all considered properties. Since

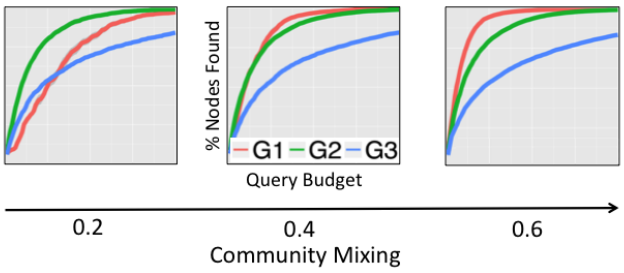


**Figure 1: (Node coverage) Results on synthetic networks with different values of community mixing $\mu$ when average degree is fixed at 15 and average community size is fixed at 300. For the individual plots, the x-axes show query budgets (0% to 20% of total nodes). The y-axes show the percent of node observed (0% to 100%). G1 methods improve as $\mu$ increases, while other methods are stable.**

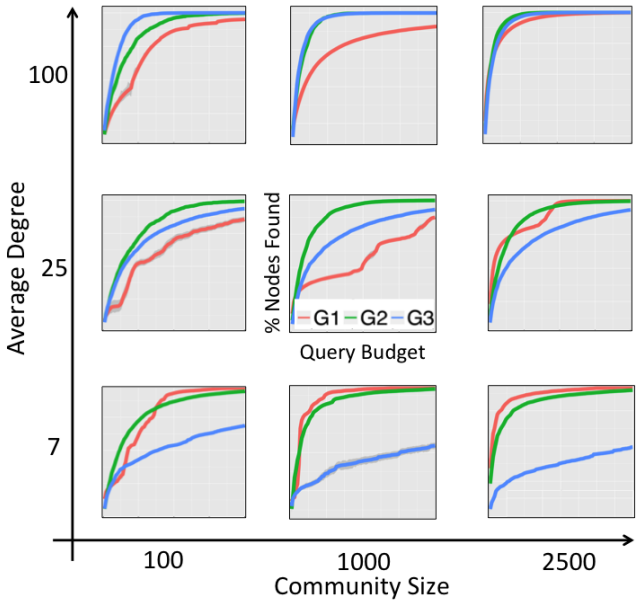


**Figure 2: (Node coverage) Results on synthetic networks with different values of $d_{avg}$ and $CS_{avg}$ when community mixing $\mu$ is fixed at 0.1. Plots to the right (along x-axis) have higher average community sizes, while plots near the top (along y-axis) have higher average degree. For the individual plots, the x-axes show query budgets ranging from 0% to 20% of total nodes in the network. The y-axes show the node coverage (measured in % of total nodes observed), ranging from 0% to 100%. The performance of the G2 random walk method is stable. G1 methods improve when $CS_{avg}$ increases, while G3 methods improve when $d_{avg}$ increases.**

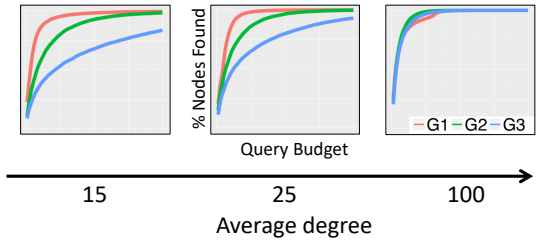


**Figure 3: (Node coverage) Results on synthetic networks with different values of average degree, when community mixing fixed at 0.6 and community size is 300. For the individual plots, the x-axes show query budgets (0% to 20% of total nodes). The y-axes show the percent of node observed (0% to 100%). G1 outperforms the others methods. The performance of G3 methods improve as average degree and average community size increases.**

the Random Walk crawler selects next node randomly from the last queried node's neighbors, the crawler can escape from the current region easily, while leaves some partial region unobserved. However, the crawler has a freedom to move back and forth, partially explored region, and them discover more nodes of this region in later steps.

**G3 - Graph Traversal-based methods:** These methods are not meaningfully affected by community size. These methods expand the frontier of the sample uniformly, and so can easily move between graph regions. However, the choice of next queried node depends on when that node was put into the query queue. It is likely that nodes with small and medium degree will be queried and the number of new nodes added to sample will be low. The performance of these methods improves as the average degree increases (moving up the y-axis in Figure 3), because they can quickly reach and escape the boundaries of a community. For large average degree, these methods outperform `Random Walk` sampling. Note that, `VMAB` is put into this group because of its performance. `VMAB` often performs poorly at the task of node coverage because new arms appear frequently, resulting in nearly random query choices.

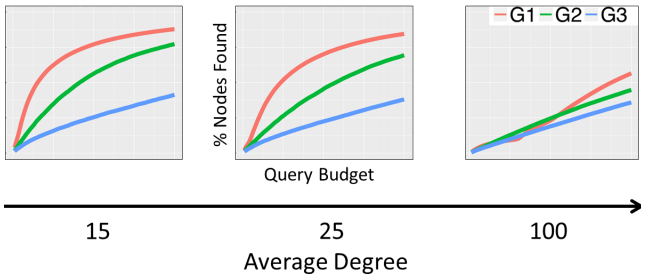


**Figure 4: (Edge coverage) Results on synthetic networks with different values of average degree, when community mixing fixed at 0.6 and community size is 1000. For the individual plots, the x-axes show query budgets (0% to 20% of total nodes). The y-axes show the percent of edge observed (0% to 100%). The performance of G3 are stable, while G1 and G2 show a decrease in performance.**

***Edge Coverage Task***: Figure 5 is interpreted similarly to Figure 2, showing how the performances of G1 - G3 change as average degree and community sizes vary, with fixed low community mixing ($\mu$ = 0.1). When varying $\mu$, performance is similar to the node coverage task, as shown in Figure 1, and due to space constraints, we do not include it. As before, these limited results are representative of the full set of results. The key observations are as follows:

**G1 - Node Importance-based methods:** These methods are the best algorithms if 1) community mixing $\mu$ is high or 2) communities are large, even if community mixing $\mu$ is low (along the x-axis in Figure 5, the performance increases). As in the node coverage task, having fuzzy borders between communities leads to improved performance by G1 methods. However, if communities are very large, then even if a G1 method gets 'stuck' in a single community, it is still able to discover many edges (here, 'large' is measured with respect to the sample budget). G1 methods also show the best performance when average degree is extremely low. However, the performance degrades when average degree increases as shown in Figure 4.

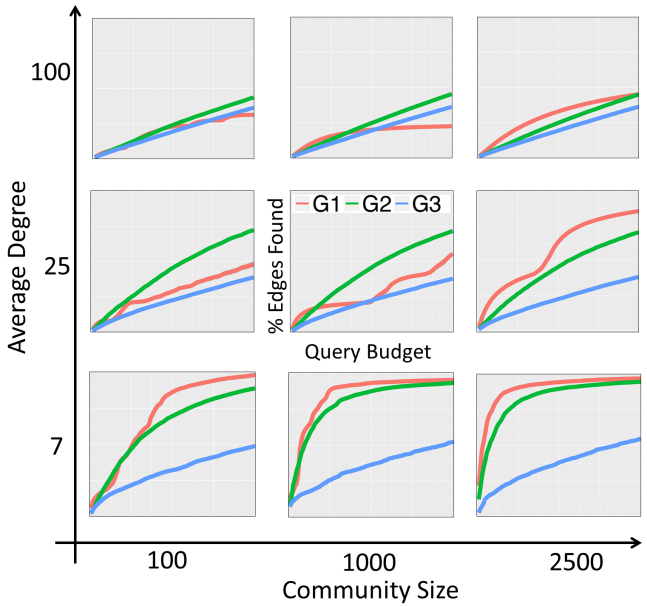


**Figure 5: (Edge coverage) Results on synthetic networks with different values of $d_{avg}$ and $CS_{avg}$ when community mixing $\mu$ is fixed at 0.1. Plots to the right have higher average community sizes, while plots near the top have higher average degree. For the individual plots, the x-axes show query budgets ranging from 0% to 20% of total nodes in the network. The y-axes show the edge coverage (measured in % of total edges observed), ranging from 0% to 100%. The G3 methods are stable throughout. The G2 method is not affected by average community size, but decreases as average degree increases. G1 algorithms improve as community size increases, but worsen as average degree increases.**

**G2 - Random Walk:** The performance seems to be very stable when community size increases regardless of whether community mixing $\mu$ is high or low.

**G3 - Graph Traversal-based methods:** These methods are stable as parameters vary, but these methods generally perform worse than `Random Walk` and methods in G1.

*4.1.2 Real-World Networks:* The previous experiments demonstrate that a major factor in the performance of each method is the ability to transition between different regions of the graph. Here, we test to see whether this conclusion holds on real graphs.

We perform three sets of controlled experiments. In each set, we locate two pairs of networks (Pair 'A' and 'B'), each containing networks that are similar with respect to two of the three structural features, but are very different with respect to the third. For example, *Wiki-Vote* and *Twitter* networks have average degree around 30 and average community size around 1100, but differ in modularity (0.42 vs. 0.81). Statistics for the selected network pairs are listed in Table 3 (with the feature being varied shown in bold).[2] In order to find communities, we use the modularity $Q$ of communities found by the *Louvain method* [5] and use it as a proxy for community mixing. As mentioned in the previous section, modularity and community mixing are inversely related ($\Uparrow Q \approx \Downarrow \mu$). Because networks are of different sizes and structures, rather than setting a fixed number of queries, we set the query budget to be 5% of the number of nodes. We perform 10 trials for each network and sampling method.

[2]Network datasets from http://networkrepository.com.

**Table 3: Network statistics of realworld networks used in the controlled experiments.**

| Test Prop. | Pair | Network | $d_{avg}$ | $CS_{svg}$ | Q |
|---|---|---|---|---|---|
| $Q$ | A | Wiki-Vote | 28.51 | 1,177.67 | **0.42** |
| | | Twitter | 33.01 | 1,129.25 | **0.81** |
| | B | Brightkite | 7.51 | 274.10 | **0.68** |
| | | MathSciNet | 4.93 | 594.09 | **0.80** |
| $CS_{avg}$ | A | Shipsec1 | 24.36 | **4,117.50** | 0.89 |
| | | Shipsec5 | 24.61 | **5,252.15** | 0.90 |
| | B | Github | 7.25 | **83.68** | 0.43 |
| | | P2P-gnutella | 4.73 | **1,276.76** | 0.50 |
| $d_{avg}$ | A | P2P-gnutella | **4.73** | 1,276.76 | 0.50 |
| | | Bingham | **72.57** | 1,250.13 | 0.45 |
| | B | Amazon | **2.74** | 272.44 | 0.99 |
| | | UK-2005 | **181.19** | 157.13 | 1.00 |

For each pair of networks, we refer to one of the networks as the *'High'-valued network (H)* and another as the *'Low'-valued network (L)*. These designations are relative to the other network in the pair (e.g., the network with higher test value is labeled 'H').

Because `Random Walk` is the most stable method (i.e., shows consistent performance regardless of the structural properties we consider), we use it as a reference point. We show the number of nodes or edges discovered by each sampling method as a *percentage improvement above (or below)* the number of nodes and edges discovered by `Random Walk`. Results are shown in Table 4. Each cell contains $\begin{bmatrix} x \\ y \end{bmatrix}$, where $x$ is the percentage improvement of the algorithm performs on the 'Low'-valued network and $y$ is the percentage improvement of the algorithm performs on the 'High'-valued network compare to `Random Walk` performance. An arrow indicates how the performance changes when the value of test properties changes from Low to High value. A symbol '⇑' represents the performance improves when the test property increases and '⇓' shows the performance degrades when test property increases.

