

January 2015

Scaling Up Network Analysis and Mining: Statistical Sampling, Estimation, and Pattern Discovery

Nesreen Kamel Ahmed
Purdue University

Follow this and additional works at: https://docs.lib.purdue.edu/open_access_dissertations

Recommended Citation

Ahmed, Nesreen Kamel, "Scaling Up Network Analysis and Mining: Statistical Sampling, Estimation, and Pattern Discovery" (2015). *Open Access Dissertations*. 1445.
https://docs.lib.purdue.edu/open_access_dissertations/1445

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries.
Please contact epubs@purdue.edu for additional information.

**PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance**

This is to certify that the thesis/dissertation prepared

By Nesreen Kamel Ahmed

Entitled
Scaling Up Network Analysis and Mining: Statistical Sampling, Estimation, and Pattern Discovery

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

<u>Jennifer Neville</u>	_____
<small>Chair</small>	_____
<u>Christopher W. Clifton</u>	_____
<u>Walid G. Aref</u>	_____
<u>Sonia Fahmy</u>	_____

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Jennifer Neville

Approved by: Sunil Prabhakar / William J. Gorman 07/09/2015
Head of the Departmental Graduate Program Date

SCALING UP NETWORK ANALYSIS AND MINING:
STATISTICAL SAMPLING, ESTIMATION, AND PATTERN DISCOVERY

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Nesreen K. Ahmed

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2015

Purdue University

West Lafayette, Indiana

*To my mother and the loving memory of my father
They are the ones who made me who I am today*

ACKNOWLEDGMENTS

I am extremely thankful and honored to be surrounded by so many exceptional and inspirational people. I believe words are not enough for expressing my gratitude to them. First and foremost, I am extremely fortunate to have my PhD advisor Jennifer Neville, for her continuous support, insightful feedback, invaluable advice over the years of my PhD journey, and for challenging me and encouraging me to find my way as a researcher. I can never thank my PhD committee members enough for their support and guidance. I am very grateful to Sonia Fahmy, Chris Clifton, and Walid Aref, for their invaluable feedback and advice that improved my dissertation and their encouragement over the years. I am thankful for them for always finding new ways to consider a problem.

I have had enjoyable, rewarding discussions, and collaborations with many professors during my PhD. I was very fortunate to work with Ramana Kompella, and I am extremely grateful for his guidance that ultimately helped me become a better researcher. I am particularly indebted and honored to have the opportunity to work with Nick Duffield. Nick has been a great mentor to me and I learned a lot from him. I can never thank Nick enough for his continuous guidance, support, and encouragement.

I specially like to thank Ahmed Elmagarmid, Greg Frederickson, Alan Qi, SVN Vishwanathan, Luo Si, David Gleich, and Susanne Hambruch for all their support and guidance. I am very grateful for Mohammad Al Hasan for his kind nature, advice, and collaboration. Special thanks to Dr. