Faster Sampling Over Online Social Networks

Zhuojie Zhou

B.S. in Mathematics, May 2010, Zhejiang University

A dissertation submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial satisfaction of the requirements
for the degree of Doctor of Philosophy

August 31, 2015

Dissertation directed by

Nan Zhang
Associate Professor of Computer Science

The School of Engineering and Applied Science of The George Washington University certifies that Zhuojie Zhou has passed the Final Examination for the degree of Doctor of Science as of May 13th 2015. This is the final and approved form of the dissertation.

Faster Sampling Over Online Social Networks

Zhuojie Zhou

Dissertation Committee:

Nan Zhang, Associate Professor of Computer Science, Dissertation Director

Xiuzhen Cheng, Professor of Computer Science, Committee Member

Hyeong-Ah Choi, Professor of Computer Science, Committee Member

Acknowledgment

Foremost, I would like to express my sincere gratitude to my advisor Dr. Nan Zhang for the continuous support of my Ph.D study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D study.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Xiuzhen Cheng, Dr. Hyeong-Ah Choi, and Dr. Hao Howie Huang, for their encouragement, insightful comments, and hard questions.

My sincere thanks also goes to Dr. Gautam Das, Azade Nazi, and Saravanan Thirumuruganathan, for offering me the greatest help and coauthoring the papers.

I thank my friends in George Washington University: Tianyi Song, Weimo Liu, and Chunqiang Hu, for enlightening me the first glance of research.

Last but not the least, I would like to thank my family: my parents Kun Zhou and Shuhui Chen, for giving birth to me at the first place and supporting me spiritually throughout my life.

Abstract

Faster Sampling Over Online Social Networks

Many online social networks feature restrictive web interfaces that only allow the query of a user's local neighborhood through the interface. To enable analytics over such an online social network through its restrictive web interface, many recent efforts reuse the existing Markov Chain Monte Carlo methods such as random walks to sample the social network and support analytics based on the samples. The problem with such an approach, however, is the large amount of queries often required (i.e., a long "mixing time") for a random walk to reach a desired (stationary) sampling distribution. In this dissertation, we targeted the problem by shortening or eliminating the mixing time and leveraging the history of random walks.

To reduce the mixing time, we have developed Modified TOpology (MTO)-Sampler which, by using only information exposed by the restrictive web interface, constructs a "virtual" overlay topology of the social network while performing a random walk, and ensures that the random walk follows the modified overlay topology (with shorter mixing time) rather than the original one. We then extend the idea of MTO from SRW to more general random walks like Metropolis-Hastings Random Walk (MTO-MHRW) and General Random Walks(MTO-GRW).

To eliminate the mixing time, we also introduced WALK-ESTIMATE (WE)-Sampler, which starts with a much shorter random walk, and then proactively estimate the sampling probability for the node taken before using acceptance-rejection sampling to adjust the probability to the predetermined target distribution. We demonstrated the superiority of MTO-Sampler and WE-Sampler over traditional random walks through theoretical analysis and extensive experiments over real world online social networks.

To leverage the history of random walks, we construct a higher-ordered Markov chain. We develop two algorithms, Circulated Neighbors and Groupby Neighbors Random Walk

(CNRW and GNRW) and prove that, no matter what the social network topology is, CNRW and GNRW offer better efficiency than baseline random walks while achieving the same stationary distribution.

Table of Contents

Acknowledgment . . . . . **iii**

Abstract . . . . . **iv**

List of Figures . . . . . **viii**

List of Tables . . . . . **x**

Chapter 1 - Introduction **1**
1.1 Aggregate Estimation over Online Social Networks 1
1.2 Existing Sampling Based Solutions and Their Problems 1
1.3 Reducing the Mixing Time (MTO-Sampler) 3
1.4 Eliminating the Mixing Time (WE-Sampler) 5
1.5 Leveraging the history of random walks 7

Chapter 2 - Preliminaries **10**
2.1 Model of Online Social Networks 10
2.2 Performance Measures for Sampling 10
2.3 Random Walk 12
2.4 Conductance: An Efficiency Indicator 14
2.5 Key for Conductance: Cross-Cutting Edges 15
2.6 Acceptance-Rejection Sampling 16

Chapter 3 - Related Works **18**
3.1 Random Walks 18
3.2 Random Walks on Social Networks 18
3.3 Improving the Efficiency of Random Walks 19

Chapter 4 - Modify-TOpology (MTO) Sampler **20**
4.1 Deterministic Identification of Non-cross-cutting Edges 20
4.2 Conditional Identification of Cross-cutting Edges 24
4.3 Experiments 27

Chapter 5 - Extension of MTO-SRW: MTO-MHRW and MTO-GRW **30**
5.1 Key Ideas for Rewiring a Metropolis-Hastings Random Walk (MTO-MHRW) . 30
5.1.1 Conductance of MHRW 30
5.1.2 Deterministic Identification of non-cross-cutting edges in MHRW 32
5.1.3 Conditional identification of non-cross-cutting edge in MHRW 36
5.2 Key Ideas for Rewiring a General Random Walk (MTO-GRW) 37
5.2.1 Conductance of GRW 38
5.2.2 Deterministic and conditional identification of non-cross-cutting edges in general random walks 39

5.3 Experiments . . . 40
5.3.1 Mixing time comparison between MHRW and MTO-MHRW . . . 40

Chapter 6 - WALK-ESTIMATE (WE) Sampler . . . 42
6.1 Key Idea of WE Sampler . . . 42
6.2 WALK . . . 43
6.2.1 IDEAL-WALK: Main Idea and Analysis . . . 43
6.2.2 Algorithm WALK . . . 49
6.3 ESTIMATE . . . 50
6.3.1 UNBIASED-ESTIMATE . . . 50
6.4 Experiments . . . 52

Chapter 7 - Leveraging the history of Random Walks . . . 55
7.1 Introduction . . . 55
7.1.1 Our Idea: History-Aware Random Walks . . . 55
7.2 CNRW: Circulated Neighbors Random Walk . . . 59
7.2.1 Main Idea and Justification . . . 60
7.2.2 Theoretical Analysis . . . 61
7.2.3 Algorithm CNRW . . . 73
7.3 GNRW: GroupBy Neighbors Random Walk . . . 74
7.3.1 Basic Idea . . . 74
7.3.2 Algorithm GNRW . . . 78
7.4 Discussions . . . 79
7.5 Experiments . . . 80
7.5.1 Experimental Setup . . . 80
7.5.2 Evaluation . . . 84

Chapter 8 - Conclusion . . . 87

References . . . 89

List of Figures

1 A concept of a random walk on the topologically modified overlay graph. . **3**

2 A figure shows that the edge e_{uv} cannot be the cross-cutting edge in theorem 1. Locally, (a) and (c) have 6 cross-cutting edges, while (b) and (d) only have 5 of them. **20**

3 An counter example of lemma 1 that edge e_{uv} must be a cross-cutting edge. **23**

4 Replace the edge e_{uv} with e_{uw} **24**

5 When replacing e_{uv} with e_{uw} from a node v whose degree is larger than 3, it may decrease the conductance if both e_{uv} and e_{wv} are cross-cutting edges. **25**

6 Bias vs. Query Cost tests for local datasets' average degree. **27**

7 Comparison between SRW and MTO on query cost and the KL divergence measure over all three datasets. **28**

8 Varying Geweke Threshold to get different KL divergence on dataset Slashdot B. KL and QC stands for KL divergence and Query Cost respectively. . **28**

9 Theoretical mixing time on latent space graph model. MTO_Both: Remove and replace edges. MTO_RM: Remove edges. MTO_RP: Replace edges. . . **28**

10 Google Plus online social network **29**

11 An example of the edge categories in rewiring MHRW. Edge e_{uv}, e_{wv} are cross-cutting edges. Edge B is a margin non-cross-cutting edge that is associated with an end of a cross-cutting edge. Edge A and edge C are inner non-cross-cutting edges that are not connected to any end of the cross-cutting edges. **31**

12 In Metropolis-Hastings Random Walk, if e_{uv} is a cross-cutting edge, we may need to check every possible separation of S and $\bar{S}$ to determine whether e_{uv} can be removed or not. **32**

13 Example of modified Graph for MHRW. Graph G is a graph with multiple edges crossing two complete graphs (multiple bridges bar-bell graph), and each of them has 6 nodes. Graph G^* is a graph modified by Theorem 3. . . **34**

14 A barbell graph example that demonstrates the starting node selection in MTO-MHRW **35**

15 Replacing e_{uv} with e_{uw} from a node v in MHRW **36**

16 Mixing time speedup distribution of initial nodes for MHRW and SRW in a subgraph of Facebook dataset. **41**

17 Minimum and maximum probabilities vs walk length **44**

18 Relative Error of the Average Estimations vs Query Cost in Google Plus. . **53**

19 Relative Error of the Average Estimations vs Query Cost in Yelp. **54**

20 Relative Error of the Average Estimations vs Query Cost in Twitter (from SNAP repository). **55**

21 Improvement Trend: Relative Error of the Average Estimations vs Query Cost in Google Plus. **56**

22 Relative Error of the Average Estimations vs Number of Samples in Google Plus. **57**

23 Demo of CNRW, it chooses the next candidate from the set $N(v) - b(u, v)$. . **60**

24 Path blocks in CNRW **63**

25 Comparison of the block distribution in CNRW and SRW. **69**

26 An example of partitioning a node's neighbors into 3 groups. **75**

27 A demo of GNRW **76**

28 Large Google Plus Graph: estimation of average degree. **84**

29 Public benchmark datasets **84**

30 Distribution of SRW, CNRW and GNRW **85**

31 Yelp dataset: GNRW strategies. **86**

32 Synthetic datasets: clustered graph **87**

33 Barbell graph size analytics. **87**

List of Tables

1 Summary of the datasets in the experiments. 81

Chapter 1 - Introduction

1.1 Aggregate Estimation over Online Social Networks

An online social network allows its users to publish contents and form connections with other users. To retrieve information from a social network, one generally needs to issue a *individual-user query* through the social network's web interface by specifying a user of interest, and the web interface returns the contents published by the user as well as a list of other users connected with the user[1].

An online social network not only provides a platform for users to share information with their acquaintance, but also enables a third party to perform a wide variety of analytical applications over the social network - e.g., the analysis of rumor/news propagation, the mining of sentiment/opinion on certain subjects, and social media based market research. While some third parties, e.g., advertisers, may be able to negotiate contracts with the network owners to get access to the full underlying database, many third parties lack the resources to do so. To enable these third-party analytical applications, one must be able to accurately estimate big-picture aggregates (e.g., the average age of users, the COUNT of user posts that contain a given word) over an online social network by issuing a small number of individual-user queries through the social network's web interface. We address this problem of third-party aggregate estimation in the paper.

1.2 Existing Sampling Based Solutions and Their Problems

An important challenge facing third-party aggregate estimation is the lack of cooperation from online social network providers. In particular, the information returned by each individual-user query is extremely limited - only containing information about the neighborhood of one user. Furthermore, almost all large-scale online social networks enforce limits on the number of web requests one can issue (e.g., 600 open graph queries per 600

[1] We currently focus on the undirected relationship between users.

seconds for Facebook[2], and 350 requests per hour for Twitter[3]). As a result, it is practically impossible to crawl/download most or all data from an online social network before generating aggregate estimations. There is also no available way for a third party to obtain the entire topology of the graph underlying the social network.

To address this challenge, a number of sampling techniques have been proposed for performing analytics over an online social network without the prerequisite of crawling [14, 16, 17, 21]. The objective of sampling is to randomly select elements (e.g., nodes/users or edges/relationships) from the online social network according to a pre-determined probability distribution, and then to generate aggregate estimations based on the retrieved samples. Since only individual local neighborhoods (i.e., a user and the set of its neighbors) - rather than the entire graph topology - can be retrieved from the social network's web interface, to the best of our knowledge, all existing sampling techniques without prior knowledge of all nodes/edges are built upon the idea of performing *random walks* over the graph which only require knowledge of the local neighborhoods visited by the random walks.

A critical problem of existing sampling techniques, however, is the large number of individual-user queries (i.e., web requests) they require for retrieving each sample. Consider the simple random walk as an example. In order to reach the stationary distribution (and thereby an accurate aggregate estimation), one may have to issue a large number of queries as a "burn-in" period of the random walk. Traditional studies on graph theory found that the length of such a burn-in period is determined by the graph *conductance* - an intrinsic property of the graph topology (formally defined in Section 1.5). In particular, the smaller the conductance is, the longer the burn-in period will be (i.e., the more individual-user queries will be required by sampling).

Unfortunately, a recent study [28] on real-world social networks such as Facebook, Livejournal, etc. found the conductance of their graphs to be substantially lower than expected. As a result, a random walk on these social networks often requires a large number

[2]https://developers.facebook.com/docs/best-practices/
[3]https://dev.twitter.com/docs/rate-limiting

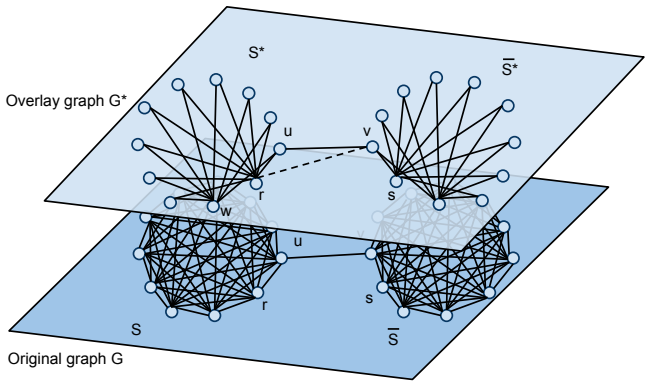


Figure 1: A concept of a random walk on the topologically modified overlay graph.

of individual-user queries - e.g., approximately 500 to 1500 single random walk length for a real-world social network Livejournal of one million nodes to achieve acceptable variance distance [28]. One can see that, in order to retrieve enough samples to reach an accurate aggregate estimation, the existing sampling techniques may require a very large number of individual-user queries.

1.3 Reducing the Mixing Time (MTO-Sampler)

We consider a novel problem of how to significantly reduce the mixing time of a random walk on a social network graph by modifying the graph topology on-the-fly (during the third-party random walk process). In the following, we shall first explain what we mean by on-the-fly topology modification, and then describe the rationale behind our main ideas for topology modification.

First, by topology modification we do *not* actually modify the original topology of the social network graph - indeed, no third party other than the social network provider has the ability to do so. What we modify is the topology of an *overlay graph* on which we perform the random walks. Fig 1 depicts an example: if we can decide that not considering a particular edge in the random walk process can make the burn-in period shorter (i.e., increase the conductance), then we are essentially performing random walks over an overlay graph on which this edge is removed. By doing so, we can achieve same accurate aggregate esti-

mation with lower query cost. One can see that, with traditional random walk techniques, the overlay graph is exactly the same as the original social network graph. Our objective here is to manipulate edges in the overlay graph so as to maximize the graph conductance.

It is important to note that the technical challenge here is not how edge manipulations can boost graph conductance - a simple method to reach theoretical maximum on conductance is to repeatedly insert edges to the graph until it becomes a complete graph. This requires the knowledge of all nodes in the social network, which a third-party does not have. The key challenge here is how to perform edge manipulations only based on the knowledge of local neighborhoods that a random walk has passed by, and yet increases the conductance of the entire graph in a significant manner. In the following, we provide an intuitive explanation of our main ideas for topology modification.

To understand the main ideas, we first introduce the concepts of *cross-cutting* and *non-cross-cutting* edges intuitively with an example in Fig 1 (we shall formally define these concepts in Section 1.5). Generally speaking, if we consider a social network graph consisting of multiple densely connected components (e.g., S and $\bar{S}$ in Fig 1), then the edges connecting them are likely to be cross-cutting edges, while edges inside each densely connected component are likely non-cross-cutting ones. A key intuition here is that the more cross-cutting edges and/or the fewer non-cross-cutting edges a graph has, the higher its conductance is. For example, Graph G in Fig 1 has a low conductance (i.e., high burn-in period) as a random walk is likely to get "stucked" in one of the two dense components which are difficult to escape, given that there is only one cross-cutting edge (u, v). On the other hand, with far fewer non-cross-cutting edges and a few additional cross-cutting edges, G^* has a much higher conductance as it is much easier now for a random walk to move from one component to the other.

With the concepts of cross-cutting and non-cross-cutting edges, we develop Modify TOpology Sampler (MTO-Sampler), a topology manipulation technique which first de-

termines[4] whether a given edge in the graph is a cross-cutting edge based solely upon knowledge of the local neighborhood topology, and then removes the edge if it is non-cross-cutting. MTO-Sampler may also "move" an edge by changing a node connected to the edge if it is determined that, by doing so, the new edge is more likely to be a cross-cutting edge. We shall show that MTO-Sampler is capable of significantly improving the efficiency of random walks: For the example in Fig 1, MTO-Sampler is capable of reducing the upper bound on mixing time (i.e., query cost of a random walk) by 97%.

1.4 Eliminating the Mixing Time (WE-Sampler)

Eliminating the mixing time means that we are going to start collecting samples from a random walk regardless of its convergence. The overview of this idea is in the design of WALK-ESTIMATE (WE)-Sampler. The WE-Sampler is enabled by two main components: WALK and ESTIMATE. The WALK component determines how many, say t, steps to (randomly) transit before taking a node v as a candidate (for sampling), and then calls upon the ESTIMATE component for an estimation of the probability for the walk to reach v after t steps. Based on the estimated probability, WALK then performs acceptance-rejection sampling on v to determine if it should be included in the final (output) sample.

For the WALK component, we start by developing IDEAL-WALK, an impractical sampler which makes two ideal-case assumptions: One is access to an oracle that precisely compute the $p_t(v)$, i.e., the probability for the walk to reach a node v at Step t. The other is access to certain *global* topological information about the underlying graph - e.g., $|E|$, the total number of edges in the graph, $\mathcal{D}(G)$, the graph diameter, λ, the spectral gap of the transition matrix, and $d_{\max}$, the maximum degree of a node in the graph - so that IDEAL-WALK can determine the optimal number of steps t to walk. We rigidly prove that no matter what the target distribution is (barring certain extreme cases, e.g., when the distri-

[4]Note that, it is impossible to assert deterministically that an edge is cross-cutting. Nonetheless, it is possible to assert deterministically that an edge is non-cross-cutting. Thus, our algorithm has two possible outputs: non-cross-cutting or uncertain. We shall show in the paper that it outputs non-cross-cutting for a large number of (non-cross-cutting) edges in real-world social networks.

bution is 1 on the starting node and 0 on all others), IDEAL-WALK *always* outperforms its corresponding traditional random walk algorithm. Further, it also produce samples with absolutely zero bias (while random walks often cannot, depending on the graph topology). We also demonstrate through analysis of multiple theoretical graph models the significance of such efficiency enhancements for the sampling process.

Of course, IDEAL-WALK makes two unrealistic assumptions, which we remove through the design of Algorithms WALK and ESTIMATE, respectively. Algorithm WALK removes the assumption of access to global parameters by requiring access to only one parameters (besides the local neighborhood of the current node): $\bar{\mathcal{D}}(G)$, i.e., an upper bound on the diameter of the graph - which is often easy to obtain (e.g., it is commonly believed that 8 to 10 is a safe bet for real-world online social networks [5, 27]).

Algorithm ESTIMATE, on the other hand, estimates $p_t(v)$, i.e., the probability for Algorithm WALK to sample node v at Step t. To illustrate our main idea here, we start by developing UNBIASED-ESTIMATE, a simple algorithm which takes a *backward* random walk from Node v for estimating $p_t(v)$. We rigidly prove the unbiasedness of the estimation produced by UNBIASED-ESTIMATE. Nonetheless, we also note its problem: a high estimation variance which grows rapidly with the number of backward steps one has to take for producing the estimation. Since the error of estimation is determined by both bias and variance, the high variance produced by UNBIASED-ESTIMATE introduces significant error in the estimation of $p_t(v)$.

To address the problem of UNBIASED-ESTIMATE, we introduce two main ideas for variance reduction in developing Algorithm ESTIMATE, our final algorithm for estimating $p_t(u)$: One is *initial crawling*, i.e., the crawl of the h-hop neighborhood (where h is a small number like 2 or 3) of the starting node to reduce the number of backward steps required by Algorithm ESTIMATE, and the second is *weighted sampling*, i.e., to carefully design the transition matrix of the backward random walk process to reduce the variance of estimation. We shall demonstrate through experimental evaluation that Algorithm ESTIMATE

significantly reduces the estimation variance for $p_t(u)$. Finally, we combine Algorithms WALK and ESTIMATE to produce Algorithm WALK-ESTIMATE.

1.5 Leveraging the history of random walks

With the broad penetration of online social networks and the multitude of information they capture, how to enable a *third party*[5] to perform efficient analytics of data available on social networks - specifically, to answer global and conditional aggregates such as SUM, AVG, and COUNT (e.g., the average friend count of all users living in Texas) - has become an increasing important research problem in the database community [29, 34, 35]. Applications of such aggregate estimations range from sociology research, understanding economic indicators to observing public health trends, bringing benefits to the entire society.

Technically, the challenge of data analytics over online social networks mainly stems from the limitations of their (web) query interfaces available to a third party - most online social networks only allow *local neighborhood queries*, with input being a node (i.e., user) and output being its immediate neighbors. Such a query interface makes retrieving the entire graph topology prohibitively expensive for a third party (in terms of query cost - due to the large size of real-world social networks). To address the problem, existing studies resort to a sampling approach, specifically the sampling of nodes (i.e., users) through the restrictive interface of a social network [38, 40], to enable aggregate estimations based on the sampled nodes. This is also the approach we focus on in the paper.

While there have been a wide variety of social network sampling designs [14, 18, 19, 21] proposed in the literature, the vast majority of them share the same core design: *random walk*, a Markov Chain Monte Carlo (MCMC) method which iteratively transits from one node to a random neighbor. The variations differ in their design of the transition probabilities, i.e., the probability distributions for choosing the random neighbor, which are naturally

[5] i.e., one who is not the social network owner - examples include sociologists, economists, etc.

adjusted for the various types of analytical tasks they target. Nonetheless, the core random walk design remains the same - after performing the random walk for a number of steps, the current node is taken as a sample - and, after repeated executions of random walks, we can generate (statistically accurate) aggregate estimation from the multiple sample nodes.

To understand the problem with this core design, it is important to observe the key bottleneck of its practical implementations. Note that such a sampling design incurs two types of overhead: One is query cost: Each transition in the random walk requires one *local neighborhood query* (described above) to be issued to the online social network, while almost all real-world social networks enforce rigid query-rate limits (e.g., Twitter allows only 15 local neighborhood queries every 15 minutes). The second type is the local processing (time and space) overhead for recording sampled nodes and computing aggregate estimations. One can see that the bottleneck here is clearly the query cost - compared with the extremely low query rate allowed by online social networks (e.g., 1 minute/query for Twitter), the local processing overhead (linear to the sample size [23]) is negligible.

With the understanding that a sampling algorithm must minimize its query cost, the problem with existing techniques can be summarized in one sentence: The core random walk design requires a long "burn-in" before a node can be taken as a sample - since each step requires one query, this leads to a high query cost and therefore a very slow sampling process over real-world online social networks.

To understand what the burn-in period is and why it is required, we note that a sample node can only be used for analytics if we know the *bias* of the sampling process, i.e., the probability for a node to be taken as a sample. The reason for this requirement is simple - only with knowledge of the sampling bias can we properly correct it to ensure equal representation of all applicable tuples. Nonetheless, actually learning the sampling distribution without knowledge of the global graph topology is difficult. A desirable property of random walk is that it asymptotically converges (as it grows longer) to a *stationary* sampling distribution that can be derived without knowledge of the graph topology (e.g., with proba-

bility proportional to a node's degree for simple random walk [33], or uniform distribution for Metropolis-Hastings random walk [33]). The number of steps required for a random walk to reach this stationary distribution is the "burn-in" period which, unfortunately, is often quite long for real-world social networks [14, 40].

Chapter 2 - Preliminaries

2.1 Model of Online Social Networks

In this paper, we consider an online social network with an interface that allows input queries of the form

$$q(v)\text{: SELECT * FROM D WHERE USER-ID = } v,$$

and responds with the information about user v (e.g., user name, self-description, user-published contents) as well as the list of all other users connected with v (e.g., v's friends in the network). This is a model followed by many online social networks - e.g., Google Plus, Facebook, etc - with the interface provided as either an end-user-friendly web page or a developer-specific API call.

Consider the social-network topology as an undirected graph $G(V, E)$, where each node in V is corresponding to a user in the social network[6], and each edge in E represents the connection between two users. One can see that the answer to query $q(v)$ ($v \in V$) is a set of nodes $N(v) \subseteq V$, such that $\forall u \in N(v)$, there is an edge $e : (u, v) \in E$. We henceforth refer to $N(v)$ as the *neighborhood* of v. We use k_v to denote the *degree* of v - i.e., $k_v = |N(v)|$. For abbreviation, we also write $e : (u, v)$ as e_{uv}.

Running Example: We shall use, throughout this paper, the 22-node, 111-edge, barbell graph shown (as the original graph G) in Fig 1 as a running example.

2.2 Performance Measures for Sampling

There are two important objectives for sampling from an online social network: minimizing the sampling bias and minimizing the query cost. We define the two performance measures corresponding to the two objectives respectively as follows.

[6]Note that without introducing ambiguity, we use "node" and "social network user" interchangeably in this paper.

Sampling Bias: Intuitively, sampling bias measures the "distance" between the stationary distribution of a random walk and the actual sampling distribution achieved by the actual execution of the random walk algorithm. To understand why this is an important measure, we note that there is an inherent tradeoff between such a distance and the query cost required for sampling: If one does not care about the sampling bias and opts to stop a random walk right where it starts, the sampling bias will be the distance between the stationary distribution and a probability distribution vector of the form $\{1, 0, \ldots, 0\}$ (i.e., while the starting node has probability 100% to be sampled, the other nodes have 0%) - i.e., an extremely large sampling bias.

We propose to measure sampling bias with three metrics for various purposes in this paper: KL-divergence [10], ℓ_2-distance [10], and a "golden measure" of the error of aggregate estimations produced by the sample nodes. Specifically, while the former two measures can be used in theoretical analysis and experimental analysis on small graphs, they are infeasible to compute for large graphs used in our experimental studies - motivating us to use the last measure as well. We shall further discuss the bias and the accuracy of various random walk algorithms in the experiments section 7.5.

Query Cost. Another key performance measure for sampling over an online social network is the *query cost* - i.e., the number of queries one has to issue in order to obtain a sample node. Note that query cost here is defined as the number of *unique queries* required, as any duplicate query can be immediately retrieved from local cache without consuming the aforementioned query rate limit enforced by the online social network.

Asymptotic Variance. Variance is one of the most important things to justify the performance of random walk algorithms. Of course, we can measure the sampling bias given certain query cost, but after sufficient number of steps, the efficiency of an estimator provided by random walk samples is directly tied to the variance, which is also important for us to theoretically compare the performance of random walks. For the purpose of this paper, we adopt the commonly used asymptotic variance in Markov Chain.

Definition 1. [Asymptotic Variance]. *Given a function $f(\cdot)$ and an estimator $\hat{\mu}_n = \frac{1}{n}\sum_{t=1}^{n} f(X_t)$, the asymptotic variance of the estimator is defined as*

$$V_\infty(\hat{\mu}) = \lim_{n\to\infty} n\mathit{Var}(\hat{\mu}_n) \tag{1}$$

Note that this does not depend on the initial distribution for X_0. In addition, in practice, we would use an estimator based on only X_t with t greater than some very large number - say h - that we believe the chain has reached a distribution close to its stationary distribution π after h steps.

2.3 Random Walk

A random walk is a Markov Chain Monte Carlo (MCMC) method which takes successive random steps on the above-described graph G according to a *transition matrix* $P = (p_{uv}), u, v \in V$, where p_{uv} represents the probability for the random walk to transit from node u to v. The premise here is that, after performing a random walk for a sufficient number of steps, the probability distribution for the walk to land on each node in G converges to a *stationary distribution* π which then becomes the sampling distribution[7]. There are many different types of random walks, corresponding to the different designs of P and different stationary distributions. In this paper, we consider the simple random walk which has a stationary distribution of $\pi(v) = k_v/(2|E|)$ for all $v \in V$.

Definition 2. ***(Simple Random Walk).*** *Given a current node v, a simple random walk chooses uniformly at random a neighboring node $u \in N(v)$ and transit to u in the next step - i.e.,*

$$P_{vu} = \begin{cases} 1/k_v & \textit{if } u \in N(v), \\ 0 & \textit{otherwise.} \end{cases} \tag{2}$$

One can see that each step of a simple random walk requires exactly one query (i.e.,

[7]That is, if we take the end node as a sample

$q(v)$ to identify the neighborhood of v and select the next stop u). Thus, the performance of sampling - i.e., the tradeoff between bias and query cost - is determined by how fast the random walk converges to the stationary distribution. Formally, we measure the convergence speed as the *mixing time* defined as follows.

Definition 3. ***(Mixing Time)*** *Given $G : (V, E)$, after t steps of simple random walk, the* relative point-wise distance *between the current sampling distribution and the stationary distribution is*

$$\triangle(t) = \max_{u,v \in V, v \in N(u)} \left\{ \frac{|P_{uv}^t - \pi(v)|}{\pi(v)} \right\} \tag{3}$$

where P_{uv}^t is the element of P^t with indices u and v. The mixing time *of the random walk is the minimum value of t such that $\triangle(t) \leq \epsilon$ where ϵ is a pre-determined threshold on relative point-wise distance.*

One can see from the definition that the relative point-wise distance $\triangle(t)$ measures the bias of the random walk after t steps. Mixing time, on the other hand, captures the query cost required to reduce the bias below a pre-determined threshold ϵ. In the following subsection, we describe a key characteristics of the graph which determines the mixing time - the conductance of the graph.

In practice, a popular technique for checking (on-the-fly) whether a random walk has reached its stationary distribution is called the *convergence monitors* (i.e. MCMC convergence diagnostics) [22]. For example, the Geweke method (summarized in [8]) considers two "windows" of a random walk with length l: Window A is formed by the first 10% steps, and Window B is formed by the last 50%. According to [39], if the random walk indeed converges to the stationary distribution after burn-in, then the two windows should

be statistically indistinguishable. Let

$$Z = \left| \frac{\bar{\theta}_A - \bar{\theta}_B}{\sqrt{\hat{S}_\theta^A + \hat{S}_\theta^B}} \right|, \tag{4}$$

where θ is the attribute that can be retrieved from nodes (a typical one is the degree of a node), and $\bar{\theta}_A, \bar{\theta}_B$ are means of θ for all nodes in Windows A and B, respectively, and S_θ^A and S_θ^B are their corresponding variances. One can see that $Z \to 0$ when the random walk converges to the stationary distribution. We use Geweke method as the convergence monitor in the experiments, and we set the threshold to be $Z \leq 0.1$ by default, while also performing tests with the threshold $Z \leq 0.01$.

A property of the graph which has been proven to be strongly correlated with the burn-in period length is the *spectral gap* of the transition matrix T. We denote the spectral gap as $\lambda = 1 - s_2$ where s_2 is the second largest eigenvalue of T.

2.4 Conductance: An Efficiency Indicator

Intuitively, the conductance Φ, which indicates how fast the simple random walk converges to its stationary distribution, measures how "well-knit" a graph is. Specifically, the conductance is determined by a *cut* of the graph G - i.e., a partition of V into two disjoint subsets S and $\bar{S}$ - which minimizes the ratio between the probability for the random walk to move from one partition to the other and the probability for the random walk to stay in the same partition. Formally, we have the following definition.

Definition 4. ***(Conductance)**. The conductance*[8] *of a graph $G : (V, E)$ is*

$$\Phi(G) = \min_{S \subseteq V} \frac{|\{e_{uv} | u \in S, v \in \bar{S}\}|}{\min\left\{|\{e_{uv} | u \in S, v \in V\}|, |\{e_{uv} | u \in \bar{S}, v \in V\}|\right\}}.$$

The relationship between the graph conductance and the mixing time of a simple ran-

[8] Rigidly, the conductance is determined by both the graph topology and the transition matrix of the random walk. Here we tailor the definition to the simple random walk considered in this paper.

dom walk is illustrated by the following inequality [3]:

$$(1 - 2\Phi(G))^t \leq \triangle(t) \leq \frac{2|E|}{\min_{v \in V} k_v} \left(1 - \frac{\Phi(G)^2}{2}\right)^t. \tag{5}$$

One can see that the graph conductance $\Phi(G)$ ranges between 0 and 1 - and the larger $\Phi(G)$ is, the smaller the mixing time will be (for a fixed threshold ϵ). Also note from (5) the log scale relationship between $\Phi(G)$ and the mixing time. This indicates a small change on $\Phi(G)$ may lead to a significant change of the mixing time. For example, increasing conductance from 0.010 to 0.012 will change the upper bound of mixing time from $46050.5 \cdot \log(c/\epsilon)$ to $31979.1 \cdot \log(c/\epsilon)$.

Running Example: The conductance of the barbell graph in the running example is $\Phi(G) = 1/(\binom{11}{2} + 1) = 0.018$. The corresponding (and unique) S and $\bar{S}$ are shown in Fig 1. Correspondingly, the mixing time to reach a relative point-wise distance of $\triangle(t) \leq \epsilon$ is bounded from above by $14212.3 \cdot \log(22.2/\epsilon)$. We shall show throughout the paper how our on-the-fly topology modification techniques can significantly increase conductance and reduce the mixing-time upper-bound for this running example.

2.5 Key for Conductance: Cross-Cutting Edges

A key observation from Definition 4 is that the graph conductance critically depends on the number of edges which "cross-cut" S and $\bar{S}$ - i.e., $|\{e_{uv} | u \in S, v \in \bar{S}\}|$. The more such cross-cutting edges there are, the higher the graph conductance is likely to be. On the other hand, since a non-cross-cutting edge is only counted in the denominator, the more non-cross-cutting edges there are in the graph, the lower the conductance is likely to be. Formally, we define cross-cutting edges as follows.

Definition 5. ***(Cross-cutting edges)***. *For a given graph $G(V, E)$, an edge e_{uv} is a cross-cutting edge if and only if there exists $S \subseteq V$ such that $u \in S$, $v \in \bar{S}$ where $\bar{S} = V \backslash S$,*

and

$$\varphi(S) = \frac{|\{e_{uv}|u \in S, v \in \bar{S}\}|}{\min\left\{|\{e_{uv}|u \in S, v \in V\}|, |\{e_{uv}|u \in \bar{S}, v \in V\}|\right\}} \tag{6}$$

takes the minimum value among all possible $S \subseteq V$.

We note that in large graphs such as online social networks, it is reasonable to assume that the number of cross-cutting edges is relatively small when compared to total number of edges in S or $\bar{S}$.

One can see that our objective of on-the-fly topology modification is then to increase the number of cross-cutting edges and decrease the number of non-cross-cutting edges as much as possible.

Running Example: For the barbell graph, adding any edge between the two halves of the graph produces a new cross-cutting edge, and increases the graph conductance from $\Phi(G) = 0.018$ to 0.035 - i.e., the mixing-time upper-bound will be reduced to 3758.1/14212.3 = 0.264 - a significant reduction of 75%.

2.6 Acceptance-Rejection Sampling

Acceptance-rejection sampling (hereafter referred to as rejection sampling) is a technique we use to "correct" a sampling distribution to the desired target distribution. To understand how, consider the case where our algorithm samples a node with probability $p(u)$, while the desired distribution assigns probability $q(u)$ to node u. In order to make the correction, we take as input a node u sampled by our algorithm, and "accept" it as a real sample with probability

$$\beta(u) = \frac{q(u)}{p(u)} \cdot \min_{v \in V} \frac{p(v)}{q(v)}, \tag{7}$$

because after such correction, the probability distribution of node u in the final sample is

$$\frac{p(u) \cdot \beta(u)}{\sum_{u \in V}(p(u) \cdot \beta(u))} = q(u), \tag{8}$$

which conforms to the desired target distribution.

A practical challenge one often faces when applying rejection sampling is the difficulty of computing $\min_v p(v)/q(v)$, especially when the graph topology is not known beforehand. Even when a theoretical lower bound on $\min_v p(v)/q(v)$ can be computed, its value is often too small to support the practical usage of rejection sampling. A common practice to address this challenge is to replace $\min_v p(v)/q(v)$ with a manual threshold (e.g., [9,12]). Note that a large threshold might introduce bias to the sample - e.g., a threshold greater than $\min_v p(v)/q(v)$ would make the computed $\beta(u) > 1$ for certain nodes, essentially undersampling them in the final sample. Nonetheless, such a large threshold also improves the efficiency of sampling, as the rejection probability will be lower. Of course, such an approximation can be made more conservatively (i.e., lower) to reduce bias, or more aggressively (i.e., higher) to make the sampling process more efficient.

Chapter 3 - Related Works

3.1 Random Walks

Random walk is an MCMC based sampling method extensively studied in statistics (e.g, [13]). Besides the traditional random walk designs described before, two key related concepts used in this paper are *burn-in period*, which captures the number of steps a random walk takes before converging to its stationary distribution [26]; and convergence monitors, heuristic techniques for measuring on-the-fly how long the burn-in period should be (i.e., determining when a random walk should be stopped and a sample taken). Examples here include Geweke, Raftery and Lewis, Gelman and Rubin convergence monitors (see [8] for a comprehensive review).

3.2 Random Walks on Social Networks

There have been extensive studies (e.g., [2, 18, 21]) on the sampling of online social networks which feature graph browsing interfaces [37] that enforce the aforementioned local-neighborhood-only access limitation. [21] introduces a taxonomy of sampling techniques - specifically, node sampling, edge sampling and subgraph sampling. For the problem studied in this paper - i.e., sampling nodes from online social networks - the usage of multiple parallel random walks is studied in [4], while several studies (e.g., [21]) demonstrates the superiority of random walk techniques such as Simple Random Walk (SRW), Metropolis-Hastings Random Walk (MHRW) over baseline solutions such as Breadth First Search (BFS) and Depth First Search (DFS). An interesting issue studied in the literature is the comparison between SRW and MHRW over real-world social networks - the finding in [14] is that MHRW is less efficient than SRW because MHRW mixes more slowly.

3.3 Improving the Efficiency of Random Walks

Most related to this paper are the previous studies on improving the efficiency of random walks over online social networks. To this end, [16] combines random jump and MHRW to efficiently retrieve uniform random node samples from an online social networks. Nonetheless, in order to enable random jumps, this study assumes access to an *ID generator* which can sample a node uniformly at random with a high hit rate - an assumption that is not satisfied by many online social networks and not assumed in this paper. Another study [32] considers frontier sampling which converts input samples with uniform distribution to output samples with arbitrary target distribution. Our study in the paper is transparent to this work - as we address the problem of generating sample nodes rather than assuming access to samples with pre-determined distributions. Also related to efficiency enhancements are [20] which introduces a non-backtracking random walk that converges faster with less asymptotic variance than SRW and [39] which modifies the topology of the underlying graph on-the-fly in order to get a faster random walk on the modified graph.

4.1 Deterministic Identification of Non-cross-cutting Edges

To illustrate the main idea of our deterministic identification of non-cross-cutting edges (for removals), we start with an example in Fig 2 to show why we can determine, based solely upon the local neighborhoods of u and v as shown in the graph, that $e : (u, v)$ (henceforth denoted by e_{uv}) in the Fig is not a cross-cutting edge. The intuition behind this is fairly simple: When u and v share a large number of common neighbors (e.g., 5 in Fig 2) but have relatively few other edges (e.g., 1 each in Fig 2), it is highly unlikely for the partition to cut through e_{uv} rather than the other edges of u and v - e.g., (u, u_0) in Fig 2 - if it cuts through any edges associated with u and v at all.

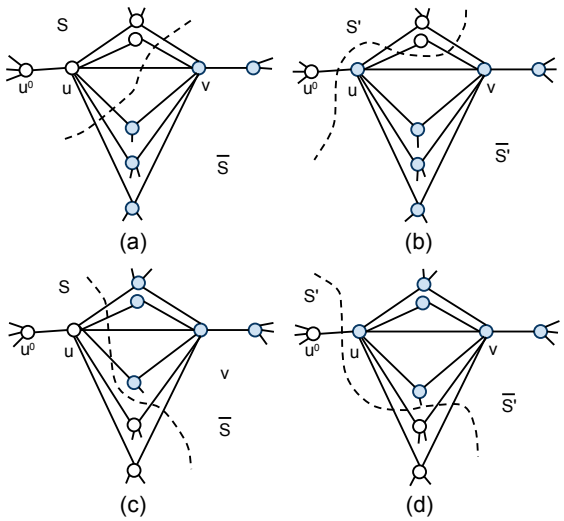


Figure 2: A figure shows that the edge e_{uv} cannot be the cross-cutting edge in theorem 1. Locally, (a) and (c) have 6 cross-cutting edges, while (b) and (d) only have 5 of them.

The rigid (dis-)proof can be constructed with contradiction. Suppose e_{uv} is a cross-cutting edge between two partitions of the graph, S and $\bar{S}$. One can see that since u and v belong to different partitions, there must be at least 6 cross-cutting edges in the subgraph (Fig 2 (a) depicts an example). We now show in the following discussion that this is actually impossible because one can always construct another partition S' and $\bar{S}'$ (by "dragging" u and v into the same part) and reduce the number of cross-cutting edges to at most 5. Note

that this contradicts the definition of S and $\bar{S}$ being a configuration which minimizes the number of cross-cutting edges. Thus, e_{uv} cannot be a cross-cutting edge.

To understand how the construction of S' and $\bar{S}'$ works, consider Fig 2 (b) as an example. For the partition illustrated in Fig 2 (a), we can "drag" u into $\bar{S}$ to form the new configuration, such that the number of cross-cutting edges associated with u and v is now at most 5, as shown in Fig 2 (b). Note that the other edges not shown in the subgraph (no matter cross-cutting or not) are not affected by the re-configuration, because all vertices associated with u are already known in the local neighborhood of u (shown in Fig 2).

More generally, for the other possible settings of S and $\bar{S}$ (such as Fig 2(c)), one can construct the re-configuration in analogy with the following general principle: First, find the "more popular" partition (i.e., either S or $\bar{S}$) among the 5 common neighbors of u and v (e.g., $\bar{S}$ in Fig 2 (a) or Fig 2 (c)). Then, drag one of u and v to ensure that both of them are in this more popular partition under the new configuration. One can see that, since at most 2 common neighbors of u and v are in the less popular partition, the number of cross-cutting edges under the new configuration is at most $2 * 2 + 1$, where $2 * 2$ is the number of cross-cutting edges associated with the 2 common neighbors in the less popular partition (at most 2 for each), and 1 is the number of cross-cutting edge associated with the other (non-common) neighbor of the node being dragged (i.e., u^0 in Fig 2 (a)).

The following theorem depicts the general case for which we can remove an edge on-the-fly to increase the graph conductance. Recall that $N(u)$ and k_u represent the set of neighbors and the degree of a node u, respectively.

Theorem 1. [Edge Removal Criteria]: *Given $G(V, E)$, $\forall u, v \in V$, if $e_{uv} \in E$ and*

$$\left\lceil \frac{|N(u) \cap N(v)|}{2} \right\rceil + 1 > \frac{1}{2} \max\{k_u, k_v\}, \tag{9}$$

then e_{uv} is not a cross-cutting edge.

Proof. Let $n = |N(u) \cap N(v)|$, without losing generality, assuming $u \in S, v \in \bar{S}$, then

there must be n cross-cutting edges in these n disjoint paths of length 2 between u and v. We denote n_u, n_v as the number of cross-cutting edges in these n paths connected with u and v, so $n_u + n_v = n$. One can see that if we try to "drag" u from $u \in S$ to $u \in \bar{S}$, all the edges connected with u would be modified, e.g. flip the edges linked to u, which means the old cross-cutting edges will be the new non-cross-cutting edges, and vice versa. As the assumption from inequality (9): $\lceil \frac{n}{2} \rceil + 1 > \frac{1}{2} \max\{k_u, k_v\}$, so either $n_u + 1 > \frac{1}{2} k_u$ or $n_v + 1 > \frac{1}{2} k_v$ holds. Without losing generality, assuming for vertex u the inequality holds, we change u from set S to $\bar{S}$, so the number of cross-cutting edges must be strictly decreasing. Since we have assumed that the number of edges in S or $\bar{S}$ is much greater than the number of cross-cutting edges, so $\Phi(G)$ must decrease according to the decrease of the number of cutting-edges, which leads to the contradiction of e_{uv} is a cross-cutting edge. □

Intuitively, theorem 1 gives us a clue that if two nodes have enough common neighbors, then we can *deterministically* say that the edge between them is non-cross-cutting. Moreover, (9) is tight - i.e., if it does not hold, then we can always construct a counter example where e_{uv} is cross-cutting - as shown in the following theorem.

Corollary 1. *For all $N(u), N(v), k_u, k_v$ which satisfy*

$$\left\lceil \frac{|N(u) \cap N(v)|}{2} \right\rceil + 1 \leq \frac{1}{2} \max\{k_u, k_v\}, \tag{10}$$

there always exists a graph $G(V, E)$ in which e_{uv} is cross-cutting.

Proof. Let $n = |N(u) \cap N(v)|$. We only need to construct a counter-example for each case that satisfies (10), but e_{uv} is a cross-cutting edge. Assume we have a graph like Fig 3, which shows the whole view of it. We let the number of common neighbors of node u and

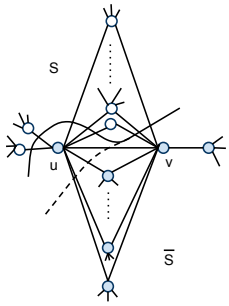


Figure 3: An counter example of lemma 1 that edge e_{uv} must be a cross-cutting edge.

v be n. Assuming $k_u \geq k_v$, from (10) we get:

$$\begin{cases} |O_u| = \max\{k_u, k_v\} - n - 1 \geq 1, & \text{if } n \text{ is even} \\ |O_u| = \max\{k_u, k_v\} - n - 1 \geq 2, & \text{if } n \text{ is odd} \end{cases} \tag{11}$$

Here $O_u = \{e_{wu} | w \in N(u) - N(v) \cup \{v\}\}$, which denotes the outer edges of u which is not linked to the node v and their common neighbors. We can carefully construct a graph like Fig 3: for each neighbor of node u and v, it only has 1-degree neighbors. So we need to prove that after assigning the degree for each node, e_{uv} will be a cross-cutting edge. If we simply let:

$$k_w \gg max\{k_u, k_v\} \;\; \forall w \in \{V - \{u, v\}\} \tag{12}$$

and then we divide these nodes into two sets S and $\bar{S}$.