First, we consider pairs with similar average degree and average community size, but different values of modularity $Q$ (0.42 vs. 0.81, and 0.68 vs. 0.8). As expected, G1 methods perform extremely well when modularity is low in pair A (⇓ indicates the G1 performance drops when modularity increases) for both node coverage and edge coverage tasks. On the other hand, G1 methods perform very well on both networks in pair B, because both networks have extremely low average degree ($d_{avg} < 10$). The performance of G3 methods are worse than `Random walk`, as we expected.

**Table 4: Experimental results of the controlled experiments. An arrow indicates how the performance changes when test property changes from Low to High (⇑: improve, ⇓: degrade). In $\begin{bmatrix} x \\ y \end{bmatrix}$, $x$ and $y$ indicate the percentage improvement of Low- and High- valued networks, respectively ('+': outperform RW, '-': underperform RW).**

| Test Prop. | Pair | Node Coverage | | Edge Coverage | |
|---|---|---|---|---|---|
| | | % Imprv. G1 vs. RW | % Imprv. G3 vs. RW | % Imprv. G1 vs. RW | % Imprv. G3 vs. RW |
| $Q$ | A | ⇓ $\begin{bmatrix} 7.62\% \\ -5.24\% \end{bmatrix}$ | ⇑ $\begin{bmatrix} -22.44\% \\ -13.97\% \end{bmatrix}$ | ⇓ $\begin{bmatrix} 21.12\% \\ -2.90\% \end{bmatrix}$ | ⇑ $\begin{bmatrix} -58.13\% \\ -49.33\% \end{bmatrix}$ |
| | B | ⇑ $\begin{bmatrix} 12.47\% \\ 19.81\% \end{bmatrix}$ | ⇑ $\begin{bmatrix} -28.27\% \\ -17.64\% \end{bmatrix}$ | ⇑ $\begin{bmatrix} 31.85\% \\ 53.49\% \end{bmatrix}$ | ⇑ $\begin{bmatrix} -47.22\% \\ -23.10\% \end{bmatrix}$ |
| $CS_{avg}$ | A | ⇑ $\begin{bmatrix} -71.52\% \\ -70.32\% \end{bmatrix}$ | ⇓ $\begin{bmatrix} -20.53\% \\ -27.68\% \end{bmatrix}$ | ⇑ $\begin{bmatrix} -19.65\% \\ -10.48\% \end{bmatrix}$ | ⇓ $\begin{bmatrix} -5.34\% \\ -6.38\% \end{bmatrix}$ |
| | B | ⇑ $\begin{bmatrix} 10.14\% \\ 15.67\% \end{bmatrix}$ | ⇑ $\begin{bmatrix} -35.21\% \\ -15.18\% \end{bmatrix}$ | ⇑ $\begin{bmatrix} 20.01\% \\ 34.68\% \end{bmatrix}$ | ⇑ $\begin{bmatrix} -58.44\% \\ -19.39\% \end{bmatrix}$ |
| $d_{avg}$ | A | ⇓ $\begin{bmatrix} 10.14\% \\ -14.38\% \end{bmatrix}$ | ⇑ $\begin{bmatrix} -15.18\% \\ -0.87\% \end{bmatrix}$ | ⇓ $\begin{bmatrix} 34.68\% \\ -3.19\% \end{bmatrix}$ | ⇓ $\begin{bmatrix} -19.39\% \\ -27.72\% \end{bmatrix}$ |
| | B | ⇑ $\begin{bmatrix} -0.40\% \\ 6.25\% \end{bmatrix}$ | ⇑ $\begin{bmatrix} 2.09\% \\ 334.34\% \end{bmatrix}$ | ⇓ $\begin{bmatrix} -0.48\% \\ -1.42\% \end{bmatrix}$ | ⇑ $\begin{bmatrix} 1.53\% \\ 82.61\% \end{bmatrix}$ |

Next, we consider networks with similar modularity and average degree, but different average community sizes (4118 vs. 5252 and 84 vs. 1277). Our earlier experiment predicted that G1 methods will perform better on networks with larger community sizes. As expected, G1 performance improves for networks with larger communities for both pairs.

Finally, we consider networks with similar modularity and average community size, but different average degrees (5 vs. 73 and 3 vs. 181). As predicted, G3 methods perform better on networks with higher average degree in both pairs. However, we see less consistent results for edge coverage. Overall, the experiments on real networks bear out our results on synthetic networks.

## 4.2 Network Types

In a real application, the network properties are not known until the sample is generated: so how can one select an appropriate crawling method? Here, we analyze how well the algorithms perform by network type (web, collaboration, technology, scientific, Facebook, recommendation and other OSNs)[3]. The statistics of all networks are listed in Table 5. Again, we set the maximum query budget to be 5 percent of total nodes and perform 10 trials for each method. We depict the mean and standard deviation of the percentage of nodes and edges found in Table 6.

We make the following observations:

(1) G1 or G2 methods are almost always the best, regardless of network type or coverage task.
(2) On collaboration, technological, recommendation, web and online social networks, G1 methods perform the best, while on Facebook and scientific networks, the G2 method is the best.
(3) Methods in G3 seem not to be a good choice when considering these two goals on all types of network.

[3]Network datasets from http://networkrepository.com.

**Table 5: Categories of the realworld networks and their structural characteristics.**

| Type | Network | $d_{avg}$ | $CS_{avg}$ | $Q$ | Properties |
|---|---|---|---|---|---|
| **Collab.** | Citeseer | 7.16 | 988.35 | 0.90 | Low degree, medium-sized and clear communities |
| | Dblp-2010 | 6.33 | 739.91 | 0.86 | |
| | Dblp-2012 | 6.62 | 1248.35 | 0.82 | |
| | MathSciNet | 4.93 | 594.09 | 0.80 | |
| **Recmnd.** | Amazon | 2.74 | 272.44 | 0.99 | Low degree, small and clear communities |
| | Github | 7.25 | 83.68 | 0.43 | |
| **FB** | OR | 25.77 | 1074.44 | 0.63 | High degree, large and clear communities |
| | Penn94 | 65.59 | 2186.11 | 0.49 | |
| | Wosn-friends | 25.77 | 856.65 | 0.63 | |
| **Tech.** | P2P-gnutella | 4.73 | 1276.76 | 0.50 | Low degree, large and clear communities |
| | RL-caida | 6.37 | 856.12 | 0.86 | |
| **Web.** | Arabic-2005 | 21.36 | 115.86 | 1.00 | High degree, medium -sized and fuzzy communities |
| | Italycnr-2000 | 17.36 | 1134.34 | 0.91 | |
| | Sk-2005 | 5.51 | 338.22 | 0.99 | |
| | Uk-2005 | 181.19 | 157.13 | 1.00 | |
| **OSNs.** | Slashdot | 10.24 | 173.87 | 0.36 | High degree, small-to-medium-sized and fuzzy communities |
| | Themarker | 29.87 | 458.90 | 0.31 | |
| | BlogCatalog | 47.15 | 1455.48 | 0.32 | |
| **Scientific** | PKUSTK13 | 68.73 | 3,514.56 | 0.88 | High degree, large and clear communities |
| | PWTK | 51.89 | 4,635.81 | 0.93 | |
| | Shipsec1 | 24.36 | 4,117.50 | 0.89 | |
| | Shipsec5 | 24.61 | 5,252.15 | 0.90 | |

As suggested by Newman in [17], a modularity $Q \geq 0.3$ indicates a good community structure. As we can see, OSNs have the lowest modularity as compared to others. This indicates that the communities are fuzzy and overlapping. Since these are social networks, people can be part of several groups in real life (group of friends, family, co-workers, etc.). So, consistent with our earlier results, G1 methods performs the best on networks in this category.