Suppose n is even. In order to achieve the minimum in the definition of conductance, there must exist the case such that we only need to decide whether node u is in S or in $\bar{S}$.

$$\text{\# Cross-Cutting Edges} = \begin{cases} n + 1, & \text{if } u \in S \\ n + |O_u|, & \text{if } u \in \bar{S} \end{cases} \tag{13}$$

If $|O_u| > 1$, we can easily assert that e_{uv} is a cross-cutting edge. If $|O_u| = 1$, we can let $|a(S)| = |a(\bar{S})|$ when $u \in S$ to minimize $\min\{|a(S)|, |a(\bar{S})|\}$. So e_{uv} is an cross-cutting edge under this circumstance.

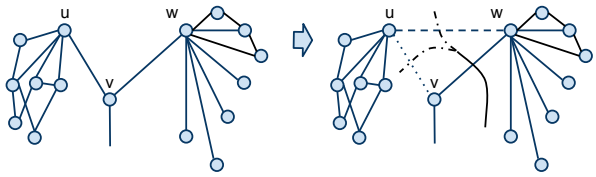


Figure 4: Replace the edge e_{uv} with e_{uw}

Also, suppose n is odd. Similarly, we have

$$\text{\# Cross-Cutting Edges} = \begin{cases} n+1, & \text{if } u \in S \\ 2\lfloor \frac{n}{2} \rfloor + |O_u|, & \text{if } u \in \bar{S} \end{cases} \tag{14}$$

Since $|O_u| \geq 2$, in the same way we know that e_{uv} is a cross-cutting edge. □

Running Example: With our on-the-fly edge removals, any random walk is essentially following an overlay topology G^* which can be constructed by applying Theorem 1 to every edge in the original graph G. For the bar-bell running example, the solid lines in Fig 1 depicts G^*. The conductance is now $\Phi(G^*) = 0.053$. Compared with the original conductance of 0.018, the corresponding lower bound on mixing time is reduced to 1638.3/14212.3 = 0.115 of the original value - a reduction of 89%.

4.2 Conditional Identification of Cross-cutting Edges

We now describe our second idea of conditionally identifying cross-cutting edges. We start with an example in Fig 4 to show why we can *replace* an existing edge with a new one such that (1) the new edge is more likely to be crosscutting, and (2) the replacement is guaranteed to *not* decrease the conductance.

Specifically, consider the replacement of e_{uv} by e_{uw} given the neighborhoods of u and v. A key observation here is that e_{uv} and e_{vw} cannot be both cross-cutting edges. The reason is that otherwise we could always "drag" v into the same partition as u and w to reduce the number of cross-cutting edges by at least 1. Given this key observation, one can

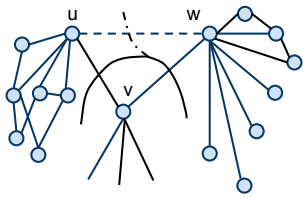


Figure 5: When replacing e_{uv} with e_{uw} from a node v whose degree is larger than 3, it may decrease the conductance if both e_{uv} and e_{wv} are cross-cutting edges.

see that the replacement of e_{uv} by e_{uw} will only have two possible outcomes:

- if e_{uv} is a cross-cutting edge, then e_{uw} must also be a cross-cutting edge because, due to the observation, e_{vw} cannot be a cross-cutting edge. Thus, the replacement leads to no change on the graph conductance.

- if e_{uv} is not a cross-cutting edge, then replacing it with e_{uw} will either keep the same conductance, or increase the conductance if e_{uw} is cross-cutting.

As such, the replacement operation never reduces the conductance, and might increase it when e_{uw} is cross-cutting. More generally, we have the following theorem.

Theorem 2. *Given $G(V, E)$, $\forall v \in V$, if $k_v = 3$, $u, w \in N(v)$, then replacing edge e_{uv} with e_{uw} will not decrease the conductance, while it also has positive possibility to increase the conductance.*

Proof. First, no matter e_{uv} is a cross-cutting edge or not, replace it with e_{uw} should at least obtain the same conductance. If e_{uv} is not a cross-cutting edge, then obviously we are not going to decrease the conductance because $a(S)$ or $a(\bar{S})$ will not change. If e_{uv} is a cross-cutting edge, we only need to prove that e_{uw} is also a cross-cutting edge. Let's assume e_{uw} is not a cross-cutting edge, then we can infer that the e_{vw} is a cross-cutting edge. But v only has degree of 3, so it is obvious that letting u, v and w be the same side will achieve less conductance, which contradicts the definition of conductance.

But if e_{vw} is a cross-cutting edge, and we replace e_{uv} with e_{uw}, then e_{uw} has the positive probability to become one more cross-cutting edge in this local view of u, v and w, which

result in higher conductance. □

Next, we are going to prove that $k_v = 3$ is actually the only case when replacement is guaranteed to not reduce the conductance, as shown by the following corollary.

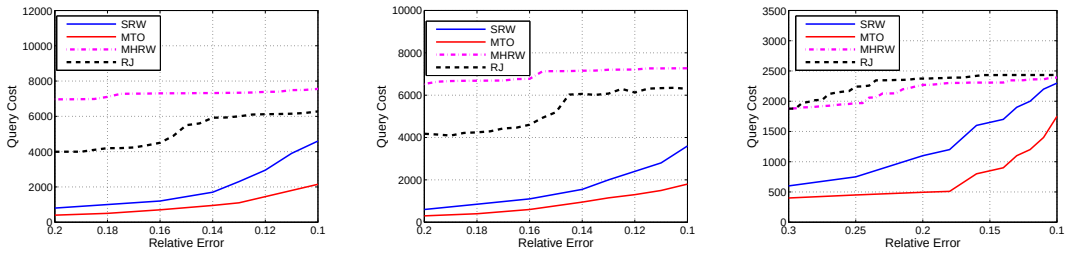
Corollary 2. *For $v \in V$, if $k_v \neq 3$, then there always exist a graph $G(V, E)$, $\forall u, w \in N(v)$, such that replacing e_{uv} with e_{uw} will decrease the conductance or have no effect.*

Proof. If $k_v = 1$, then we could not cut it to disconnect the graph. If $k_v = 2$, we need to check some possible situations. If none of these edges linked to v are cross-cutting edges, then replacing would not has effect on the conductance. If either e_{uv} or e_{wv} is a cross-cutting edge, then replace one of them with e_{uw} will not generate another cross-cutting edge; because now $k_v = 1$, and it should belongs to one side of the separation, S or $\bar{S}$.

So we only need to consider the situation when $k_v \geq 4$. See Fig 5. There exist the case when both e_{uv} and e_{wv} are cross-cutting edges. Then replacing e_{uv} with e_{wv} would decrease the number of cross-cutting edges from 2 to 1 locally, which may lead to dramatic decrease of the conductance of the graph.

The uniqueness of $k_v = 3$ is that there would not exist the case when both e_{uv} and e_{wv} are cross-cutting edges. □

Running Example: With Theorem 5.1.3, an example of the replacement operations one can perform over the bar-bell running example in Fig 1 is to replace e_{ur} with e_{rv}, given that u (after edge removals) has a degree of 3. Compared with the original conductance of $\Phi(G)$ = 0.018 and the post-removal conductance of $\Phi(G^*)$ = 0.053, the conductance is now further increased to $\Phi(G^{**}) = 0.105$. The corresponding lower bound on mixing time is reduced to 416.6/1638.3 = 0.25 of the post-removal bound - a further reduction of 75% - and 416.6/14212.3 = 0.029 of the original bound - an overall reduction of 97%.



(a) Slashdot A (b) Slashdot B (c) Epinions

Figure 6: Bias vs. Query Cost tests for local datasets' average degree.

4.3 Experiments

We started by comparing the performance of Simple Random Walk (SRW) and MTO-Sampler over real-world social networks using all three performance measures described above - KL-divergence, relative error vs. query cost, and theoretical mixing time.

Local Datasets: We started by testing the relative error vs. query cost tradeoff of SRW, MTO, MHRW and RJ for estimating aggregate query answers. Since only topological information is available for local datasets, we used the average degree as the aggregate query. Fig 6 depicts the performance comparison for the three real-world social networks. Here each point represents the average of 20 runs of each algorithm, and the query cost (i.e., y-axis) represents the maximum query cost for a random walk to generate an estimation with relative error above a given value (i.e., x-axis). For random jump in the experiments, we set the probability of jumping as 0.5. One can see that, for all three datasets, our MTO-Sampler achieves a significant reduction of query cost compared with the SRW sampler, MHRW sampler and Random Jump sampler.

We also tested the KL-divergence measured by performing an extremely long execution of SRW and MTO in Fig 7 - with each producing 20000 samples - to estimate the sampling probability for each node. The Geweke threshold was set to be 0.1 for the test. One can see that our MTO-Sampler not only requires fewer queries for generating each sample (i.e., converges to the stationary distribution faster), but also produces less bias than the SRW

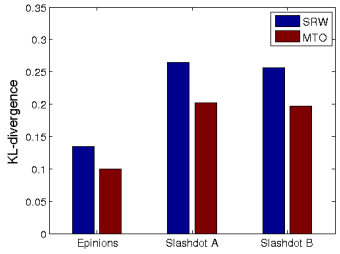


Figure 7: Comparison between SRW and MTO on query cost and the KL divergence measure over all three datasets.

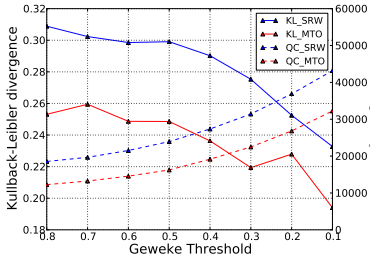


Figure 8: Varying Geweke Threshold to get different KL divergence on dataset Slashdot B. KL and QC stands for KL divergence and Query Cost respectively.

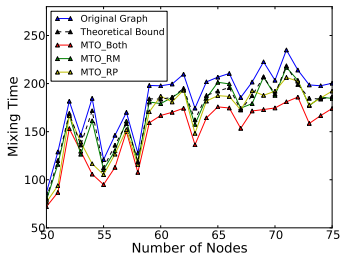


Figure 9: Theoretical mixing time on latent space graph model. MTO_Both: Remove and replace edges. MTO_RM: Remove edges. MTO_RP: Replace edges.

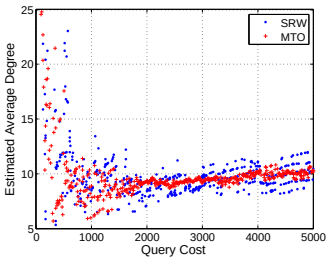
sampler.

To further test the bias of samples generated by our MTO-Sampler, we also conducted the test while varying the Geweke threshold from 0.1 to 0.8 on the dataset Slashdot B. Fig 8 depicts the change of measured bias for SRW and MTO, respectively. One can see from the figure that our MTO-Sampler achieves smaller bias than SRW for all cases being tested. In addition, a smaller threshold leads to a smaller bias and larger query cost, as indicated by the definition of Geweke convergence monitor.

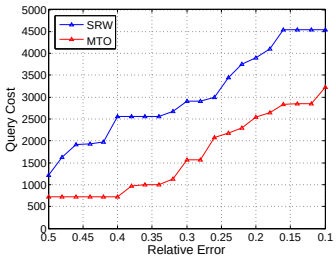
Google Plus online social network: For Google Plus, we do not have the ground truth as the entire social network is too large (about 85.2 million users in Feb 2012[9]) to be crawled. Thus, we performed the tests in two steps. First, we continuously ran each sampler until their Geweke convergence monitor indicated that it had reached its stationary distribution. We then used the final estimation as the presumptive ground truth which we refer to as the *converged value*. In the second step, we used the converged value to compute the relative error vs. query cost tradeoff as previously described.

Fig 10(a) shows the estimated average degree when running SRW and MTO-Sampler random walk on Google Plus. It clearly shows that MTO-Sampler's variance is smaller and converges faster than simpler random walk. Fig 10(b) and 10(c) illustrate the comparison

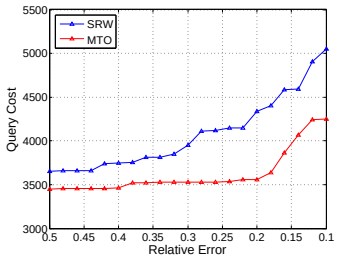
[9]Estimated by Paul Allen's model, http://goo.gl/nZCzN



(a) Estimated Average Degree



(b) Average Degree



(c) Average Self-description Length

Figure 10: Google Plus online social network

between SRW and MTO of the relative error vs query cost of multiple attributes. We note that the self-description length is the number of characters in users' self-description. One can see that our MTO-Sampler significantly outperforms SRW.

Synthetic Social Networks: Finally, we conducted further analysis of our MTO-Sampler, in particular the individual effects of edge removals (RM) and edge replacements (RP), using the synthetic latent space model. Fig 9 depicts the results when the number of nodes in the graph varies from 50 to 100 (with the latent space model, we distributed these nodes in an area of $[0, 4] \times [0, 5]$, and set $r = 0.7$). We derived the theoretical mixing time from the second largest eigenvalue modulus of the transition matrix. One can see from the figure that our final MTO-Sampler achieves better efficiency than the individual applications of edge removal and replacement. In addition, the theoretical model represents a conservation estimation that is outperformed by the real efficiency of MTO-Sampler.

Chapter 5 - Extension of MTO-SRW: MTO-MHRW and MTO-GRW

5.1 Key Ideas for Rewiring a Metropolis-Hastings Random Walk (MTO-MHRW)

In this section, we introduce our main ideas for rewriting a Metropolis-Hastings Random Walk (MHRW) on-the-fly. Specifically, we start by introducing the concept of graph *conductance* - a key efficiency indicator - for MHRW in Section 5.1.1. Then, in Section 5.1.1, we explain that conductance for a MHRW is mainly determined by the number of *cross-cutting* and *margin non-cross-cutting (defined in Section 5.1.1)* edges in the graph. Thus, to increase graph conductance and enable a more efficient random walk, we should aim to increase the number of cross-cutting edges and reduce the number of margin non-cross-cutting ones in the overlay graph.

5.1.1 Conductance of MHRW

Recall from the introduction that graph *conductance* is a key efficiency indicator for random walks. We formally define the conductance of MHRW as follows.

Definition 6. ***(MHRW Conductance)****. The conductance of a MHRW on the graph* $G : \langle V, E\rangle$ *is*

$$\begin{aligned}\Phi^{MH}(G) &= \min_{S\subset V} \frac{\sum_{v\in S, w\notin S} \pi(v)P_{v,w}}{\min\left\{\pi(S), \pi(\bar{S})\right\}} \\ &= \min_{S\subset V} \frac{\sum_{v\in S, w\notin S} \min\{\frac{1}{k_v}, \frac{1}{k_w}\}}{\min\left\{|S|, |\bar{S}|\right\}}\end{aligned} \tag{15}$$

Noted that the definition of conductance (15) is different from that of the SRW: MHRW chooses the next edge in a *non-uniform* manner. This makes the MTO-Sampling over MHRW (MTO-MHRW) a more complicated problem for two reasons. First, it is harder to identify non-cross-cutting edges in MHRW since the probability of choosing the next edge amongst the candidates is not the same. Second, we cannot guarantee that removing the non-cross-cutting edges will definitely improve the conductance in MHRW. A key obser-

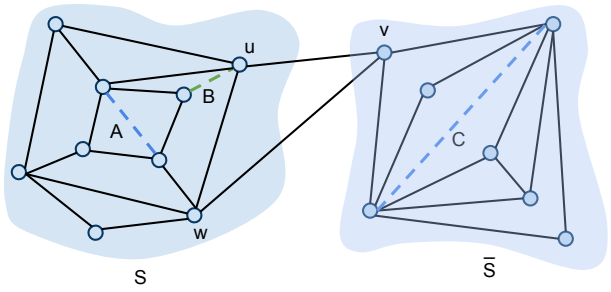


Figure 11: An example of the edge categories in rewiring MHRW. Edge e_{uv}, e_{wv} are cross-cutting edges. Edge B is a margin non-cross-cutting edge that is associated with an end of a cross-cutting edge. Edge A and edge C are inner non-cross-cutting edges that are not connected to any end of the cross-cutting edges.

vation from Definition 6 is that MHRW conductance critically depends on the degree of the nodes that are at the margin of S or $\bar{S}$. In order to increase the conductance of MHRW, we will first introduce three types of edges in MHRW, and then discuss the strategy of rewiring these edges.

We define three types of edges in MHRW: *cross-cutting edges, margin non-cross-cutting edges and inner non-cross-cutting edges*. Cross-cutting edges are those edges that are crossed by the optimum cut (optimum partition S and $\bar{S}$ in (15)). Others are non-cross-cutting edges including margin non-cross-cutting edges and inner non-crdoss-cutting edges. An edge is a margin non-cross-cutting edge when it is connected to a cross-cutting edge, and inner non-cross-cutting edges have no connections with any of the cross-cutting edges. For instance, in Fig 11, edge e_{uv}, e_{wv} are cross-cutting edges, edge A and C are inner non-cross-cutting edges, and edge B is an margin non-cross-cutting edge.

When we identify a non-cross-cutting edge and remove it from the graph, chances are the conductance will increase if it is a margin non-cross-cutting edge otherwise the conductance remains the same. By definition (e.g. equation (15)), removing a margin non-cross-cutting edge will improve the conductance by increasing the weight of cross-cutting edges in the graph. However, removing inner non-cross-cutting edges like A and C will not change the conductance of MHRW, because $|S|$ and $|\bar{S}|$ in the denominator of (15) will not change when we modify the topology of the graph.

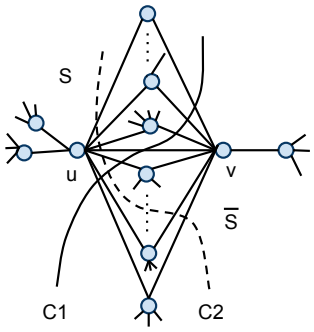


Figure 12: In Metropolis-Hastings Random Walk, if e_{uv} is a cross-cutting edge, we may need to check every possible separation of S and $\bar{S}$ to determine whether e_{uv} can be removed or not.

5.1.2 Deterministic Identification of non-cross-cutting edges in MHRW

We illustrate how the deterministic identification works (see Fig 12) by showing that it will lead to contradiction if we assume that e_{uv} is an cross-cutting edge.

Applying the same idea we have discussed in Simple Random Walk. For every cut that separates u and v, if we modify the cut (i.e. make another configuration of S and $\bar{S}$), and this modification reduce the conductance, thus contradict the definition of conductance. For example in Fig. 12, e_{uv} is a cross-cutting edge, and $C1$ and $C2$ are two possible cuts. If we can construct another configuration of S and $\bar{S}$ that has lower conductance and e_{uv} are not separated, we can guarantee that e_{uv} is not a cross-cutting edge.

Let $N = N(u) \cap N(v)$, then the problem can be described as: for every subset $P \subseteq N$, can we guarantee that one of the following inequality holds:

$$\sum_{p \in P} \omega_{pu} > \frac{1}{2} \sum_{r \in N(u)} \omega_{ru} \tag{16}$$

$$\sum_{q \in (N-P)} \omega_{qv} > \frac{1}{2} \sum_{s \in N(v)} \omega_{sv} \tag{17}$$

It is trivial to notice that for every subset P, if one of the above two inequalities holds, we can simply "drag" u or v from one side to the other, which will generate lower conductance and contradict with the definition of cross-cutting edge because we are "flipping" the edge

around to change cross-cutting edge into non-cross-cutting edge and vice versa. This is similar to the techniques we have discussed in SRW.

One observation is that if $\forall w \in N$, $\omega_{wu} = \omega_{wv}$, then the inequality can be transformed to:

$$\sum_{p \in P} \omega_{pu} > \frac{1}{2} \cdot \min \left\{ \sum_{r \in N(u)} \omega_{ru}, \sum_{s \in N(v)} \omega_{sv} \right\}. \tag{18}$$

This is similar to the *balanced partition problem*. Given a set of weights of the edges connecting u and v: $\Omega = \{\omega_{uw_1}, \omega_{uw_2}, \ldots, \omega_{uw_n}\}$ ($0 \leq \omega_{uw_i} \leq 1, \forall i \in \{1 \ldots n\}$), we can find a subset $\Omega_0 \subseteq \Omega$ to minimize

$$|\sum_{\omega \in \Omega_0} \omega - \sum_{\nu \in \bar{\Omega}_0} \nu|, \tag{19}$$

here we denote $\bar{\Omega}_0 = \Omega - \Omega_0$. If

$$\min \left\{ \sum_{\omega \in \Omega_0} \omega, \sum_{\nu \in (\bar{\Omega}_0)} \nu \right\} > \frac{1}{2} \cdot \min \left\{ \sum_{r \in N(u)} \omega_{ru}, \sum_{s \in N(v)} \omega_{sv} \right\}, \tag{20}$$

then we can always assert that e_{uv} is a non-cross-cutting edge.

For every possible cut that separates S and $\bar{S}$, we can "drag" u or v, which flips the cross-cutting edges connected to u or v to non-cross-cutting edges and vice versa. Therefore, for every configuration (e.g. every subset $P \subseteq N(u) \cap N(v)$) we construct a new configuration with lower conductance. We summarize the former discussion as a theorem:

Theorem 3. *Given a uniform stationary distribution MHRW, and an edge e_{uv}, for all sub-*

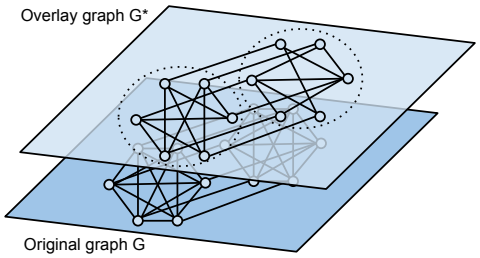


Figure 13: Example of modified Graph for MHRW. Graph G is a graph with multiple edges crossing two complete graphs (multiple bridges bar-bell graph), and each of them has 6 nodes. Graph G^* is a graph modified by Theorem 3.

sets $P \subset N(u) \cap N(v)$*, if either*

$$\sum_{p \in P} \omega_{pu} > \frac{1}{2} \sum_{r \in N(u)} \omega_{ru} \tag{21}$$

or

$$\sum_{q \in (N-P)} \omega_{qv} > \frac{1}{2} \sum_{s \in N(v)} \omega_{sv} \tag{22}$$

holds, then e_{uv} *is not a cross-cutting-edge. Here* $\omega_{uv} = \min\{\frac{1}{k_u}, \frac{1}{k_v}\}$.

MHRW Example in Fig 13: With our on-the-fly edge removals, a Metropolis-Hastings random walk is essentially following an overlay topology G^* that can be constructed by applying Theorem 3 to every edge in the original graph G. For a multiple bridges bar-bell graph, the solid lines in Fig 13 depicts G^*. Recall that the mixing time derived from equation (15), we calculated that $\Phi(G) = 1.0$. And after applying Theorem 3, we got $\Phi(G) = 1.4$, which yielded a 98% speedup of the mixing-time from 35.497 to 17.862.

However, there are two challenges when we remove edges. First, solving the balanced partition problem for an arbitrary real number is NP-complete. It is almost impossible to determine whether an edge can be removed in MHRW if the number of common neighbors is large. Trivial solution is to set a threshold, and only to consider those two nodes with

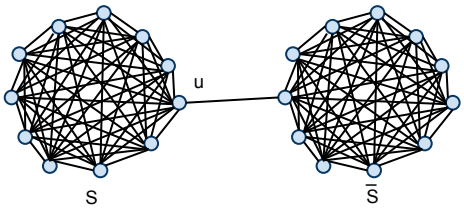


Figure 14: A barbell graph example that demonstrates the starting node selection in MTO-MHRW

the number of common neighbors less than the threshold. Although we leave some pair of nodes untouched, the point is if we do not modify the topology, we are not going to reduce the conductance, but removing edges will possibly increase the conductance.

(1) Only a part of the removed edges that have impact on the conductance of MHRW. One can dramatically improve the performance if it found margin non-cross-cutting edges early in the MTO-MHRW. By definition (i.e. equation (15)), removing margin non-cross-cutting edges will increase the conductance by increase the weight of cross-cutting edges in the graph, while removing inner non-cross-cutting edges will not change the conductance of MHRW. Practically, the improvement of the conductance highly depends on the percentage of the edges associated with cross-cutting edges and the selection of the initial nodes. If the percentage is small, or the initial node is far from the cross-cutting edges, then it is hard for the MHRW to increase the conductance. We will discuss the experimental results in section 5.3.1.

(2) Once MTO-MHRW finds a margin cross-cutting-edge, it has a higher probability to find another when compared with MTO-SRW. This will improve the performance even more if one start with a node near margin cross-cutting-edges. To illustrate the idea of it, we start with a barbell graph example, see Fig. 14. If we start with node u, we can easily verify that every edge associated with u in the left cluster S is a margin non-cross-cutting edge. Without losing generality, we assume that the next step of the MTO-MHRW will not cross edge e_{uv}, because S and $\bar{S}$ are symmetric, and node v is equivalent to u. Therefore, any attempt to check edge $e_{pu}(\forall p \in S)$ will lead to the conclusion that e_{pu} is removable

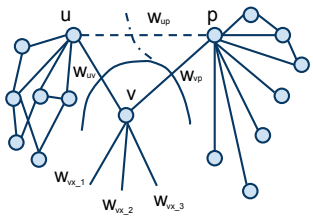


Figure 15: Replacing e_{uv} with e_{uw} from a node v in MHRW

(according to Theorem 3). This removing process can be repeated k times (until it fail to pass the condition in Theorem 3) because we are still at the node u, moreover, each time we remove an edge associated with u, we are increasing the probability of staying at node u. Thus, we have a higher probability to find another margin non-cross-cutting edge.

5.1.3 Conditional identification of non-cross-cutting edge in MHRW

Inspired by the idea of conditional identification of non-cross-cutting edge in SRW, the goal of it in MHRW is similar: (1) the new edge being replaced is more likely to be cross-cutting, and (2) the replacement is guaranteed not to *not* decrease the conductance.

We will demonstrate the idea in the following example of Fig 15. The difference between SRW and MHRW is that each edge of MHRW is assigned with an "weight" ω_{uv}, thus we may not only consider how many edges are connected to node v (like the constraint $k_v = 3$ in theorem), but also the ergodic flow of it. Assume the MHRW encounters with node v, and we want to replace edge e_{uv} with e_{up}. Also if

$$\sum_{x \in (N(v) - \{u\})} \omega_{vx} < \omega_{uv} + \omega_{vp} \tag{23}$$

holds, here $\omega_{uv} = \min\{\frac{1}{k_u}, \frac{1}{k_v}\}$, then we have the following observations:

1. Edge e_{uv} and e_{vp} cannot be cross-cutting edge at the same time. Simply, if we assume they are both cross-cutting edges, then we can easily "drag" v to the other side to make a less ergodic flow between the new configuration S and $\bar{S}$ to reduce the

conductance, thus it contradicts to the definition of conductance.

2. If e_{uv} is a cross-cutting edge, then after replacing e_{uv} with e_{up}, e_{up} must be the new cross-cutting edge. Since we have the knowledge of ω_{uv} and ω_{up}, and when

$$\omega_{up} > \omega_{uv}, \tag{24}$$

we will replace the edge e_{uv} with e_{up}.

3. If e_{vp} is a cross-cutting edge, then after the replacement we will add another cross-cutting edge e_{up}, which definitely increase the conductance.

4. If e_{uv} and e_{vp} are both non-cross-cutting edges, we don't need to worry about decreasing the conductance, since in MHRW, we can never change the conductance by manipulating the edges completely within (the edge's nodes are not associate with cross-cutting edges) S or $\bar{S}$.

Therefore, we can safely replace edge e_{uv} with e_{up} under the condition of (23) and (24) without decreasing the conductance. The benefit is that when e_{uv} or e_{up} is cross-cutting-edges, we can increase the conductance *conditionally*. We summarize the former discussion as a theorem:

Theorem 4. *Given a uniform stationary distribution MHRW on graph $G(V, E)$, for three nodes $u, v \in V$ and $e_{uv}, e_{vp} \in E$, and if $\sum_{x \in (N(v)-\{u\})} \omega_{vx} < \omega_{uv} + \omega_{vp}$ and $\omega_{up} > \omega_{uv}$, then replacing e_{uv} with e_{up} will not decrease the conductance, while it also has positive possibility to increase the conductance.*

5.2 Key Ideas for Rewiring a General Random Walk (MTO-GRW)

In this section, we introduce our main ideas for rewriting a General Random Walk (GRW) on-the-fly. Specifically, we start by introducing the concept of graph *conductance* - a key efficiency indicator - for GRW in Section 5.2.1. In Section 5.2.1 we also explain that

conductance for a GRW is mainly determined by the ergodic flow between the partitions of the graph. As we show in Section 5.2.2, there are two things we can do to achieve a higher conductance in GRW: we can deterministically identify an edge as non-cross-cutting and therefore remove it from the overlay graph, thus we can potentially remove margin non-cross-cutting edges to increase the conductance; and we can conditionally identify an edge as cross-cutting and therefore insert it into the overlay graph to increase the conductance.

5.2.1 Conductance of GRW

We consider general random walks (GRW) that are *time-reversible* random walks. A random walk is time-reversible if and only if:

$$\pi_u P_{uv} = \pi_v P_{vu}, \forall u, v \in V. \tag{25}$$

This is equivalent to the time-reversible Markov Chain. Obviously for time-reversible random walks,

$$Q_{uv} \triangleq \pi_u P_{uv} = \pi_v P_{vu} = Q_{uv}. \tag{26}$$

We also denote the *ergodic flow* between two sets as

$$Q(S_1, S_2) = \sum_{u \in S_1, v \in S_2} Q_{uv}. \tag{27}$$

Definition 7. ***(General Random Walk Conductance)****. Given a graph $G(V, E)$, and a GRW's transition matrix P, the conductance of the GRW on this graph is*

$$\Phi(G, P) = \min_{S \subset V} \frac{Q(S, \bar{S})}{\min\{\pi(S), \pi(\bar{S})\}} \tag{28}$$

The general conductance is determined by both the graph topology and the transition matrix of the random walk. Intuitively, the conductance Φ, which indicates how fast the simple random walk converges to its stationary distribution, measures how "well-knit" a

graph is. Specifically, the conductance is mainly determined by a *optimum ergodic flow* of the graph G - i.e., a minimum ergodic flow $Q(S, \bar{S})$ in (28) - which indicates the probability for the random walk to move from one partition to the other. One can expect that the minimum occurs when $\pi(S) \approx \pi(\bar{S})$.

5.2.2 Deterministic and conditional identification of non-cross-cutting edges in general random walks

To illustrate the general extension of the MTO-SRW and MTO-MHRW, we demonstrate the deterministic and conditional identification using ergodic flow of the GRW, and we show that one can still rewire the topology of the graph to get speed-ups on mixing time.

Theorem 5. *For any node pair $u, v \in V$, if for all subsets $R \subset N(u) \cap N(v)$, either*

$$Q(u, R) > \frac{1}{2} Q_u \tag{29}$$

or

$$Q(v, N(u) \cap N(v) - R) > \frac{1}{2} Q_v \tag{30}$$

holds, then edge e_{uv} is not a cross-cutting-edge. Here $Q_u = \sum_{i \in N(u)} Q(u, i)$.

Proof. Suppose e_{uv} is a cross-cutting-edge, we know that for any subset R of u and v's common neighbors, if either of these two inequality holds, which means we can drag either u or v to the other side (e.g. from S to $\bar{S}$ and vice versa) to get a smaller ergodic flow between S and $\bar{S}$, then we can claim that e_{uv} is not a cross-cutting edge, which contradicts to the assumption. □

By the definition of conductance of SRW and MHRW, we are sure that removing non-cross-cutting edges will definitely reduce the number of edges in S or $\bar{S}$ in SRW and increase the probability of MHRW moving from S to $\bar{S}$ (and vice versa, see definition (15)).

Therefore, removing non-cross-cutting edges in SRW and MHRW will meet our goal of improving the conductance. However, this is not the case when it comes to GRW. Intuitively, we want to remove the non-cross-cutting edges once we have deterministically identified them to reduce the probability of them staying in S or $\bar{S}$. But we are not sure that removing a non-cross-cutting will improve the conductance of GRW. The problem with removing an edge in GRW is that we don't even know how the edge removal procedure will affect the conductance without any knowledge of the detailed algorithm of the random walk.

The problem can be solved by assuming that the random walk designer give out the details of the conductance so that MTO-Sampling could decide whether it is appropriate to remove certain non-cross-cutting edge. In other words, MTO-Sampling can work as expected as long as the GRW designer can answer the question "Will removing this non-cross-cutting edge increase $\Phi(G, P)$?" whenever MTO-Sampling identifies a non-cross-cutting edge.

The conditional identification here is also similar to MHRW. We can set the weight of the edge ω_{uv} as the ergodic flow Q_{uv}.

Theorem 6. *Given a general random walk on graph $G(V, E)$, for three nodes $u, v \in V$ and $e_{uv}, e_{vp} \in E$, and if $\sum_{x \in (N(v)-\{u\})} Q_{vx} < Q_{uv} + Q_{vp}$ and $Q_{up} > Q_{uv}$, then replacing e_{uv} with e_{up} will not reduce the conductance, while it also has positive possibility of increasing the conductance.*

5.3 Experiments

5.3.1 Mixing time comparison between MHRW and MTO-MHRW

We start MHRW and MTO-MHRW from a node (i.e. the initial node for MHRW and MTO-MHRW), and record their SLEM mixing time. The speedup is calculated as:

$$\text{Speedup} = \frac{T - T_{\text{MTO}}}{T}, \tag{31}$$

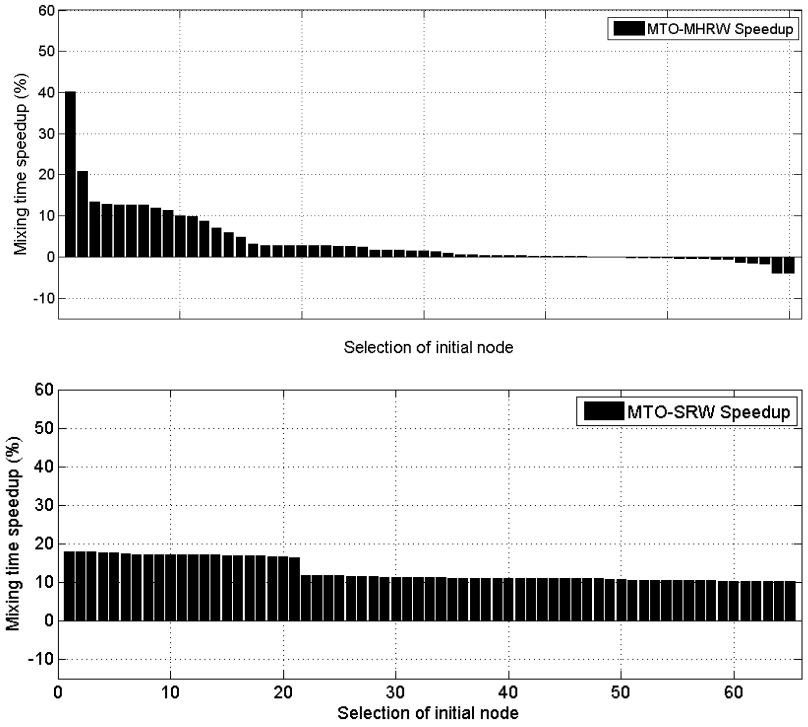


Figure 16: Mixing time speedup distribution of initial nodes for MHRW and SRW in a subgraph of Facebook dataset.

where T is the SLEM mixing time $1/\log(1/\mu)$. Fig 16 depicts the speedup distribution of arbitrary chosen starting points (same initial nodes set for MHRW and MTO-MHRW) from the Facebook-small dataset. Starting from about 20% them, we got over 10% speedup of SLEM mixing time. And if we are lucky to start from a node like the first one in Fig 16, then we gain a speedup over 40%. Others don't make significant improvement due to the constraints arising from the intrinsic property of MHRW: few removal edges are capable to affect the conductance, while almost all the removal edges can do so for MTO-SRW. When compared with the performance of SRW, the selection of the initial node has much less impact on the performance gain of MTO-SRW algorithm.

Chapter 6 - WALK-ESTIMATE (WE) Sampler

6.1 Key Idea of WE Sampler

Our objective is to significantly reduce the query cost of node sampling over an online social network by (nearly) eliminating the costly waiting process (mixing time). Of course, as one can see from the above discussions, if we do not wait for the convergence to stationary distribution, we must somehow estimate the probability for our short walk to take a node as a sample (i.e., the node's *sampling probability*) before we can use the node as a sample. This is exactly what we do - i.e., we introduce a novel idea of having a (much) shorter, say t-step, walk, before taking a node v as a sample candidate, but follow it up with a *proactive* process which estimates v's sampling probability $p_t(v)$ - i.e., the probability for our walk to reach node v at Step t, so that we can then use acceptance-rejection sampling to "correct" the sampling probability to the desired distribution. As we shall prove in the paper, even though the acceptance-rejection step introduces additional query cost, the savings from having a shorter walk in the first place far outweighs the additional cost, leading to a significantly more efficient sampling process.

Based on this idea, we develop Algorithm WALK-ESTIMATE. The algorithm takes as input a random walk based MCMC sampler, and produces samples according to the exact same distribution as the input sampler - i.e., the stationary distribution of the MCMC process. One can see that this design makes WALK-ESTIMATE transparent to the desired target distribution, making it a swap-in replacement for any random walk sampler being used (e.g., simple random walk [22, 24], Metropolis-Hastings random walk [22, 24]. As we shall demonstrate through theoretical analysis and experimental evaluation over real-world social networks, while the proactive probability-estimation process may consume a small number of queries, the significant savings from the shorter walk more than offset the additional consumption, and lead to a much more efficient sampling process over online social networks.

6.2 WALK

We start with developing Algorithm WALK which significantly improves the efficiency of sampling by having a much shorter random walk followed by a rejection sampling process. Note that, for the ease of discussions, we separate out the discussion of sampling-probability estimation to the ESTIMATE component discussed in the next section - i.e., Algorithm WALK calls upon Algorithm ESTIMATE as a subroutine.

In this section, we first illustrate the key rationale behind our design with an ideal-case algorithm, IDEAL-WALK, and then present theoretical analysis which shows that a shorter walk followed by an acceptance-rejection procedure can almost always outperform the traditional random walk, no matter what the starting point is or the graph topology looks like. To study how much improvement a short walk can offer, we describe case studies with the underlying graph generated from various theoretical graph models. Finally, we conclude this section with the practical design of Algorithm WALK.

6.2.1 IDEAL-WALK: Main Idea and Analysis

The key rationale behind our idea of performing a short walk followed by acceptance-rejection sampling can be stated as follows. Recall from Section 2 that the long walk is required by traditional random walks to reduce the "distance" between its sampling distribution and the target, stationary, distribution - a distance often measured by the difference (e.g., ℓ_∞-norm) between the two probability vectors.

Consider how such a difference changes as the walk becomes longer. When the walk first starts, the sampling distribution is extremely skewed - i.e., $p_1(v) = 1$ on one node (the starting one) and 0 on all others - leading to an extremely large distance. As the walk proceeds, the distance decreases quickly - for example, as long as the walk length exceeds the graph diameter, all values in the sampling probability vector become positive[10], while

[10]Note that here we assume each node has a nonzero (can be arbitrarily small) probability to transit to itself, to eliminate trivial cases where the graph is not irreducible.

the maximum value in the vector tends to decrease exponentially with the (initial few) steps taken - leading to a sharp decrease of the distance.

Nonetheless, it is important to note that the speed of reduction on the distance becomes *much slower* as the random walk grows longer. A simple evidence is the asymptotic nature of burn-in as discussed in Section 2 - which shows that, for some graphs, the ultimate reduction to zero distance never completes with a finite number of steps. Figure 17 demonstrates a concrete example for a random scale free network with 31 nodes generated by the Barabasi-Albert model [6], where number of edges to attach from a new node to existing nodes is 3. One can see from the figure that the speed of reduction declines sharply once the random walk grows longer than the graph diameter. In summary, one can observe the following "behavior pattern" of traditional random walks: to achieve a preset goal of shrinking the distance measure below a threshold, the random walk makes significant progress in the first few steps. Nonetheless, the "benefit-cost ratio" diminishes quickly as the random walk continues. As a result, a random walk might require a very long burn-in period to achieve the preset distance threshold.

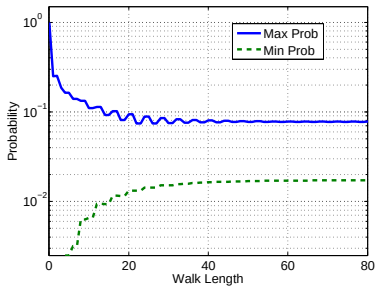


Figure 17: Minimum and maximum probabilities vs walk length

Standing in sharp contrast to the above described behavior pattern is the performance of using acceptance-rejection sampling to achieve the pre-determined target distribution (instead of waiting for convergence). Interestingly, applying rejection sampling at the beginning of a random walk is often extremely costly - or even outright infeasible. For example, no rejection sampling can correct to a uniform target distribution before the walk is at least as long as the graph diameter. On the other hand, as the walk becomes longer, the cost of

applying rejection sampling to reach the target distribution becomes much smaller - as we shall demonstrate as follows.

Consider an example where the target distribution is uniform. Note from acceptance-rejection sampling in Section 2.6 that, in this case, the cost of rejection sampling is simply determined by (to be exact, inversely proportional to) the minimum value in the input sampling distribution - as the probability of accepting a sample is exactly the minimum probability multiplied by the number of nodes in the graph. As discussed above, this minimum probability grows from 0 at the very beginning to a positive value when the walk reaches the diameter, and often increases rapidly at the initial stage of random walk (see again Figure 17). Correspondingly, the cost of rejection sampling drops significantly with a longer random walk.

One can observe from the above discussion an interesting distinction between two (competing) methods, (a) wait for the sampling distribution to converge to the stationary one, and (b) taking the current sampling distribution and directly "correct" it through rejection sampling: These two methods are better applied at different stages of a random walk process. Specifically, at the very beginning, method (b) is extremely costly or outright infeasible - so we should follow method (a) and walk longer for the sampling distribution to grow closer to the target vector. Nonetheless, after a certain number of steps, the direct correction (i.e., method (b)) becomes the better option because of the slower and slower convergence speed. Therefore, we should stop waiting for further convergence, and instead use rejection sampling to directly reach the desired distribution.

Of course, the above discussions leaves an important question unanswered: Given a reasonable threshold on the distance (between achieved sampling distribution and the desired stationary one), is there always a tipping point where we switch for waiting to correction? Note that the reason why the threshold value is important here can be observed from the extreme cases: when the threshold is extremely large, there is no need to switch because even the initial (one 1 and all other 0) distribution already satisfies the threshold. On the other

hand, when the threshold trends to 0, as we discussed above, there are graphs for which the convergence length tends to infinity - i.e., it is always better to switch to rejection sampling as long as it has a finite cost. One can see from the extreme cases that whether switching to rejection sampling is effective in practice depends on whether the switch is necessary for *reasonable* thresholds that are just small enough to support real-world applications over the samples taken from online social networks. To this end, we have the following theorem.

Theorem 7. *Given an input random walk which has a transition design with spectral gap λ, to guarantee an ℓ_∞-variation distance of Δ between the sampling and target distributions, the expected query cost per sample of IDEAL-WALK which performs the random walk for t_{opt} steps followed by rejection sampling, where*

$$t_{\text{opt}} = \frac{-\log(-2|E| \cdot W(-\frac{1}{2e|E|d_{\max}}) \cdot d_{\max})}{\log(1-\lambda)}, \tag{32}$$

where $d_{\max}$ is the maximum degree of all nodes in the graph and W is the Lambert W-function, is always smaller than that of the the input random walk as long as $0 < \Delta < 1/(2|E|)$. Specifically, the ratio between the query cost per sample of IDEAL-WALK and the input random walk is at most

$$\begin{aligned} &\frac{t_{\text{opt}} - 2t_{\text{opt}} \cdot |E| \cdot \Delta}{1 - 2|E| \cdot (1-\lambda)^{t_{\text{opt}}} \cdot d_{\max}} \\ \leq &\frac{-\log(-2|E| \cdot W(-\frac{1}{2e|E|d_{\max}}) \cdot d_{\max})}{\log(\Delta/d_{\max})} \cdot \frac{1 - 2|E| \cdot \Delta}{1 + \frac{1}{W(-\frac{1}{2e|E|d_{\max}})}}. \end{aligned} \tag{33}$$

Proof. According to the ℓ_∞-norm mixing time of a Markov chain, if the random walk starts from v, we have a tight bound (tight in the worst-case scenario [41])

$$|p_t(u) - p(u)| \leq (1-\lambda)^t \cdot \deg(v) \tag{34}$$

As such, to guarantee an ℓ_∞ variation distance of Δ, the probability for rejection sam-

pling to accept a sample taken after a walk of t steps is at least

$$\omega \geq \frac{1/(2|E|) - (1-\lambda)^t \cdot \deg(v)}{1/(2|E|) - \Delta} \tag{35}$$

$$= \frac{1 - 2|E| \cdot (1-\lambda)^t \cdot d_{\max}}{1 - 2|E| \cdot \Delta}. \tag{36}$$

Thus, in order to guarantee an ℓ_∞ variation distance of Δ in the worst-case scenario, the expected query cost per sample achieved by IDEAL-WALK is at most

$$c \leq \min_{t:t>0} \frac{t - 2t \cdot |E| \cdot \Delta}{1 - 2|E| \cdot (1-\lambda)^t \cdot d_{\max}}; \tag{37}$$

while the expected query cost per sample achieved by the input random walk is

$$c_{\mathrm{RW}} = \frac{\log(\Delta/d_{\max})}{\log(1-\lambda)}. \tag{38}$$

Note that for any Δ, we always have $c \leq c_{\mathrm{RW}}$ because when $t = c_{\mathrm{RW}}$, we have

$$\frac{t - 2t \cdot |E| \cdot \Delta}{1 - 2|E| \cdot (1-\lambda)^t \cdot d_{\max}} = c_{\mathrm{RW}}. \tag{39}$$

Intuitively, this simply means that by running IDEAL-WALK for as long as the input random walk (and therefore being able to skip rejection sampling), IDEAL-WALK is essentially reduced to the input random walk. Our task now is to determine whether $c < c_{\mathrm{RW}}$. We shall start with an intuitive discussion of why c tends to be much smaller than c_{RW} for almost all realistic requirements of Δ, and then present the formal analysis. Intuitively, consider the case where Δ is reduced by half, to $\Delta/2$. Note from (38) that the change of c_{RW} is always constant no matter how large (or small) Δ is - i.e.,c_{RW} will increase by $-\log 2/\log(1-\lambda)$. On the other hand, note from (37) that the change of c is not always constant. Instead, the increase of c is at most $t \cdot |E| \cdot \Delta/(1 - 2|E| \cdot (1-\lambda)^t \cdot d_{\max})$ for t which minimizes (37) for the original value of Δ. Note that as Δ becomes smaller, t either

stays the same or becomes larger. As such, the smaller Δ is, the smaller the increase of c will be. This is the intuitive explanation of why c tends to be much smaller than c_{RW} when Δ is reasonably small.