Other networks typically have higher values of modularity $Q$ (ranging between 0.4 and 0.9), and the determining factor of which method is best thus depends on average community size and average degree. The ratio between average degree and average community size of collaboration, technology is around 200, while this ratio on recommendation and web networks is about 35. This ratio indicates how large the community is compared to average degree. The average degrees of these four types of networks are quite low, between 2 and 15. As we expected from the previous experiment, G1 methods perform very well in this case. In contrast, on average, Facebook and scientific computing networks have communities only 12 and 80 times larger than their average degrees, which range from 15 to 70. Method in G2 performs the best.

In view of our earlier experiments, we see that in the networks with communities that are small relative to average degree, G1 methods quickly see all of a community, and then have trouble escaping (because of the strong community structure). However, when communities are large relative to average degree, both G1 and G2 methods tend to stay in the same community for much longer, and G1 methods perform the best.

**Table 6: Summary of the network characteristics and performance of algorithms. Algorithms tend to perform similarly on networks in the same category.**

| Type | Network | Node coverage | | | Edge coverage | | |
|---|---|---|---|---|---|---|---|
| | | G1 | G2 | G3 | G1 | G2 | G3 |
| **Collaboration:** Low $d_{avg}$, Medium $CS_{avg}$, High $Q$ | Citeseer | **25.66±2.94** | 25.15±0.55 | 21.01±0.23 | **20.48±1.68** | 19.49±0.42 | 15.1±0.52 |
| | Dblp-2010 | **32.59±0.12** | 26.53±0.37 | 18.22±0.16 | **29.16±0.08** | 19.53±0.39 | 12.72±0.29 |
| | Dblp-2012 | **38.33±0.04** | 31.74±0.28 | 26.21±0.11 | **33.47±0.03** | 22.39±0.36 | 17.09±0.07 |
| | MathSciNet | **36.14±0.09** | 30.17±0.26 | 24.85±0.13 | **36.27±0.06** | 23.63±0.3 | 18.17±0.09 |
| **Recommendation:** Low $d_{avg}$, Low $CS_{avg}$, High $Q$ | Amazon | **5.71±0.16** | 5.73±0.06 | 5.85±0.18 | **5.60±0.06** | **5.61±0.08** | 5.50±0.17 |
| | Github | **53.59±0.02** | 46.33±0.24 | 30.02±0.08 | **72.57±0.02** | 60.47±0.29 | 25.13±0.18 |
| **Facebook:** High $d_{avg}$, High $CS_{avg}$, High $Q$ | OR | 38.99±2.50 | **55.94±0.68** | 51.00±0.22 | **31.05±3.69** | 27.37±0.57 | 16.2±0.17 |
| | Penn94 | 75.30±1.05 | **82.52±0.34** | 80.07±0.24 | **24.04±1.74** | 19.47±0.41 | 12.39±0.13 |
| | Wosn-friends | 38.20±3.05 | **55.80±0.49** | 50.93±0.19 | **30.92±3.12** | 27.85±0.7 | 16.46±0.17 |
| **Technology**: Low $d_{avg}$, High $CS_{avg}$, High $Q$ | P2P-gnutella | **36.02±0.11** | 32.71±0.17 | 27.74±0.22 | **26.96±0.08** | 20.02±0.1 | 16.13±0.17 |
| | RL-caida | **28.86±0.12** | 27.71±0.47 | 26.62±0.10 | **39.57±0.18** | 30.21±0.85 | 20.26±0.11 |
| **Web**: High $d_{avg}$, Medium $CS_{avg}$, Low $Q$ | Arabic-2005 | **9.47±2.49** | 6.47±0.72 | 9.54±1.48 | **6.97±0.94** | 5.40±1.28 | 6.75±0.95 |
| | Italycnr-2000 | 8.52±2.25 | **15.66±6.37** | 13.65±2.84 | 14.9±3.83 | **24.59±12.51** | 11.93±2.89 |
| | Sk-2005 | **10.33±0.87** | 6.41±1.04 | 8.21±0.65 | **9.69±0.51** | 6.21±0.96 | 8.03±1.02 |
| **OSNs**: High $d_{avg}$, Low-to-medium $CS_{avg}$, Low $Q$ | Slashdot | **70.68±0.01** | 61.23±0.25 | 36.81±0.75 | **75.85±0.01** | 57.74±0.24 | 21.84±0.56 |
| | BlogCatalog | **90.38±0.02** | 90.38±0.37 | 90.38±0.49 | **90.51±0.01** | 82.28±0.32 | 18.81±0.26 |
| | Themarker | **89.48±0.01** | 86.04±0.2 | 47.40±0.12 | **82.28±0.01** | 67.4±0.25 | 19.72±0.12 |
| **Scientific:** High $d_{avg}$, High $CS_{avg}$, High $Q$ | PKUSTK13 | 7.40±0.51 | **43.94**±9.74 | 33.78±1.51 | 5.68±0.17 | **10.58**±0.61 | 9.41±0.22 |
| | PWTK | 5.61±0.12 | **20.08**±2.68 | 15.45±0.74 | 5.27±0.05 | **8.13**±0.21 | 7.99±0.10 |
| | Shipsec1 | 7.81±0.44 | **27.47**±1.43 | 21.77±0.52 | 7.80±0.89 | **9.71**±0.54 | 9.17±0.35 |
| | Shipsec5 | 8.17±0.81 | **27.85**±2.46 | 20.02±0.91 | 8.75±0.50 | **9.79**±0.52 | 9.15±0.44 |