Formally, we start with determining the optimal value of t that minimizes (37). Let

$$f(t) = \frac{t - 2t \cdot |E| \cdot \Delta}{1 - 2|E| \cdot (1-\lambda)^t \cdot d_{\max}}. \tag{40}$$

To satisfy $df(t)/dt = 0$, we have

$$t = \frac{2|E| \cdot (1-\lambda)^t \cdot d_{\max} - 1}{2\log(1-\lambda) \cdot |E| \cdot (1-\lambda)^t \cdot d_{\max}} \tag{41}$$

$$= \frac{1}{\log(1-\lambda)} \left(1 - \frac{1}{2|E| \cdot (1-\lambda)^t \cdot d_{\max}}\right) \tag{42}$$

Thus, the optimal t which minimizes $f(t)$ is

$$t_{\mathrm{opt}} = \frac{-\log(-2|E| \cdot W(-\frac{1}{2e|E|d_{\max}}) \cdot d_{\max})}{\log(1-\lambda)}, \tag{43}$$

where W is the Lambert W-function.

An interesting observation from (43) is that t_{opt} is indeed irrelevant to Δ. In other words, no matter how stringent (or loose) the requirement on Δ is, as long as Δ is smaller than $1/(2|E|)$ (i.e., every node must have a positive probability to be selected), IDEAL-WALK always outperforms the input random walk.

Finally, note that

$$c \leq \frac{-\log(-2|E| \cdot W(-\frac{1}{2e|E|d_{\max}}) \cdot d_{\max})}{\log(1-\lambda)} \cdot \frac{1 - 2|E| \cdot \Delta}{1 + \frac{1}{W(-\frac{1}{2e|E|d_{\max}})}}. \tag{44}$$

Hence the query-cost ratio upper bound in the theorem. □

One can make an interesting observation from the proof of the theorem on how the performance of IDEAL-WALK changes when the walk length it takes grows larger. Initially,

a larger t leads to a smaller c, i.e., the expected query cost per sample for IDEAL-WALK, until t reaches the optimal value t_{opt}. Afterwards, a larger t will lead to a larger c, until $c = c_{\text{RW}}$ and IDEAL-WALK becomes equivalent with the input random walk.

6.2.2 Algorithm WALK

While the above theoretical analysis demonstrates the significant potential of query-cost savings by our WALK-ESTIMATE scheme, there is one key issue remaining before one can instantiate our idea into a practical WALK algorithm: in practice when the graph topology is not known beforehand, how can we determine the number of steps to walk before calling the ESTIMATE algorithm and performing the rejection sampling process? As one can see from the above discussions, an overly small length would lead to most samples being rejected, while an overly large one would incur unnecessary query cost for the WALK step.

Fortunately, we found through studies over real-world data (more details in the experimental evaluation section) that the setting of walk length is usually easy in practice as long as we set the walk length conservatively rather than aggressively. To understand why, note from the above case study, specifically the change of query cost per sample with walk length, that the query cost drops sharply before reaching the optimal walk length, the increase afterwards is much slower. As such, a reasonable strategy for setting the walk length is to be conservative rather than aggressive - i.e., giving preference to a longer, more conservative walk length. As we shall further elaborate in the experiments section, we use a default walk length of two times the graph diameter, which is conservatively estimated to be 10 for real world online social networks.

It is important to note that, while our experiments demonstrate that the above described heuristic strategy for setting the walk length works well over real world social networks, it is not a theoretically proven technique that works for all graphs. A simple counterexample here is the above-discussed Barbell graph - i.e., two complete graphs connected by one

node, with one edge connected to each half. One can see that, while the graph has a very short diameter (i.e., 3), a random walk of length 6 is highly unlikely to cross to the other half of the graph, unless it starts from one of the three nodes that connect the two halves together. As such, the above heuristics for setting the walk length would yield an extremely small sample-acceptance probability and, therefore, a high query cost.

6.3 ESTIMATE

Algorithm WALK leaves as an open problem of the estimation of sampling probability for a given node, so as rejection sampling can be applied to reach the input target distribution. In this section, we address this problem with Algorithm ESTIMATE. Specifically, we shall first describe a simple algorithm which, somewhat surprisingly, provides a completely unbiased estimation for the sampling probability with just a few queries. Unfortunately, we also point out a problem of this simple method: its high estimation variance which, despite the unbiasedness, still leads to a large error. To address this problem, we develop two heuristics, initial crawling and weighted sampling, to significantly reduce the estimation variance while requiring only a small number of additional queries.

6.3.1 UNBIASED-ESTIMATE

Unbiased Estimation of Sampling Probability: Recall that we use $p_t(u)$ to denote the probability for a node u to be visited at Step t of a random walk conducted by WALK, and $N(u)$ is the set of neighbors of u. To illustrate the key idea of UNBIASED-ESTIMATE, we start by considering the case where the input random walk is the simple random walk. One can see that

$$p_t(u) = \sum_{u' \in N(u)} \frac{p_{t-1}(u')}{|N(u')|}. \tag{45}$$

Thus, given u, a straightforward method of estimating $p_t(u)$ is to select uniformly at

random a neighbor of u (i.e., $u' \in N(u)$), so as to reduce the problem of estimating $p_t(u)$ to estimating $p_{t-1}(u')$, because an unbiased estimation of $p_t(u)$ is simply

$$\tilde{p}_t(u) = \frac{|N(u)|}{|N(u')|} \cdot p_{t-1}(u'). \tag{46}$$

An important property of such an estimation is that as long as we can obtain an unbiased estimation of $p_{t-1}(u')$, say $\tilde{p}_{t-1}(u')$, then the estimation for $p_t(u)$ will also be unbiased. The reason for the unbiasedness can be stated as follows: Note that due to the conditional independence of the estimation of $\tilde{p}_{t-1}(u')$ with the selection of u' from $N(u)$, we have

$$E(\tilde{p}_t(u)) = \sum_{u' \in N(u)} \frac{1}{|N(u)|} \cdot \frac{|N(u)|}{|N(u')|} \cdot E(\tilde{p}_{t-1}(u')) \tag{47}$$

$$= \sum_{u' \in N(u)} \frac{1}{|N(u')|} \cdot E(\tilde{p}_{t-1}(u')) \tag{48}$$

$$= \sum_{u' \in N(u)} \frac{1}{|N(u')|} \cdot p_{t-1}(u') = p_t(u), \tag{49}$$

where $E(\cdot)$ represents the expected value taken over the randomness of the estimation process.

Given the unbiasedness property, we can run a recursive process for estimating $p_t(u)$ (with a decreasing subscript t) until reaching $p_0(w)$. Now we have $p_0(w) = 1$ if w is the starting node of the random walk and 0 otherwise. One can see that this recursive process leads to an unbiased estimation of $p_t(u)$. We refer to this estimation method as UNBIASED-ESTIMATOR. The generic design of UNBIASED-ESTIMATOR (for any input random walk) is depicted in Algorithm 1.

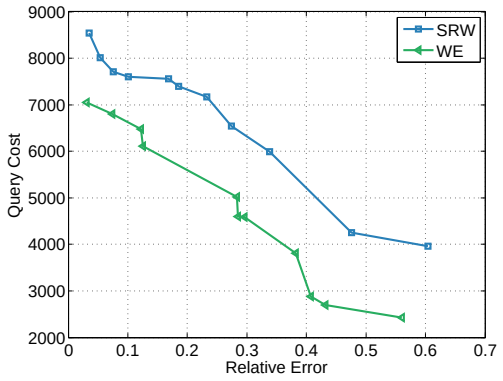
Algorithm 1 UNBIASED-ESTIMATE

1: **Input:** Node u, Starting node w, Length of walk t
2: **If** $t = 0$ and u == w then **return** 1
3: **If** $t = 0$ and u != w then **return** 0
4: Choose a node u' uniformly at random from $N(u)$
5: **return** $\frac{|N(u)|}{|N(u')|}$ · UNBIASED-ESTIMATE(u', w, $t-1$)

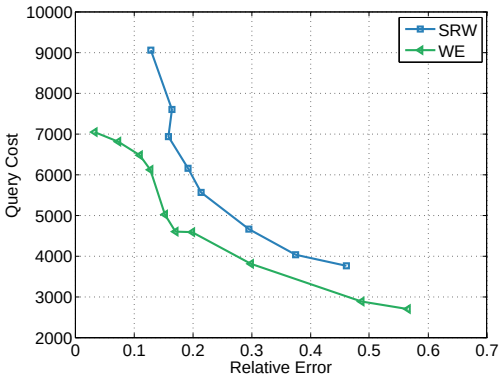
Analysis of Estimation Variance: While the above UNBIASED-ESTIMATOR produces an unbiased estimations of the sampling probability, it also has an important problem: a high estimation variance which leads to a high estimation error (unless the estimator is repeatedly executed to reduce variance - which would lead to a large query cost nonetheless). Specifically, the estimation variance on the last few steps of the recursive process (i.e., with the smallest subscript in $p_t(u)$) are amplified significantly in the final estimation. To see this, consider a simple example of a k-regular graph. With UNBIASED-ESTIMATOR, the estimation of $p_t(u)$ is either $\tilde{p}_t(u) = 1$ (when the node w encountered at $p_0(w)$ is the staring node) or 0 otherwise. As a result, the relative standard error for the estimation of $p_t(u)$ is exactly $\sqrt{(1 - p_t(u))/p_t(u)}$. Since $p_t(u)$ is usually extremely small for a large graph, the relative standard error can be very high for UNBIASED-ESTIMATOR.

6.4 Experiments

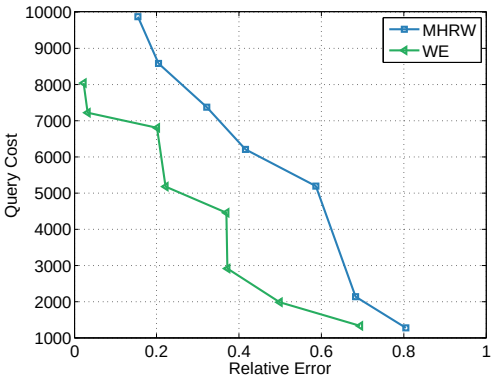
Aggregate estimation: We started by testing how WE performs against the baseline SRW and MHRW on the fundamental tradeoff in social network sampling - i.e., sample bias vs. query cost. The results over Google Plus are shown in Figure 28. Specifically, subgraphs (a) and (b) depict SRW and WE with SRW as input random walk, while (c) and (d) are corresponding to MHRW. The AVG aggregate used to measure sample bias is AVG degree for (a) and (c) and AVG self-description length for (b) and (d). As one can see from the figure, our algorithm significantly outperforms SRW and MHRW - i.e., offers substantially smaller relative error for the same query cost - on both aggregates tested. Figure 19 shows the results over the Yelp dataset. Specifically, subgraph (a) shows the AVG aggregate of the node attributes, i.e. star rating while subgraphs (b), (c), and (d) is the results of AVG aggregate of the topological properties, i.e, degree, shortest path length, and clustering coefficient. The results confirm the fact that WE provides smaller relative error with the same query cost. We also test our algorithm in Twitter dataset and the results in Figure 20 shows that AVG of the in-degree, out-degree, shortest path length, and clustering coefficient of the



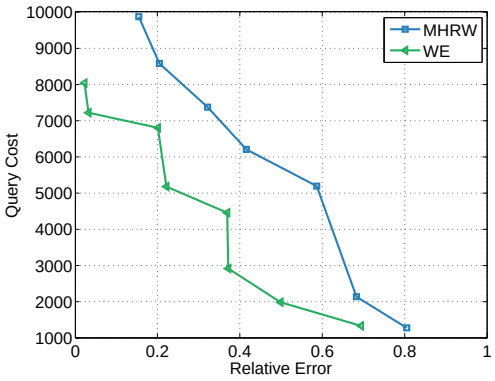
(a) Average Degree (SRW)



(b) Average Self-description Length (SRW)



(c) Average Degree (MHRW)



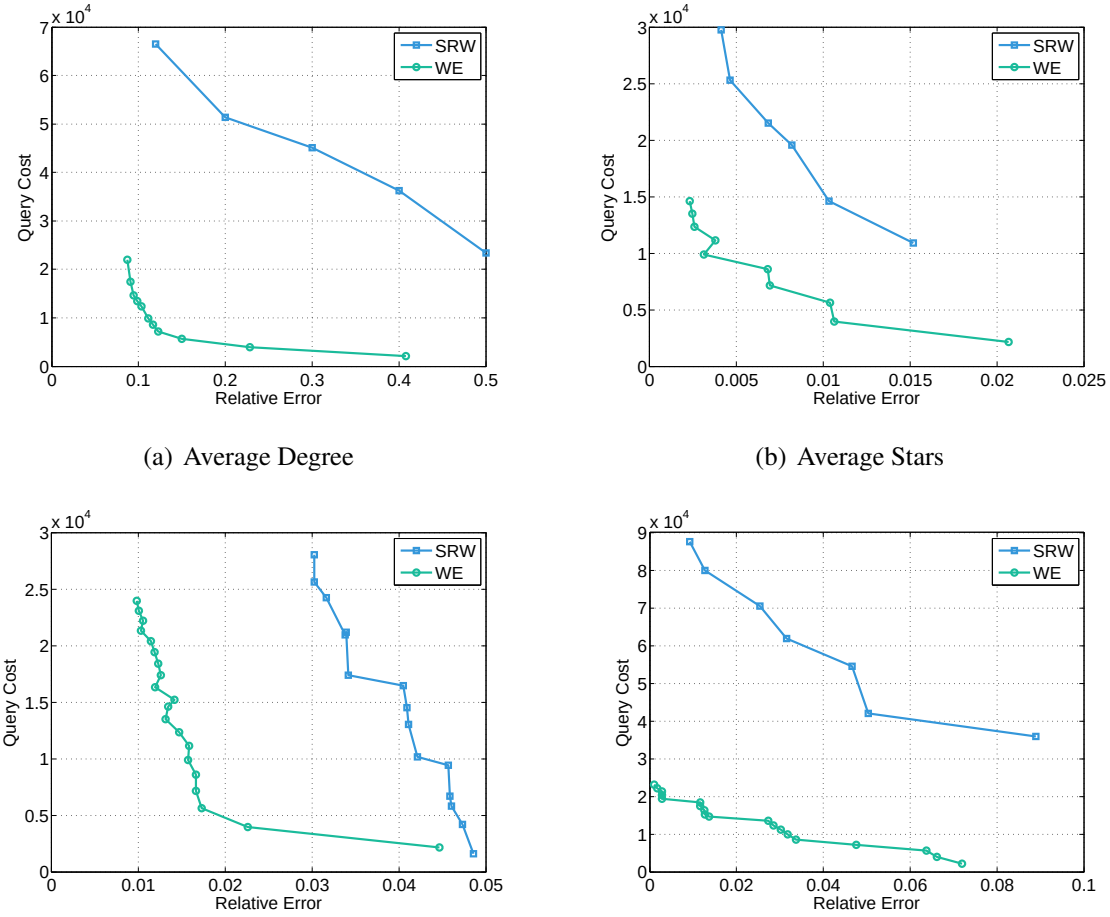
(d) Average Self-description Length (MHRW)

Figure 18: Relative Error of the Average Estimations vs Query Cost in Google Plus.

samples retrieved by the WE has smaller relative error than SRW for the same query cost.

We also study how our proposed variance reduction techniques improves the efficiency of our algorithm by comparing the performance of WE, WE-None, WE-Crawl and WE-Weighted, again according to how the relative error of aggregate estimation changes with the query cost. Figure 21 depicts the result for Google Plus, according to the same subgraph setup (i.e., random-walk/aggregate combination) as Figure 28. One can see that, as expected in all cases, WE outperforms the single-heuristics variants, which in turn outperform the theoretical variant of the algorithm.

Finally, we tested the quality of samples obtained by WE, in order to verify that the above-tested performance enhancements are not merely from walks being shorter, but from



(a) Average Degree

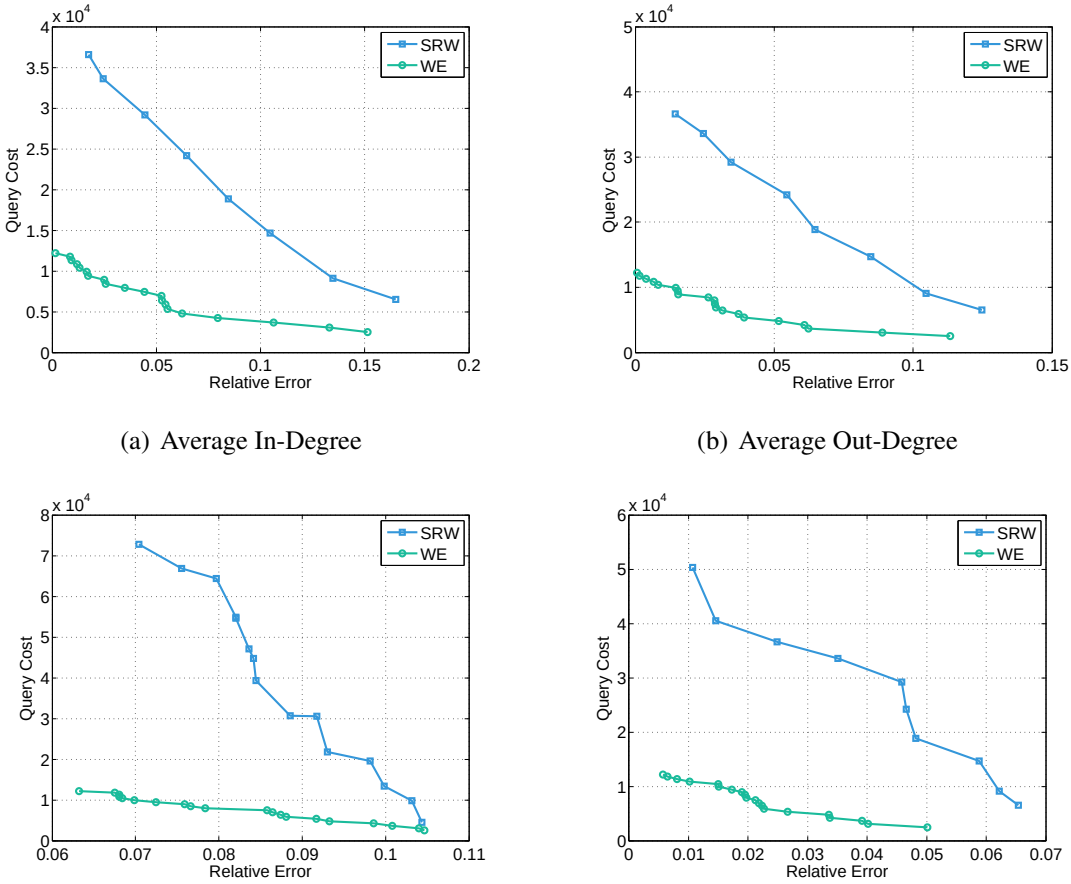
(b) Average Stars

(c) Average Shortest Path

(d) Average Local Clustering Coefficient

Figure 19: Relative Error of the Average Estimations vs Query Cost in Yelp.

an equal or higher quality sample as well. To this end, Figure 22 depicts how the relative errors on AVG estimations change with the number of samples produced by SRW, MHRW, and the corresponding WEs, respectively on Google Plus - again according to the same subgraph setup as Figure 28. One can see that in all cases, the samples produced by WE achieves smaller relative error than the corresponding input random walks (with Geweke convergence monitor), indicating the smaller sample bias achieved by WE. The results for the Yelp and Twitter are similar to those of Google Plus and, due to space limitations, are not included in the paper. More experimental results can be found in the technical report [30].



(a) Average In-Degree

(b) Average Out-Degree

(c) Average Local Clustering Coefficient

(d) Average Local Clustering Coefficient

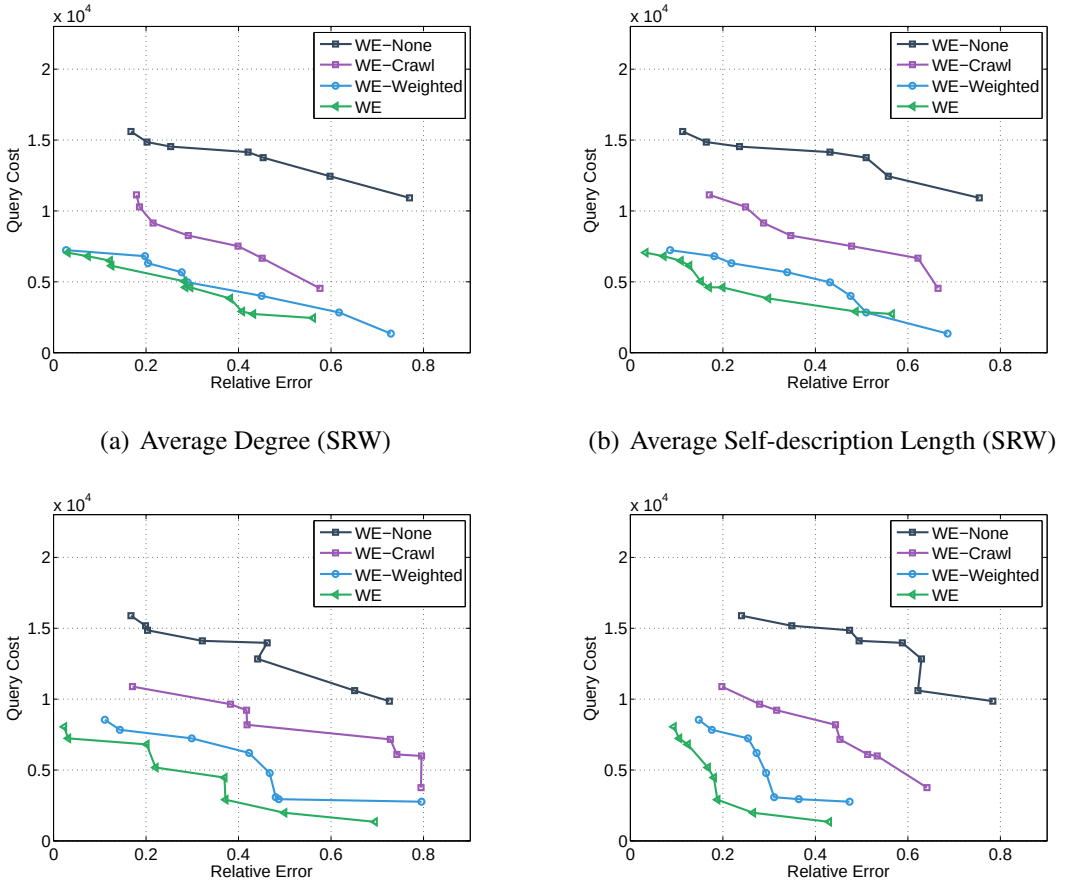
Figure 20: Relative Error of the Average Estimations vs Query Cost in Twitter (from SNAP repository).

Chapter 7 - Leveraging the history of Random Walks

7.1 Introduction

7.1.1 Our Idea: History-Aware Random Walks

The focus of this method is to offer a *"drop-in" replacement* for this core design (of random walk), such that existing sampling-based analytics techniques over online social networks, no matter which analytics tasks they support or graph topologies they target, can have a better efficiency without changing other parts of their design. To do so, we shorten the burn-in period (i.e., reduce the query cost) of the fundamental random-walk design



(a) Average Degree (SRW)

(b) Average Self-description Length (SRW)

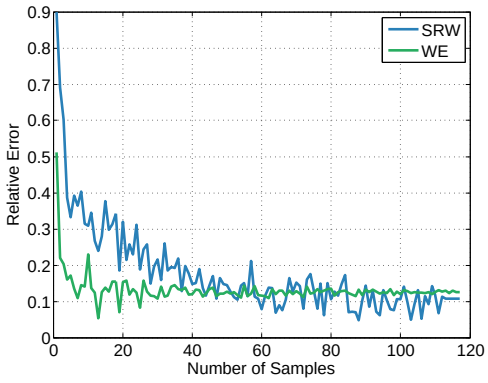
(c) Average Degree (MHRW)

(d) Average Self-description Length (MHRW)

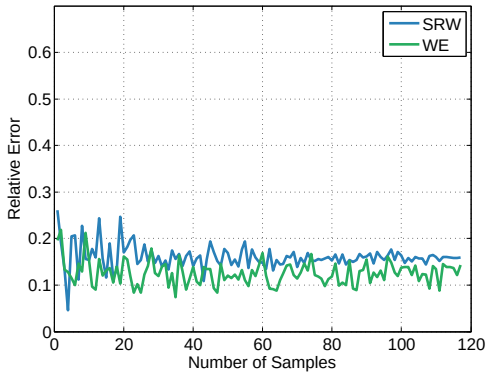
Figure 21: Improvement Trend: Relative Error of the Average Estimations vs Query Cost in Google Plus.

without changing the stationary distribution it achieves - ensuring the transparency of this change to how the core design is called upon in data analytics.

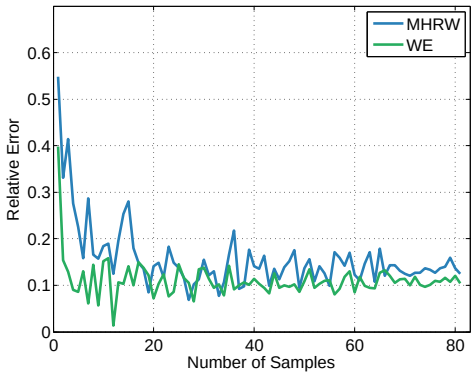
Technically, our key idea here is motivated by an observation on the potential waste of queries caused by the existing random walk design: Most existing random walk based techniques are Markov Chain Monte Carlo (MCMC) methods that are *memoryless* - i.e., they do not take into account the historic nodes encountered in a random walk in the design of future transitions. These methods, while simple, waste substantial chances for leveraging historic queries to speed up random walks - a waste we aim to eliminate with our proposed ideas.



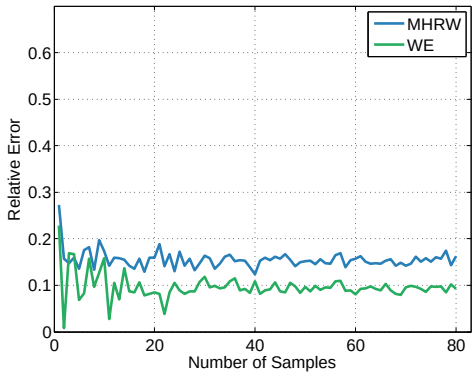
(a) Average Degree (SRW)



(b) Average Self-description Length (SRW)



(c) Average Degree (MHRW)



(d) Average Self-description Length (MHRW)

Figure 22: Relative Error of the Average Estimations vs Number of Samples in Google Plus.

Our main idea is to introduce historic dependency to the design of random walks over graphs. Specifically, we consider the nodes already visited by the current random walk while deciding where to go for the next step. We start with developing a simple algorithm called Circulated Neighbors Random Walk (CNRW). The difference between CNRW and the traditional random walk is sampling with and without replacement when deciding on the next move. Specifically, consider a random walk with the last move being $u \rightarrow v$. To determine the next move, the traditional random walk design is to sample uniformly at random from the neighbors of v. With CNRW, this sampling is done *with replacement* - in other words, if the random walk has transmitted through $u \rightarrow v \rightarrow w$ before, then we exclude w from being considered for the next move, until the random walk has passed through $u \rightarrow v \rightarrow x$ for all neighbors x of v. We prove that, while CNRW shares the exact same stationary distribution as the traditional simple random walk, it is provably more (or equally) efficient than the traditional random walk no matter what the underlying topology is.

A rationale behind the design of CNRW can be explained with the following simple example. Note that if a random walk over a large graph comes back to $u \rightarrow v$ after just passing though $u \rightarrow v \rightarrow w$, it is likely an indication that $v \rightarrow w$ leads to a small component that is not well connected to the rest of the graph (as otherwise the probability of going back to u should be very small). Falling into such a small component "trap" is undesirable for a random walk, which needs to "spread" to the entire graph as quickly as possible. Thus, an intuitive idea to improve the efficiency of a random walk is to avoid following $v \rightarrow w$ again, so as to increase the chance of following the other (hopefully better-for-random-walk) edges associated with v. This intuition leads to the "circulated", without-replacement, design of CNRW[11].

Based on the idea of CNRW, we develop GroupBy Neighbors Random Walk (GNRW),

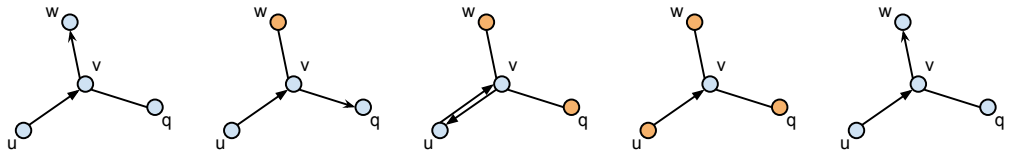
[11] One might wonder why the circulation is conditioned upon traveling through an edge (i.e., $u \rightarrow v$) again rather than traveling through a vertex (say v) again - this is indeed a subtle point which we further address in Section 3.2.

which further improves the performance of a random walk by considering not only the nodes visited by a random walk, but also the observed attribute values of these visited nodes. To understand the key idea of GNRW, consider the following intuitive observation on the structure of an online social network: users with similar attribute values (e.g., age, interests, occupation) are more likely to be connected with each other. Leveraging this observation, to decide which neighbor of v to transit to from $u \rightarrow v$, GNRW partitions all neighbors of v into a number of strata according to their values on an attribute of interest (often the measure attribute to be aggregated - see discussions in Section 7.3.1). Then, GNRW first selects a stratum uniformly at random without replacement - i.e., it "circulates" among the strata until selecting each stratum once. Then, GNRW chooses a neighbor of v from the selected stratum, again uniformly at random without replacement. This neighbor then becomes the next step.

Similar to the theoretical analysis of CNRW, we also prove that GNRW shares the same stationary distribution as the traditional random walk. Intuitively, the design of GNRW aims to "speed up" the random walk by alternating between different attribute values faster, instead of getting "stuck" on a small component of the graph sharing the same (or similar) attribute value. We shall demonstrate through experimental evaluation that GNRW can significantly improve the efficiency of random walks, especially when the ultimate objective of these random walks is to support aggregate estimations on attributes used for stratification in GNRW.

7.2 CNRW: Circulated Neighbors Random Walk

In this section, we develop Circulated Neighbors Random Walk (CNRW), our first main idea for introducing historic dependency to the design of random walks over graphs. Specifically, we start with describing the key idea of CNRW, followed by a theoretical analysis of (1) its equivalence with simple random walk (SRW) in terms of (stationary)



(a) $N(v)$ (b) $N(v) - \{w\}$ (c) $N(v) - \{w, q\}$ (d) $N(v) - \{w, q, u\}$ (e) $N(v)$

Figure 23: Demo of CNRW, it chooses the next candidate from the set $N(v) - b(u, v)$.

sampling distribution, and (2) its superiority over SRW on sampling efficiency. Finally, we present the pseudo code for Algorithm CNRW.

7.2.1 Main Idea and Justification

Key Idea of CNRW: In traditional random walks, the transition at each node is *memoryless* - i.e., when the random walk arrives at a node, no matter where the walk comes from (i.e., what the incoming edge is) or which nodes the walk has visited, the outgoing edge is always chosen uniformly at random from all edges attached to the node. The key idea of CNRW is to replace such a memoryless transition to a stateful process. Specifically, given the previous transition of the random walk $u \rightarrow v$, instead of selecting the next node to visit by *sampling with replacement* from $N(v)$, i.e., the neighbors of v, we perform such sampling *without replacement*.

Such a change is demonstrated through an example in Figure 23: When a CNRW transits through $u \rightarrow v$ for the first time, it selects the next node to visit in the same way as traditional random walk - i.e., by choosing w uniformly at random from $N(v)$. Nonetheless, if the random walk transits through $u \rightarrow v$ again in the future, instead of selecting the next node from $N(v)$, we limit the choice to be from $N(v) - \{w\}$. One can see that, given a transition $u \rightarrow v$, our selection of the next node to visit is a process of *sampling without replacement* from $N(v)$. This, of course, continues until $\forall w \in N(v)$, the random walk has passed through $u \rightarrow v \rightarrow w$, at which time we reset memory and restart the process of sampling without replacement. Also, we introduce a notation $b(u, v)$, which is defined as a

set of nodes in $N(v)$ that we have passed through. Thus, we generalize the idea of CNRW as:

1. Each time when the random walk travels from u to v, we uniformly choose the next candidate node w from $N(v) - b(u, v)$.

2. Let $b(u, v) \leftarrow b(u, v) \cup \{w\}$. If $b(u, v) = N(v)$, let $b(u, v) = \emptyset$.

Intuitive Justification: To understand why CNRW improves the efficiency of the sampling process, we start with an intuitive explanation before presenting in the next subsection a rigid theoretical proof of CNRW's superiority over the traditional simple random walk (SRW) algorithm.

Intuitively, the justification for CNRW is quite straightforward: If a random walk travels back to $u \rightarrow v$ after only a small number of steps, it means that the choice of $v \rightarrow w$ is not a "good" one for sampling because, ideally, we would like the random walk path to "propagate" to all parts of graph as quickly as possible, instead of being stuck at a small, strongly connected subgraph like one that includes u, v and w. As such, it is a natural choice for CNRW to avoid following $v \rightarrow w$ again in the next transition, but to instead choose a node from $N(v) - \{w\}$ uniformly at random.

7.2.2 Theoretical Analysis

In this section, we will first introduce the concept of *path blocks* and then theoretically show that CNRW and SRW have the same stationary distribution - with the fundamental difference between the two being the how path blocks alternate during the random walk process. Finally, we prove that CNRW produces more accurate samples (i.e. with smaller or equal asymptotic variance than SRW) regardless the graph topology.

Path blocks: Intuitively, path blocks divide a random walk's path into consecutive segments. Formally, we have the definition of path blocks:

Definition 8. [Path blocks]. *Given a random walk over graph $G(V, E)$, denote its path as $X_0, X_1, \ldots, X_m, \ldots$, where $X_i \in V$. A path block B_{ij} is defined as*

$$B_{ij} = \{X_i, X_{i+1}, \ldots, X_j\}, j > i. \tag{50}$$

An interesting observation critical for the design of CNRW is that for path block B_{ij}, X_i can be the same as X_j, which is called *recurrence* in a Markov Chain [31]. As a typical positive recurrent Markov Chain, SRW will always go back to the same node if the number of steps is sufficiently large. Formally, positive recurrence means that if we denote $M_i = \sum_{n=0}^{\infty} I\{X_n = i | X_0 = i\}, i \in V$, then its expected value $E(M_i)$ is always finite. The reason why recurrence is an important concept for CNRW is because, as one can see from its own, the key difference between CNRW and SRW is how we select the next transition when a recurrence happens.

As mentioned in the introduction, for the purpose of CNRW design, one actually has two choices on how to define a recurrence - based on edge-based and node-based path blocks, respectively:

$$\text{Edge-based: recurrence if } X_i = X_{j-1}, X_{i+1} = X_j \tag{51}$$

$$\text{Node-based: recurrence if } X_i = X_j \tag{52}$$

In CNRW (and the following discussions in GNRW), we choose the edge-based design for the following main reason: Note that with the edge-based definition, path blocks separated by recurrences have much longer expected length than with the node-based design, because it takes much more steps for a random walk to travel back to an edge than a node. As such, edge-based path blocks tend to have similar distribution for each block (proved in Theorem 8), leading to a smaller inter-pathblock variance, which in turn results in a more significant reduction of asymptotic variance brought by CNRW's without-replacement design (proved in Theorem 9). We also conducted extensive experiments to verify the supe-

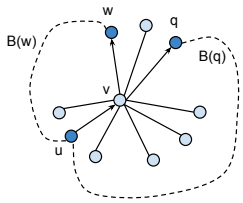


Figure 24: Path blocks in CNRW

riority of edge-based design - though the results are not included in this paper due to space limitations.

Sampling Distribution: For SRW, the sampling distribution is $\pi_j = k_j/2|E|$. CNRW, on the other hand, is a higher-order Markov Chain. We will introduce a specific kind of path blocks first.

Definition 9. [Path blocks $B(i)$ rooted on edge e_{uv}]. *Given a random walk with subsequence $u \to v \to i \to u_1 \to u_2 \to \cdots \to u_h \to u \to v \to \cdots$ where $i \neq u$, if there does* not *exist $j \in [1, h-1]$ with $u_i = u$ and $u_{i+1} = v$, then we call the prefix of this subsequence ending on u_h - i.e., $u \to v \to i \to u_1 \to u_2 \to \cdots \to u_h$ an instance of* path block $B(i)$ *rooted on e_{uv}.*

What we really mean here is that once a random walk reaches $u \to v$ for the first time, the remaining random walk can be partitioned into consecutive, *non-overlapping* path blocks rooted on edge e_{uv}, each of which starts with $u \to v$ and ends with the last node visited by the random walk before visiting $u \to v$ again. Using this definition, the following theorem indicates the equivalence of sampling distributions of CNRW and SRW.

Theorem 8. *Given a graph $G(V, E)$, CNRW has the stationary distribution $\pi(v) = k_v/2|E|$.*

Proof. We will construct the proof in two steps.

- First, we prove that the stationary distribution is the same (i.e., $\forall s \in V, \pi(s) = k_s/(2|E|)$ where k_s is the degree of s for simple random walk) after applying CNRW to any *single* edge of the graph.

- Second, we prove that, $\forall k > 1$, if the stationary distribution is the same after applying CNRW to any set of $(k-1)$ edges, then the stationary distribution will remain the same after we apply CNRW to any additional (k-th) edge in the graph.

Step 1.

We start with the first step - i.e., applying CNRW over only one edge, say $u \to v$. We denote this CNRW as CNRW(1). Let SRW be the original simple random walk. The most critical concept for our proof is a *path block* $B(i)$ for each neighbor of v, i.e., $i \in N(v)$. According to the definition in the revised version (also discussed in response to D2), $B(i)$ represents a segment of the random walk that starts with $u \to v \to i$ and ends before the random walk travels back through $u \to v$ next time - e.g., if a segment of random walk is $u \to v \to i \to \cdots \to x \to u \to v \to j \to \cdots \to y \to u \to v$ (the two abbreviated parts in the middle do not contain $u \to v$), then this segment can be rewritten as $B(i) \to B(j) \to u \to v$, where $B(i) = u \to v \to i \to \cdots \to x$ and $B(j) = u \to v \to j \to \cdots \to y$ (here x and y can be any arbitrary node that has an edge linked to u[12]).

Let $X^{CNRW(1)}$ and X^{SRW} be a random walk performed by CNRW(1) and SRW, respectively. Let $X_m^{CNRW(1)}$ (resp. X_m^{SRW}) be the node accessed by $X^{CNRW(1)}$ (resp. X^{SRW}) at the m-th step. The sampling probability for any node $j \in V$ according to the stationary distribution of the random walks are

$$\pi^{CNRW(1)}(j) = \lim_{n\to\infty} \frac{\sum_{m=1}^{n} Pr\{X_m^{CNRW(1)} = j\}}{n}, \tag{53}$$

$$\pi^{SRW}(j) = \lim_{n\to\infty} \frac{\sum_{m=1}^{n} Pr\{X_m^{SRW} = j\}}{n}, \tag{54}$$

We now consider $Y^{CNRW(1)}$ (resp. Y^{SRW}), i.e., an infinite sub-sequence of $X^{CNRW(1)}$ (resp. X^{SRW}) which includes all steps occurring after the first visit of $u \to v$. There are two important observations here:

- First, both $Y^{CNRW(1)}$ and Y^{SRW} can be completely separated into path blocks $B(i_1)$

[12]Note that, as further discussed in our response to D2, $B(i)$ and $B(j)$ are not overlapping on $u \to v$.

$\rightarrow B(i_2) \rightarrow B(i_3) \rightarrow \cdots$ where $\forall h, i_h \in N(v)$. The sole difference between CNRW(1) and SRW is on *how a path block is chosen* (i.e., CNRW(1) guarantees $i_1 \neq i_2 \neq \cdots \neq i_{|N(v)|}$ while SRW does not). The internal transitions within a path block, on the other hand, are indistinguishable for both approaches - i.e., for CNRW(1), once $B(i_1)$ is chosen for a slot, the internal transitions within $B(i_1)$ follow the exact same probability distribution as that of a $B(i_1)$ chosen for SRW.

- Second, since $Y^{CNRW(1)}$ (resp. Y^{SRW}) is equivalent with $X^{CNRW(1)}$ (resp. X^{SRW}) after the first appearance of $u \rightarrow v$, as long as $u \rightarrow v$ appears in the random walk ($X^{CNRW(1)}$), there must be

$$\lim_{n\rightarrow\infty} \frac{\sum_{m=1}^{n} Pr\{X_m^{CNRW(1)} = j\}}{n} \tag{55}$$

$$= \lim_{n\rightarrow\infty} \frac{\sum_{m=1}^{n} Pr\{Y_m^{CNRW(1)} = j\}}{n} \tag{56}$$

$$\lim_{n\rightarrow\infty} \frac{\sum_{m=1}^{n} Pr\{X_m^{SRW} = j\}}{n} \tag{57}$$

$$= \lim_{n\rightarrow\infty} \frac{\sum_{m=1}^{n} Pr\{Y_m^{SRW} = j\}}{n} \tag{58}$$

Note that if $u \rightarrow v$ never appears in $X^{CNRW(1)}$, then this Step is already proved: Since CNRW(1) never kicks in, it of course yields the exact same stationary distribution as SRW.

Note that, according to the design of CNRW, once we rewrite $Y^{CNRW(1)}$ to path blocks, i.e., $B(i_1) \rightarrow B(i_2) \rightarrow B(i_3) \rightarrow \cdots$, we will see that all $|N(v)|$ path blocks appear in an iterative fashion (e.g., no path block $B(i_x)$ appears twice before another $B(i_y)$ appears

once). In other words,

$$\lim_{n\to\infty} \frac{\sum_{m=1}^{n} Pr\{Y_m^{CNRW(1)} = j\}}{n} \tag{59}$$

$$= \frac{\sum_{i\in N(v)} \lim_{n'\to\infty} \frac{\sum_{b=1}^{n'} Pr\{B(i)_b = j\}}{n'}}{|N(v)| \cdot n'} \tag{60}$$

$$= \lim_{n\to\infty} \frac{\sum_{i\in N(v)} \sum_{b=1}^{n} Pr\{B(i)_b = j\}}{|N(v)| \cdot n}. \tag{61}$$

where $B(i)_b$ is the node at Step b of path block $B(i)$.

On the other hand, consider the rewrite of Y^{SRW} to path blocks. In this case, the path blocks do not appear in an iterative fashion. Instead, each path block is generated i.i.d. uniformly from all $|N(v)|$ possible path blocks. Let $P(B(i))$ be the probability for B_i to be chosen at each slot - one can see that $P(B_i) = 1/|N(v)|$. Thus,

$$\lim_{n\to\infty} \frac{\sum_{m=1}^{n} Pr\{Y_m^{SRW} = j\}}{n} \tag{62}$$

$$= \lim_{n\to\infty} \sum_{i\in N(v)} P(B(i)) \cdot \frac{\sum_{b=1}^{n} Pr\{B(i)_b = j\}}{n} \tag{63}$$

$$= \lim_{n\to\infty} \frac{\sum_{i\in N(v)} \sum_{b=1}^{n} Pr\{B(i)_b = j\}}{|N(v)| \cdot n}. \tag{64}$$

One can observe from (61) and (64) that

$$\lim_{n\to\infty} \frac{\sum_{m=1}^{n} Pr\{Y_m^{CNRW(1)} = j\}}{n} = \lim_{n\to\infty} \frac{\sum_{m=1}^{n} Pr\{Y_m^{SRW} = j\}}{n}. \tag{65}$$

This, combined with (53), (54) and (56), (58), leads to

$$\pi^{CNRW(1)}(j) = \pi^{SRW}(j), \tag{66}$$

i.e., the stationary distribution achieved by CNRW(1) is exactly the same as that of SRW.

Step 2.

Now we need to prove the second step: if we assume that stationary distribution is the same after applying CNRW to any set of $(k-1)$ edges, then the stationary distribution will remain the same after we apply CNRW to any additional k-th edge in the graph. We denote this CNRW as CNRW(k), and let $u \to v$ be the additional k-th edge. Our definition of path blocks $B(i)$ for $i \in N(v)$ remains exactly the same as in Step 1. Note that

$$\pi^{CNRW(k)}(j) = \lim_{n\to\infty} \frac{\sum_{m=1}^{n} Pr\{X_m^{CNRW(k)} = j\}}{n} \tag{67}$$

$$\pi^{CNRW(k-1)}(j) = \lim_{n\to\infty} \frac{\sum_{m=1}^{n} Pr\{X_m^{CNRW(k-1)} = j\}}{n} \tag{68}$$

The difference between CNRW(k) and CNRW(k-1) is only on how a path block is chosen. For CNRW(k), once $B(i_1)$ is chosen for a slot, the internal transitions within $B(i_1)$ follow the exact same probability distribution as that of a $B(i_1)$ chosen for CNRW(k-1). Therefore, according to the design of CNRW, once we rewrite CNRW(k) to path blocks, i.e. $B(i_1)B(i_2)B(i_3)\ldots$, we will get:

$$\lim_{n\to\infty} \frac{\sum_{m=1}^{n} Pr\{Y_m^{CNRW(k)} = j\}}{n} = \lim_{n\to\infty} \frac{\sum_{i\in N(v)} \sum_{b=1}^{n} Pr\{B(i)_b = j\}}{|N(v)| \cdot n}. \tag{69}$$

And CNRW(k-1) is just SRW based on the route $u \to v$ (since edge $e_{u,v}$ is the additional k-th edge that has been applied by CNRW(k)), $B(i)$ is chosen uniformly at random in CNRW(k-1). Thus,

$$\lim_{n\to\infty} \frac{\sum_{m=1}^{n} Pr\{Y_m^{CNRW(k-1)} = j\}}{n} \tag{70}$$

$$= \lim_{n\to\infty} \sum_{i\in N(v)} P(B(i)) \cdot \frac{\sum_{b=1}^{n} Pr\{B(i)_b = j\}}{n} \tag{71}$$

$$= \lim_{n\to\infty} \frac{\sum_{i\in N(v)} \sum_{b=1}^{n} Pr\{B(i)_b = j\}}{|N(v)| \cdot n}. \tag{72}$$

We can combine the above equations (67) - (72), which leads to

$$\pi^{CNRW(k)}(j) = \pi^{CNRW(k-1)}(j). \tag{73}$$

□

One can see from the theorem that, for both SRW and CNRW, the probability for a node to be sampled is proportional to its degree. As such, CNRW can be readily used as a drop-in replacement of SRW.