All in all, the G1 and G2 methods are the best, depending on structural properties. G1 methods expand the sampled network by quickly filling out the unobserved nodes (or edges) in a particular region before moving out of the region. However, these methods are obstructed by sharp community borders. In contrast, the G2 method `Random Walk` has the freedom to move around, and so the crawler observes parts of many communities before it fills out individual regions. Because of this, a `Random Walk` takes longer to fully explore regions, but reaches more of the network.

## 5 CONCLUSION

Data collection is the first process of any network analysis task. However, the literature contains a vast selection of network sampling algorithms, and so it is often difficult for users to select a single method that works well for their data, as sampling methods that work well on one network may not work well on a different network. In this paper, we performed a large-scale, comprehensive study to understand how the structural features of networks affect the performance of sampling methods. We identified three network properties of interest: community separation, community size, and average degree. We performed a large set of controlled experiments on synthetic and real graphs, and considered two sampling goals: *node* and *edge* coverage. We considered nine important sampling methods, ranging from well-understood, classical methods like `Random Walk` and `BFS` to modern, cutting-edge algorithms. We performed experiments on real and synthetic networks, and demonstrated that the performance of the sampling methods is highly dependent on the network structure, and in particular, whether the sampling method is able to transition between different regions of the graph. As a result of our experiments, we categorized the nine crawling methods into three groups: `Node Importance-based`, `Random Walk-based` and `Graph Traversal-based` approaches.

We observed that `Random Walk` and `Node Importance-based` algorithms performed well, depending on the network structure. In particular, on networks with clear and sharp community structure, the `Node Importance-based` algorithms tend to get 'stuck' in a region of a graph, while the `Random Walk` method is able to transition between regions. However, when boundaries between communities are fuzzier, or the communities overlap, the `Node Importance-based` methods demonstrate excellent performance.