Improvement on Sampling Efficiency: Having proved that CNRW and SRW produce the same (distribution of) samples - we now develop theoretical analysis on the efficiency of CNRW and its comparison with SRW. Note that, since the sampling distributions of both CNRW and SRW converge asymptotically to the stationary distribution (i.e., probability proportional to a node's degree), for a fair comparison between the efficiency of these two algorithms, we must first establish a measure of the *distance* between the sampling distribution achieved at h steps and the (eventual) stationary distribution. The reason is that, only when such a distance measure is defined, we can then compare the number of steps each algorithm requires to reduce the distance below a given threshold ϵ - apparently, the one which requires fewer steps achieves better sampling efficiency.

For the purpose of this paper, we follow the commonly used *asymptotic variance*. While there are multiple ways to understand the "physical meaning" of this measure, a simple one is that it measures the mean square error (MSE) one would get for an AVG aggregate (on the measure function $f(\cdot)$ used in the definition) by taking into account the nodes encountered in the first h steps of a random walk. One can see that, the lower this MSE is, the closer the sampling distribution at Step h is to the stationary distribution.

With the asymptotic variance measure, we are now ready to compare the sampling efficiency of CNRW and SRW. The following theorem shows that for any measure function

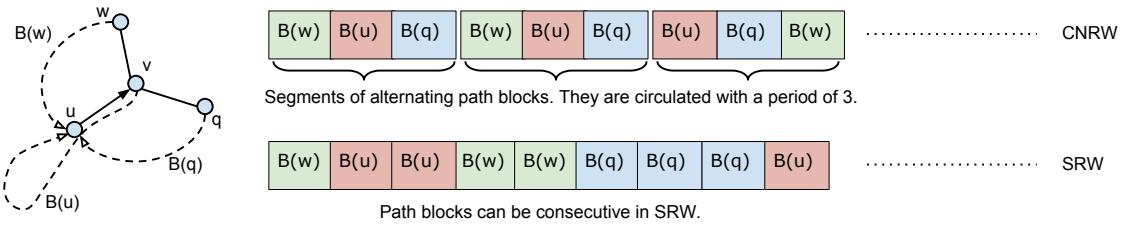


Figure 25: Comparison of the block distribution in CNRW and SRW.

$f(\cdot)$ and any value of h (i.e., the number of steps), CNRW always achieves a lower or equal variance - i.e., better or equal sampling efficiency - than SRW. Before presenting the rigid proof, we first briefly discuss the intuition behind this proof as follows.

The proof is constructed through induction. As such, we first consider the impact of CNRW on changing the transition after one edge, i.e., as in the above running example, the transition after $u \rightarrow v$. A key concept used in the proof is a segmentation of a long random walk into *segments* according to $u \rightarrow v$. Specifically, every segment of the random walk except the first one starts and ends with $u \rightarrow v$. Figure 25 shows an example of this segmentation. We can see that CNRW are generating segments containing alternating path blocks.

With the segmentation, a key idea of our proof is to introduce an *encoding* of each segment according to the first node it visits after $u \rightarrow v$. For example, in the case shown in Figure 25 where v has three neighbors $\{u, w, q\}$, we have three possible path blocks: $\{B(u), B(w), B(q)\}$. It is important to note that, since we are only considering the change after $u \rightarrow v$ at this time, every path block can be considered as being drawn from the exact same distribution no matter if CNRW or SRW is used - because CNRW does not make any changes once the first node after $u \rightarrow v$, i.e., the code, is determined, until the next time $u \rightarrow v$ is visited. This observation enables us to simple consider CNRW and SRW as sequences of *codes* (i.e. we can map path blocks into codes) in the efficiency comparison - as what happens within each path block is anyway oblivious to whether CNRW or SRW is being used.

Now studying the sequence of codes for CNRW and SRW, we make a straightforward yet interesting observation: given a sequence containing $\{B(u), B(w), B(q)\}$ with length h, the number of occurrences of $B(u)$, $B(w)$, and $B(q)$ will be the same (at least within ± 1 range) in CNRW, because they will alternately appear every three codes. On the other hand, with SRW, the number of occurrences for $\{B(u), B(w), B(q)\}$ is *statistically* equal but have an inherent variance in practice due to randomness. The elimination of this randomness/variance is exactly why CNRW tends to generate samples with a smaller variance than SRW - as rigidly shown in the following theorem.

Theorem 9. *Given a graph $G(V, E)$, any property function f, and the following two estimators for μ based on SRW ($\hat{\mu}$) and CNRW ($\hat{\mu'}$):*

$$\hat{\mu}_n = \frac{1}{n}\sum_{t=1}^{n} f(X_t), \quad \hat{\mu}'_n = \frac{1}{n}\sum_{t=1}^{n} f(X'_t), \tag{74}$$

then CNRW will have no greater asymptotic variance (defined in Section 2.2) than SRW

$$V_\infty(\hat{\mu'}) \leq V_\infty(\hat{\mu}). \tag{75}$$

Proof. The claim of asymptotic variance in Theorem 9 can be divided into 3 steps:

1. We reduce the problem to comparing asymptotic variance when the transition matrix P and P' (for SRW and CNRW accordingly) differ only for transitions involving one node's neighbors. In other words, given nodes u, v, we only consider the circulated neighbors rule for CNRW when $u \to v$, not all the nodes in G.

2. We then divide the random walk's trace into path blocks. These blocks are all starting with the route of $u \to v \to i$, $i \in N(v)$.

3. We see that blocks are equally likely, but may have difference distribution for their

contents. Blocks in CNRW are alternating, and they will have lower or equal asymptotic variance.

Step 1. Looking at one node's neighbors is enough. If we can prove for only one node's neighbors that CNRW achieves a smaller asymptotic variance, we can recursively apply it to each node and its neighbors. Therefore, given a specific node $v \in V$, one of its neighbor u, we only need to consider the circulated neighbors rule to $N(v)$ with an incoming route $u \to v$.

Step 2. For all the neighbors of v: $i \in N(v)$, we have path blocks $B(i)$ rooted on edge e_{uv}. For example, $B(w), B(q)$ are two blocks in Figure 24.

Step 3. We will show that the alternating appearance of the blocks $B(i)$ lowers (or at least not increase) the asymptotic variance. We note that the probability of the blocks $P(B(i)) = P(B(j)), i, j \in N(v)$, because $P(u \to v \to i) = P(u \to v \to j)$. Also, if we denote the number of block $B(i)$ before M blocks as $K_{B(i)}(M)$, then

$$|K_{B(i)}(M) - K_{B(j)}(M)| \leq 1, \quad \forall i, j \in N(v), M \geq 1 \tag{76}$$

holds for CNRW but not SRW. The sampling for the blocks is therefore stratified in CNRW. According to a Lemma in [31], the asymptotic variance of CNRW is no more than SRW.

Lemma [31]: *Let $Z_1, Z_2, \ldots$ be an irreducible Markov chain with state space $\{0, 1, 2\}$, whose invariant distribution, ρ, satisfies $\rho(0) = \rho(1)$. Let Q_z for $z = 0, 1, 2$ be distributions for pairs $(H, L) \in \mathbb{R} \times \mathbb{R}^+$ having finite second moments. Conditional on $Z_1, Z_2, \ldots$, let (H_i, L_i) be drawn independently from Q_{Z_i}. Define*

$$Z_i' = \begin{cases} Z_i & \text{if } Z_i = 2 \\ Z_k + \sum_{j=1}^{i-1} I_{\{0,1\}}(Z_j) \ \text{(modulo 2)} & \text{if } Z_i \neq 2 \end{cases} \tag{77}$$

where $k = \min\{i : Z_i \neq 2\}$. (In other words, the Z_i' are the same as the Z_i except that the positions where 0 or 1 occurs have their values changed to a sequence of alternating 0s

and 1*s.) Conditional on* $Z_1, Z_2, \ldots$*, let* (H_i', L_i') *be drawn independently from* $Q_{Z_i'}$*. Define two families of estimators as follows:*

$$R_n = \sum_{i=1}^{n} H_i \Big/ \sum_{i=1}^{n} L_i, \quad R_n' = \sum_{i=1}^{n} H_i' \Big/ \sum_{i=1}^{n} L_i' \tag{78}$$

Then the asymptotic variance of R' *is no greater than that of* R*. In other words,*

$$\lim_{n\to\infty} nVar(R_n') \leq \lim_{n\to\infty} nVar(R_n) \tag{79}$$

This lemma justifies the claim that partial stratification of sampling for blocks cannot increase asymptotic variance. And it is easily to extend to the case the state space is $\{0, 1, 2, \ldots, k\}$ as long as we keep the stratification of the states. In applying this lemma to the proofs in our theorem, $Z_1, Z_2, \ldots$ are identifiers for the type of each block, we can use $0 = B(i_0), 1 = B(i_1), \ldots$ etc. Please refer to [31] for more details. □

Theorem 9 establishes CNRW's superiority while the following Theorem 10 shows how significant the superiority over a concrete example: barbell graph. The probability of propagating from one subgraph to another in CNRW is much greater than the one in SRW.

Theorem 10. *Given a barbell graph* G*, which contains two complete subgraphs* G_1, G_2*. If we choose the initial node* $v \in G_1$*, then the expected value of* t_{CNRW} *and* t_{SRW} *satisfy that*

$$\frac{P_{CNRW}}{P_{SRW}} > \frac{|G_1|}{|G_1| - 1} \ln |G_1| \tag{80}$$

where P_{CNRW} *and* P_{SRW} *are the probability for CNRW and SRW to travel from* G_1 *to* G_2*.*

Proof. We denote the edge e_{uw} $(u \in G_1, w \in G_2)$ as the bridging edge in G. Now we only need to consider the transition probability of u. For SRW,

$$P_{SRW} = P(u \to w) = \frac{1}{|N(u)|} = \frac{1}{|G_1|}, \tag{81}$$

where $|G_1|$ is the number of nodes in G_1. For CNRW,

$$P_{CNRW} = P(u \to w) = \frac{1}{|G_1| - 1} \sum_{s \in \{G_1 - u\}} P(u \to w | s \to u) \tag{82}$$

$$= \frac{1}{|G_1| - 1} \sum_{i=0}^{|G_1|-1} \frac{1}{|N(u)| - i|} \tag{83}$$

$$> \frac{1}{|G_1| - 1} \int_1^{|N(u)|} \frac{1}{x} dx \tag{84}$$

$$= \frac{1}{|G_1| - 1} \ln |G_1| \tag{85}$$

$$= \left(\frac{|G_1|}{|G_1| - 1} \ln |G_1| \right) P_{SRW} \tag{86}$$

Eq (82) $\Rightarrow$ (83) because the previous accessed neighbors are evenly distributed among $N(u)$. □

7.2.3 Algorithm CNRW

Algorithm 2 Circulated Neighbors Random Walk

```
/* Given x_0 = u, x_1 = v, b(x_0, x_1) = ∅ */
/* Function b(u, v) can be implemented as a HashMap, and their initial value are all ∅. */
for i = 2 → sample_size do
    /* S denotes the next possible candidates */
    S ← N(x_{i-1}) − b(x_{i-2}, x_{i-1})
    if S ≠ ∅ then
        x_i ← uniformly choose a node from S
        b(x_{i-2}, x_{i-1}) = b(x_{i-2}, x_{i-1}) ∪ {x_i}
    else
        x_i ← uniformly choose a node from N(x_{i-1})
        b(x_{i-2}, x_{i-1}) = ∅
    end if
end for
```

Algorithm implementation. Algorithm 2 depicts the pseudo code for Algorithm CNRW. We note that the only data structure we maintain (beyond what is required for SRW) is a *historic hash map of outgoing transitions* $b(u, v)$ for each edge e_{uv} we pass through - i.e.,

if the random walk has passed through edge e_{uv} before, then $b(u, v)$ contains the neighbors of v which have been chosen previously (during the walk) as the outgoing transitions from v.

Time and space complexity. CNRW requires a hash map that continuously records the outgoing transitions for each edge, i.e. the key-value pair: $e_{uv} \rightarrow b(u, v)$. Assume CNRW walks K steps, and the keys e_{uv} are uniformly distributed among all the edges in the graph for CNRW (the same as SRW), then this hash map as a whole occupies expected $O(K)$ space (according to [11] if we are using dynamic perfect hashing). Also, each step's amortized expect time complexity is $O(1)$, so the total expected time complexity for K steps is $O(K)$.

7.3 GNRW: GroupBy Neighbors Random Walk

7.3.1 Basic Idea

Recall from Section 7.2.1 the main idea of CNRW: Given an incoming transition $u \rightarrow v$, we essentially *circulate* the next transition among the neighbors of v, ensuring that we do not attempt the same neighbor twice before enumerating every neighbor. The key idea of GNRW is a natural extension: Instead of performing the circulation at the granularity of each neighbor (of v), we propose to first *stratify* the neighbors of v into groups, and then circulate the selection among all groups. In Figure 26, for example, if we visit $u \rightarrow v$ for the second time, with the last chosen transition being from v to a(nother) node in S_2, then this time we will randomly pick one group from S_1 and S_3 (with probability proportional to the number of not-yet-attempted transitions in each group), and then pick a node from the chosen group uniformly at random. GNRW can be summarized as:

1. It has a global groupby function $g(\cdot)$ that will partition a node's neighbors into disjoint groups. For example, $g(N(v)) = \{S_1, S_2, \ldots, S_m\}$.

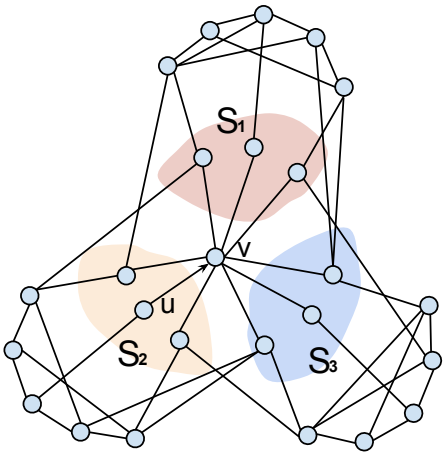


Figure 26: An example of partitioning a node's neighbors into 3 groups.

2. Each time when the random walk travels from u to v, we choose the next candidate group S_i from $N(v) - S(u,v)$ with probability $|S_i|/|N(v) - S(u,v)|$, where $S(u,v)$ is a set of groups we have accessed before.

3. Within S_i, we uniformly at random choose the next candidate node w from $S_i - b_{S_i}(u,v)$, where $b_{S_i}(u,v)$ is defined similar to CNRW's - with a range limited in S_i.

4. Let $b(u,v) \leftarrow b(u,v) \cup \{w\}$, and $S(u,v) \leftarrow S(u,v) \cup \{S_i\}$. If $b(u,v) = N(v)$, let $b(u,v) = \emptyset$. If $S(u,v)$ contains all the possible groups, let $S(u,v) = \emptyset$.

One can see from the design of GNRW that, like CNRW, it does not alter the stationary distribution of simple random walk - no matter how the grouping strategy is designed. Also, just like CNRW, GNRW guarantees a smaller or equal asymptotic variance than SRW. This is formalized in the following theorem, the proof of which can be constructed in analogy to Theorem 11.

Theorem 11. *Given a graph $G(V,E)$, GNRW has the stationary distribution $\pi(v) = k_v/2|E|$. And for any property function f, the following two estimators for μ based on SRW ($\hat{\mu}$) and GNRW ($\hat{\mu^*}$):*

$$\hat{\mu}_n = \frac{1}{n}\sum_{t=1}^{n} f(X_t), \quad \hat{\mu}_n^* = \frac{1}{n}\sum_{t=1}^{n} f(X_t^*), \tag{87}$$

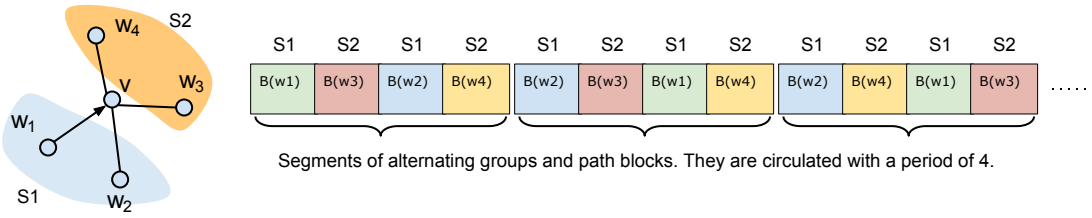


Figure 27: A demo of GNRW

then GNRW will have no greater asymptotic variance than SRW

$$V_\infty(\hat{\mu^*}) \leq V_\infty(\hat{\mu}). \tag{88}$$

Proof.

$$\pi_j^{GNRW} = \lim_{n\to\infty} \frac{1}{n} \sum_{m=1}^{n} Pr\{X_m^{GNRW} = j\} \tag{89}$$

$$= \lim_{n\to\infty} \frac{\sum_i \sum_{B(j)\in S_i} \sum_{X_m \in B(j)} Pr\{X_m^{GNRW} = j\}}{n} \tag{90}$$

$$= \lim_{n\to\infty} \frac{\sum_i \sum_{X_m \in B(i)} Pr\{X_m^{SRW} = j\}}{n} \tag{91}$$

$$= \lim_{n\to\infty} \frac{1}{n} \sum_{m=1}^{n} Pr\{X_m^{SRW} = j\} \tag{92}$$

$$= \pi_j^{SRW}. \tag{93}$$

Eq. (90) $\Rightarrow$ (91) because GNRW will iterate all the path blocks in $N(v) = \cup_i S_i$, and each path block has the same probability to be accessed (S_i is chosen with probability proportional to its size). The rest of the proof is straightforward and similar to CNRW's.

□

In the following discussions, we first describe the rationale behind GNRW, and then discuss the design of the grouping strategy (for the neighbors of v), in other words, the design of groupby function $g(N(v)) = \{S_1, S_2, \ldots, S_m\}$. To understand the rationale,

we start by considering two extremes of the grouping-strategy design. At one extreme is to group neighbors of v in a completely random fashion. With this strategy, GNRW is exactly reduced to CNRW - i.e., every neighbor of v is circulated, with the order being a random permutation. At the other extreme is the ideal scenario for GNRW - as illustrated in Figure 26 - when nodes leading to similar path blocks are grouped together. The intuitive reason why GNRW outperforms CNRW in this case is easy to understand from Figure 26: Circulating among the three groups, instead of attempting a group more than once, can make the random walk "propagate" faster to the entire graph rather than being "stuck" in just one cluster.

More formally, the advantage offered by GNRW can be explained according to the path-block encoding scheme introduced in Section 7.2.2. Consider an example in Figure 27 where v has 4 neighbors. We encode the path block from each neighbor as random variables $B(w_1), \ldots, B(w_4)$. Suppose that neighbors are partitioned into two groups: $S_1 : \{w_1, w_2\}$ and $S_2 : \{w_3, w_4\}$. One can see that GNRW is like stratified sampling, while CNRW is a process of simple random sampling (without replacement). As long as the intra-group variance for the two groups is smaller than the population variance, GNRW offers overall a lower asymptotic variance. To understand why, consider an extreme-case scenario with zero intra-group variance (i.e., $B(w_1) = B(w_2)$, $B(w_3) = B(w_4)$). One can see that, while $GNRW$ achieves zero overall variance, CNRW still has positive variance due to the inter-group variance (i.e., $B(w_1) \neq B(w_3)$).

Having discussed the rationale behind GNRW, we now turn our attention to the design of the grouping strategy. One can see from the above discussion that the main objective here is the group together neighbors that lead to similar path blocks - i.e., random walks starting from nodes in the same group should share similar characteristics. We would like to make two observations for the design: First, *locality* is a property widely recognized for social networks - i.e., users with similar attribute values (e.g., age, occupation) tend to have similar friends (and therefore lead to similar path blocks). Thus, grouping neighbors of v

based on any attribute is likely to outperform the baseline of random group assignments.

Second, which attribute to use for group assignments also has an implication on the potential usage of samples for analytical purposes. For example, if one knows beforehand that samples taken from a social network will be used to estimate the average age of all users, then designing the grouping strategy based on user age is an ideal design that will likely lead to a more accurate estimation of the average age. To understand why, note that with this grouping strategy, the random walk is likely to quickly "propagate" to users of different age groups, instead of being "stuck" at a tight community formed by users of similar ages. Thus, if one knows an important aggregate query which the collected samples will be used to estimate, then the grouping strategy should be designed according to the attribute being aggregated in the query. We shall verify this intuition with experimental results in Section 7.5.2.

7.3.2 Algorithm GNRW

Algorithm implementation. Algorithm 3 depicts the pseudo code for Algorithm GNRW. We note that the data structures we maintain are two *hash maps*: $S(u, v)$ and $b_{S_i}(u, v)$. $S(u, v)$ is a mapping from (u, v) to the current set of groups that GNRW have accessed before based on the route $u \rightarrow v$; $b_{S_i}(u, v)$ is a mapping from (u, v, S_i) to the current set of nodes that GNRW have accessed before based on the route $u \rightarrow v$ and the outgoing group S_i.

Time and space complexity. GNRW requires two hash maps that continuously records the outgoing groups and edges for each edge, i.e. the key-value pairs: $e_{uv} \rightarrow S(u, v)$ and $(e_{uv}, S_i) \rightarrow b_{S_i}(u, v)$. Similar to CNRW, we assume GNRW walks K steps, and it also has the amortized expected $O(K)$ time complexity and $O(K)$ space complexity, because the keys e_{uv} and (e_{uv}, S_i) are uniformly distributed among their possible values [11].