Finally, we showed how a user can select an appropriate crawling method based on the network type: in particular, the `Random Walk` method is suitable for crawling Facebook and scientific computing networks, but for collaboration, recommendation, technological, web and other online social networks, Node Importance-based methods are best.

## 6 FUTURE WORK

As part of our future research, we plan to investigate different types of query responses. In this work, we assumed that all neighbors are returns for each query. However, in certain settings, this assumption may not hold, and the crawler may need multiple queries to obtain all of a node's neighbors. For example, an online social network API may divide a node's neighbors into *pages*, where each page contains $k$ records and only one page is returned at a time. Also, we want to expand our findings and give the insight on directed networks. It is not clear that our results will generalize to such a setting, and we plan to investigate it further.

## 7 ACKNOWLEDGEMENTS

The authors would like to thank Jeremy Wendt of Sandia National Laboratories for thoughtful comments and conversations.

## REFERENCES

[1] Serge Abiteboul, Mihai Preda, and Gregory Cobena. 2003. Adaptive on-line page importance computation. In *Proceedings of the 12th international conference on World Wide Web*.

[2] Nesreen K. Ahmed, Jennifer Neville, and Ramana Kompella. 2014. Network Sampling: From Static to Streaming Graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 8, 2 (2014).

[3] Yong-Yeol Ahn, Seungyeop Han, Haewoon Kwak, Sue Moon, and Hawoong Jeong. 2007. Analysis of topological characteristics of huge online social networking services. In *International conference on WWW*.

[4] Konstantin Avrachenkov, Prithwish Basu, Giovanni Neglia, Bruno Ribeiro, and Don Towsley. 2014. Pay few, influence most: Online myopic network covering. In *Computer Communications Workshops*.

[5] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* (2008).

[6] Zahy Bnaya, Rami Puzis, Roni Stern, and Ariel Felner. 2013. Bandit algorithms for social network queries. In *2013 International Conference on Social Computing*. IEEE, 148–153.

[7] Minas Gjoka, Maciej Kurant, Carter T Butts, and Athina Markopoulou. 2009. Unbiased sampling of facebook. *preprint arXiv* 906 (2009).

[8] Maciej Kurant, Athina Markopoulou, and Patrick Thiran. 2010. On the bias of BFS (breadth first search). In *Teletraffic Congress (ITC), 2010 22nd International*. IEEE, 1–8.

[9] Maciej Kurant, Athina Markopoulou, and Patrick Thiran. 2011. Towards unbiased BFS sampling. *IEEE Journal on Selected Areas in Communications* 29, 9 (2011), 1799–1809.

[10] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a social network or a news media?. In *19th international conference on World wide web*.

[11] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. 2008. Benchmark graphs for testing community detection algorithms. *Physical review E* 78, 4 (2008), 046110.

[12] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 631–636.

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

[14] Arun S Maiya and Tanya Y Berger-Wolf. 2010. Online sampling of high centrality individuals in social networks. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 91–98.

[15] Arun S Maiya and Tanya Y Berger-Wolf. 2010. Sampling community structure. In *Proceedings of the 19th international conference on World wide web*. ACM, 701–710.

[16] Alan Mislove, Massimiliano Marcon, Krishna P Gummadi, Peter Druschel, and Bobby Bhattacharjee. 2007. Measurement and analysis of online social networks. In *7th ACM SIGCOMM conference on Internet measurement*. ACM, 29–42.

[17] Mark EJ Newman. 2004. Fast algorithm for detecting community structure in networks. *Physical review E* 69, 6 (2004), 066133.

[18] Mostafa Salehi, Hamid R Rabiee, and Arezo Rajabi. 2012. Sampling from complex networks with high community structures. *Chaos* 22, 2 (2012).

[19] Shaozhi Ye, Juan Lang, and Felix Wu. 2010. Crawling online social graphs. In *12th International Asia-Pacific Web Conference*.