Algorithm 3 Groupby Neighbors Random Walk

/* Given $x_0 = u$, $x_1 = v$, a groupby function $g(\cdot)$ */
/* And we assume that all $S(u, v)$ and $b_{S_i}(u, v)$ should be initialized as $\emptyset$. */
for $i = 2 \rightarrow sample_size$ **do**
  $g(N(x_{i-1}) = \{S_1, S_2, \ldots, S_m\}$
  $CS \leftarrow \{S_1, S_2, \ldots, S_m\} - S(u, v)$
  if $CS \neq \emptyset$ **then**
    $S_i \leftarrow$ choose a group with probability $|S_i|/|CS|$
    $U \leftarrow S_i - b_{S_i}(x_{i-2}, x_{i-1})$
    if $U \neq \emptyset$ **then**
      $x_i \leftarrow$ uniformly choose a node from U
      $b_{S_i}(x_{i-2}, x_{i-1}) = b_{S_i}(x_{i-2}, x_{i-1}) \cup \{x_i\}$
    else
      $x_i \leftarrow$ uniformly choose a node from S_i
      $b_{S_i}(x_{i-2}, x_{i-1}) = \emptyset$
    end if
  else
    $S_i \leftarrow$ uniformly choose a group from $\{S_1, S_2, \ldots, S_m\}$
    $U \leftarrow S_i - b_{S_i}(x_{i-2}, x_{i-1})$
    if $U \neq \emptyset$ **then**
      $x_i \leftarrow$ uniformly choose a node from U
      $b_{S_i}(x_{i-2}, x_{i-1}) = b_{S_i}(x_{i-2}, x_{i-1}) \cup \{x_i\}$
    else
      $x_i \leftarrow$ uniformly choose a node from S_i
      $b_{S_i}(x_{i-2}, x_{i-1}) = \emptyset$
    end if
  end if
end for

7.4 Discussions

CNRW applied to Non-Backtracking Random Walk (NB-SRW) It is important to note that the idea of CNRW - i.e., changing transition upon visiting an edge $u \rightarrow v$ from sampling with replacement to sampling without replacement - is an idea that can be applied to *any* base random walk algorithm, including both SRW and NB-SRW [19]. For example, if we apply the idea to NB-SRW, then the resulting algorithm (say NB-CNRW) will work as follows: Upon visiting $u \rightarrow v$, instead of sampling the next node with replacement from $N(v)\backslash u$ (like in NB-SRW), we would sample it without replacement from $N(v)\backslash u$. Note the difference between NB-CNRW and the CNRW algorithm presented in the paper (which

is based on SRW): With CNRW, the sampling is done over $N(v)$ while with NB-CNRW, the sampling is done over $N(v)\backslash u$ - indeed a carry-over change from NB-SRW.

How does graph size affects CNRW and GNRW. First, we note that the graph size is unlikely to be a main factor in the historic visit probability. To understand why, consider SRW over an undirected graph and the probability of going back to the starting node (which, without loss of generality, is the probability for historic visit to occur). Note that the probability of going back to the starting node at Step i keeps decreasing with i, until the random walk visits the starting node again [7]. Thus, the historic visit probability mainly depends on the first k steps after visiting a node, where k is a small constant. In more intuitive terms, a random walk is mostly likely to go back to a node only a few steps after visiting the node (because, after only a few steps, the random walk is likely still within a "tightly connected" local neighborhood of the node). This essentially means that the historic visit probability is unlikely to be sensitive to graph size - after all, even when the graph size tends to infinity, the probability of visiting (or re-visiting) a node within a constant number of steps is unlikely to change much. As an extreme-case example, growing the graph beyond the k-hop neighborhood of the starting node does not change the historic visit probability within k steps at all.

7.5 Experiments

7.5.1 Experimental Setup

Hardware and platform: We conducted all experiments on a computer with Intel Core i3 2.27GHz CPU and 64bit Ubuntu Linux OS.

Datasets: We tested three types of datasets in the experiments: well-known public benchmark datasets that are small subsets of real-world social networks, large online social networks such as Google Plus and Yelp, and synthetic graphs (for demonstrating extreme-case scenarios) - e.g., barbell graphs and small clustered graphs. We briefly describe the three

types of datasets we used respectively as follows (see the summary of these datasets in Table 1).

	nodes	edges	average degree	average clustering coefficient	number of triangles
Facebook	775	14006	36.14	0.47	954116
Google Plus	240276	30751120	255.96	0.51	2576826580
Yelp	119839	954116	15.92	0.12	4399166
Youtube	1134890	2987624	5.26	0.08	3056386
Clustering graph	90	1707	37.93	0.99	23780
Barbell graph	100	2451	49.02	0.99	39200

Table 1: Summary of the datasets in the experiments.

Public Benchmark:

The Facebook dataset is a public benchmark dataset collected from [1]. It is a previously-captured topological snapshot of Facebook and it has been extensively used in the literature (e.g., [25]). Specifically, the graph we used is from the "1684.edges" file. Youtube is another large public benchmark graph collected from [36]. In these public benchmark dataset, we simulated a restricted-access web interface, and ran our algorithms over the simulated interface.

Large Online Social Graphs:

Google Plus[13]. To test the scalability of our algorithms over a large graph, we performed experiments over a large graph we crawled from Google Plus that consists of 240,276 users. We observe that the interface provided by Google Social Graph API strictly adheres to our access model - i.e., each API request returns the local neighborhood of one user.

Yelp dataset[14]. We extracted the largest connected subgraph containing 119,839 users (out of 252,898 users) from the dataset. We restored all the dumped JSON data into MongoDB to simulate API requests.

Since we focus on sampling undirected graphs in this paper, for datasets that feature directed graphs, we first converted it to an undirected one by only keeping edges that appear

[13]https://plus.google.com/

[14]http://www.yelp.com/dataset_challenge

in both directions in the original graph. Note by following this conversion strategy, we guarantee that a random walk over the undirected graph can also be performed over the original directed graph, with an additional step of verifying the existence of the inverse direction (resp. $v \rightarrow u$) before committing to an edge (resp. $u \rightarrow v$) in the random walk.

Synthetic Graphs:

We also tested our algorithms over synthetic graphs, such as barbell graphs and graphs with high clustering coefficients for two main purposes: One is to demonstrate the performance of our algorithms over "ill-formed" graphs as these synthetic graphs have very small conductance (i.e., highly costly for burning in). The other is to control graph parameters such as number of nodes that we cannot directly control over the above-described real-world graphs. It is important to note that our usage of a theoretical graph generation model does not indicate a belief of the model being a representation of real-world social network topology.

Algorithms: We tested five algorithms in the experiments: Simple Random Walk (SRW) [21], Metropolis-Hastings Random Walk (MHRW) [15], Non-Backtracking Simple Random Walk (NB-SRW) [19] - a state-of-the-art random walk algorithm which uses an order-2 Markov Chain, and two algorithms proposed in this paper: Circulated Neighbors Random Walk (CNRW) in Section 7.2 and Groupby Neighbors Random Walk (GNRW) in Section 7.3. For each algorithm, we ran it with a query budget ranging from 20 to 1000, and take the returned sample nodes to measure their quality (see performance measures described below). For GNRW, we tested various grouping strategies, as elaborated in Section 7.5.2.

It is important to understand why we included MHRW in the algorithms for testing. Note that, while SRW, NB-SRW and our two algorithms all share the same (target) sampling distribution - i.e., each node is sampled with probability proportional to its degree - MHRW has a different sampling distribution - i.e., the uniform distribution. Thus, it is impossible to compare the samples returned by MHRW with the other algorithms properly.

We note that our purpose of including MHRW here is to simply verify what has been recently shown in the literature [14] and [19] - i.e., for practical purposes such as aggregate estimation over social networks, the performance of MHRW is much worse than the other SRW based algorithms, justifying the usage of SRW as a baseline in our design.

Performance Measures. Recall from Section 2.2 that a sampling algorithm for online social networks should be measured by *query cost* and *bias* - i.e., the distance between the actual sampling distribution and the (ideal) target one, which in our case is $\pi(v) = k_v/(2|E|)$. To measure the query cost, one simply counts the number of unique queries issued by the sampler. The measurement of bias, on the other hand, requires us to consider two different methods (and three measures) described as follows.

For a small graph, we measured bias by running the sampler for an extremely long amount of time (long enough so that each node is sampled multiple times). We then estimated the sampling distribution by counting the number of times each node is retrieved, and compared this distribution with the target distribution to derive the bias. Their distances are measured in two forms: (1) KL-divergence [10], and (2) ℓ_2-distance [10] between the two distribution vectors. Let P and P_{sam} be the ideal and measured sampling distribution vectors, respectively.

- To measure the distance in KL-divergence, we compute $D_{\text{KL}}(P||P_{\text{sam}}) + D_{\text{KL}}(P_{\text{sam}}||P)$ where

$$D_{\text{KL}}(P||Q) = \sum_{v_i \in V} \ln\left(\frac{P(v_i)}{Q(v_i)}\right) P(v_i), \tag{94}$$

- For ℓ_2-norm, we use $\|P - P_{\text{sam}}\|_2$.

Note that compared with the KL-divergence based measure, the ℓ_2-norm one is more sensitive to "outliers" - i.e., large differences on the sampling probability for a single node - hence our usage of both measures in the experiments.

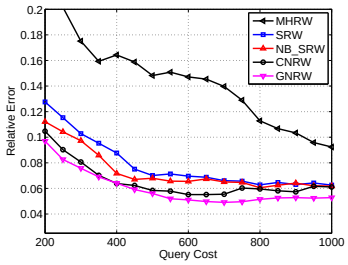
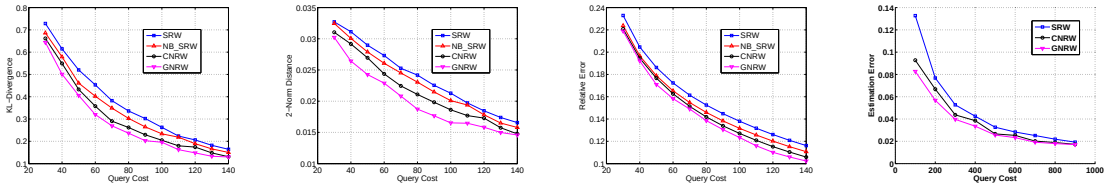


Figure 28: Large Google Plus Graph: estimation of average degree.

For a large graph like Google Plus and Yelp we used, it is no longer feasible to directly measure the sampling probability distribution (because sampling each node multiple times to obtain a reliable estimation becomes prohibitively expensive). Thus, we considered another measure of bias, *aggregate estimation error*, for experiments over large graphs. Specifically, we first used the collected samples to estimate an aggregate over all nodes in the graph - e.g., the average degree or reviews count - and then compare the estimation with the ground truth. One can see that, since SRW, NB-SRW and both of our algorithms all share the exact same target sampling distribution, a sampler with a smaller bias tends to produce an estimation with lower relative error - justifying our usage of it in the experiments.

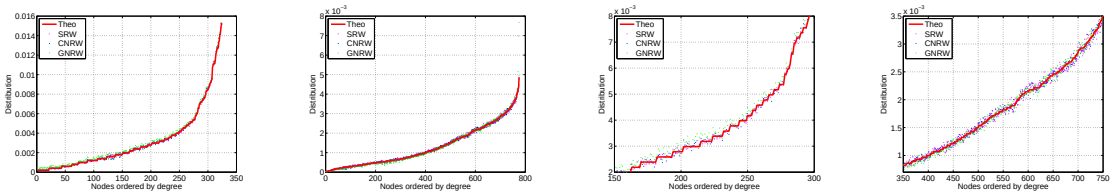
7.5.2 Evaluation



(a) Facebook KL-divergence (b) Facebook ℓ_2-distance (c) Facebook Estimation error (d) Youtube Estimation error

Figure 29: Public benchmark datasets

We start with the experiments that show how SRW, CNRW and GNRW have the same stationary distribution. In Figure 30, we ran 100 instances of each random walk for 10000



(a) facebook dataset 1 (b) facebook dataset 2 (c) facebook dataset 1 (zoomed) (d) facebook dataset 2 (zoomed)

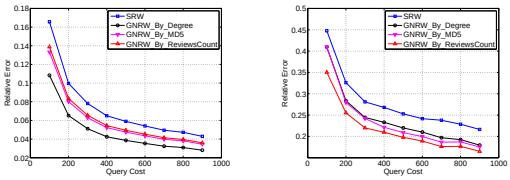
Figure 30: Distribution of SRW, CNRW and GNRW

steps in two datasets (both are from the Facebook dataset [1]), and then we used the samples collected from each random walk to calculate the sampling distribution. We ordered the distribution of the nodes by the their degree, and we also included the theoretical distribution of all the nodes (i.e. the red solid line in the figure). One can see that all the three random walks converge to the same stationary distribution.

We then compare of all five algorithms over the Google Plus dataset, with bias measure being the relative error for estimating the average degree. Figure 28 depicts the change of relative error with query cost. One can make two observations from the figure. One, our proposed algorithms CNRW and GNRW significantly outperform the other algorithms. For example, to achieve a relative error of 0.06, CNRW and GNRW only requires a query cost of around 486 and 447, respectively, while SRW requires over 800, NB-SRW requires 795, and MHRW never achieves a relative error under 0.08 after issuing 1000 queries. Second, we can also observe from the figure that MHRW performs much poorer than the other algorithms. Thus, we do not further include MHRW in experimental results in the paper.

With Figure 28 establishing the superiority of our algorithms over the existing ones with the relative error measure, we also confirmed the superiority with the other two measures, KL-divergence and ℓ_2-norm, this time over the public benchmark dataset. Figure 29 depicts the results for Facebook and Youtube. One can observe from Figure 29 that our CNRW and GNRW algorithms consistently outperform both SRW and SRW according to all three measures being tested. In addition, GNRW outperforms CNRW, also for all three measures.

To further study the design of GNRW, specifically the criteria for grouping nodes to-

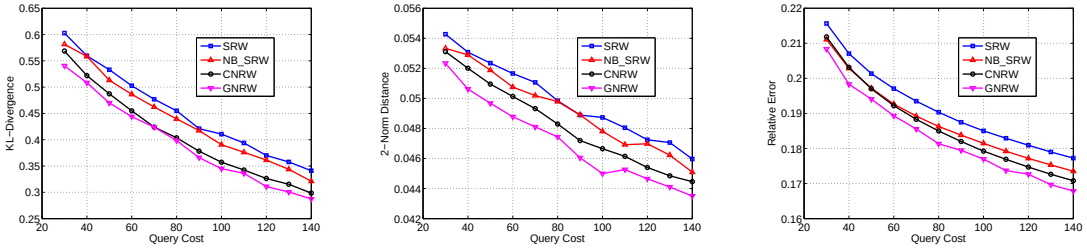


(a) Estimate Average Degree (b) Estimate Average Reviews Count

Figure 31: Yelp dataset: GNRW strategies.

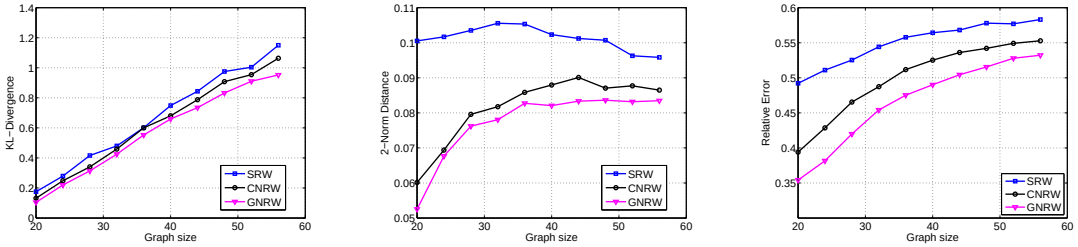
gether, we tested GNRW with three different grouping strategies: random grouping (i.e., GNRW-By-MD5, as we group nodes together according to the MD5 of their IDs), grouping by similar degrees (GNRW-By-Degree), and grouping by the value of an attribute "reviews count" (GNRW-By-ReviewsCount). Figure 31 depicts the performance of all three strategies. One can make an interesting observation from the figure: While all three variations significantly outperform the baseline SRW algorithm, the best-performing variation indeed differs when the relative error is computed from different aggregates. Specifically, when the aggregate is average degree, GNRW-By-Degree performs the best. When the aggregate is average review count, on the other hand, GNRW-By-ReviewsCount performs the best. This verifies our discussions in Section 7.3.1 - i.e., if the aggregate of interest is known before hand, choosing the grouping strategy in alignment with the aggregate of interest can lead to more accurate aggregate estimations from samples.

Finally, we studied the performance of our algorithms over two "ill formed" graphs, the clustered graph and a barbell graph. The clustered graph is combined with 3 complete graphs with graph sizes as 10, 30 and 50. We also vary the size of the barbell graph from 20 to 56 nodes, to observe the change of performance of the algorithms with graph size. The results are shown in Figures 32 and 33. One can see from the results that, even for these ill-formed graphs, our proposed algorithms consistently outperform SRW and NB-SRW for varying graph sizes, according to all three bias measures being used.



(a) KL-divergence (b) ℓ_2-distance (c) Estimation error

Figure 32: Synthetic datasets: clustered graph



(a) KL-divergence (b) ℓ_2-distance (c) Estimation error

Figure 33: Barbell graph size analytics.

Chapter 8 - Conclusion

We have systematically designed three methods of faster sampling on online social networks.

For shorting the mixing time of random walks, we have initiated a study of enabling faster random walk over an online social network (with a restrictive web interface) by "rewiring" the social network on-the-fly. We showed that the key for speeding up a random walk is to increase the conductance of the graph topology followed by the random walk. As such, we developed MTO-Sampler which provably increases the graph conductance by constructing an overlay topology on-the-fly through edge removals and replacements. We provided theoretical analysis and extensive experimental studies over real-world social networks to illustrate the superiority of MTO-Sampler on achieving a smaller sampling bias while consuming a lower query cost.

For eliminating the mixing time of random walks, we developed WALK-ESTIMATE, a

general purpose technique for faster sampling of nodes over an online social network with any target (sampling) distribution. Our key idea is to conduct a random walk for a small number of steps, and follow it with a proactive estimation of the sampling distribution of the node encountered before applying acceptance-rejection sampling to achieve the target distribution. Specifically, we presented two main components of WALK-ESTIMATE, WALK which determines the number of steps to walk, and ESTIMATE which enables an unbiased estimation of the sampling distribution. Theoretical analysis and extensive experimental evaluations over synthetic graphs and real-world online social networks demonstrated the superiority of our technique over the existing random walks.

For leveraging history of random walks, we considered a novel problem of leveraging historic transitions in the design of a higher-ordered MCMC random walk technique, in order to enable more efficient sampling of online social networks that feature restrictive access interfaces. Specifically, we developed two algorithms: (1) CNRW, which replaces the memoryless transition in simple random walk with a memory-based, sampling-without-replacement, transition design, and (2) GNRW, which further considers the observed attribute values of neighboring nodes in the transition design. We proved that while CNRW and GNRW achieve the exact same target (sampling) distribution as traditional simple random walks, they offer provably better (or equal) efficiency no matter what the underlying graph topology is. We also demonstrated the superiority of CNRW and GNRW over baseline and state-of-the-art sampling techniques through experimental studies on multiple real-world online social networks as well as synthetic graphs.

References

[1] Stanford large network dataset collection http://snap.stanford.edu/data/.

[2] E. M. Airoldi. Sampling algorithms for pure network topologies. *SIGKDD Explorations*, 7:13–22, 2005.

[3] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6:83–96, 1986. 10.1007/BF02579166.

[4] N. Alon, C. Avin, M. Koucky, G. Kozma, Z. Lotker, and M. R. Tuttle. Many random walks are faster than one. In *SPAA*, 2008.

[5] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna. Four degrees of separation. In *Proc. of the 3rd Annual ACM Web Science Conf.*, pages 33–42. ACM, 2012.

[6] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[7] R. Burioni and D. Cassi. Random walks on graphs: ideas, techniques and results. *Journal of Physics A: Mathematical and General*, 38(8):R45, 2005.

[8] M. K. Cowles and B. P. Carlin. Markov chain monte carlo convergence diagnostics: A comparative review. *Journal of the American Statistical Association*, 91:883–904, 1996.

[9] A. Dasgupta, G. Das, and H. Mannila. A random walk approach to sampling hidden databases. In *SIGMOD*, 2007.

[10] Y. Dodge, D. Cox, D. Commenges, P. J. Solomon, S. Wilson, et al. *The Oxford dictionary of statistical terms*. Oxford University Press, 2003.

[11] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with 0 (1) worst case access time. *Journal of the ACM (JACM)*, 31(3):538–544, 1984.

[12] C. J. Geyer. Practical markov chain monte carlo. *Statistical Science*, 1992.

[13] W. R. Gilks. *Markov Chain Monte Carlo In Practice*. Chapman and Hall/CRC, 1999.

[14] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. In *INFOCOM*, 2010.

[15] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[16] 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 *MobiArch*, 2011.

[17] L. Katzir, E. Liberty, and O. Somekh. Estimating sizes of social networks via biased sampling. In *WWW*, 2011.

[18] M. Kurant, M. Gjoka, C. T. Butts, and A. Markopoulou. Walking on a graph with a magnifying glass: stratified sampling via weighted random walks. In *SIGMETRICS*, 2011.

[19] C.-H. Lee, X. Xu, and D. Y. Eun. Beyond random walk and metropolis-hastings samplers: why you should not backtrack for unbiased graph sampling. *Sigmetrics Performance Evaluation Review*, pages 319–330.

[20] C.-H. Lee, X. Xu, and D. Y. Eun. Beyond random walk and metropolis-hastings samplers: Why you should not backtrack for unbiased graph sampling. SIGMETRICS, 2012.

[21] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *SIGKDD*, 2006.

[22] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2008.

[23] R.-H. Li, J. X. Yu, X. Huang, and H. Cheng. Random-walk domination in large graphs. In *ICDE*, 2014.

[24] L. Lovász. Random walks on graphs: A survey. *Combinatorics, Paul Erdos is Eighty*, 2(1):1–46, 1993.

[25] J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. In *NIPS*, 2012.

[26] S. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. Cambridge University Press, 2nd edition, 2009.

[27] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *SIGCOMM*, 2007.

[28] A. Mohaisen, A. Yun, and Y. Kim. Measuring the mixing time of social graphs. In *SIGCOMM*, 2010.

[29] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *SIGMOD*, 2008.

[30] A. Nazi, Z. Zhou, S. Thirumuruganathan, N. Zhang, and G. Das. Walk, not wait: Faster sampling over online social networks. *CoRR*, abs/1410.7833, 2014.

[31] R. M. Neal. Improving asymptotic variance of mcmc estimators: Non-reversible chains are better. *Technical Report*, 2004.

[32] B. Ribeiro and D. Towsley. Estimating and sampling graphs with multidimensional random walks. In *SIGCOMM*, 2010.

[33] F. Spitzer, F. Spitzer, F. Spitzer, and A. Mathematician. *Principles of random walk*. Springer, 1964.

[34] S. Thirumuruganathan, N. Zhang, V. Hristidis, and G. Das. Aggregate estimation over a microblog platform. In *SIGMOD*, 2014.

[35] X. Yan, B. He, F. Zhu, and J. Han. Top-K aggregation queries over large networks. In *ICDE*, 2010.

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

[37] N. Zhang and G. Das. Exploration of deep web repositories. In *Proc. of the VLDB Endowment (VLDB), Tutorial*, 2011.

[38] N. Zhang and G. Das. Exploration and mining of web repositories. In *WSDM*, 2014.

[39] Z. Zhou, N. Zhang, Z. Gong, and G. Das. Faster random walks by rewiring online social networks on-the-fly. ICDE, 2013.

[40] Z. Zhou, N. Zhang, Z. Gong, and G. Das. Faster random walks by rewiring online social networks on-the-fly. In *ICDE*, 2013.

[41] Z. A. Zhu, S. Lattanzi, and V. S. Mirrokni. A local algorithm for finding well-connected clusters. *CoRR*, 2013.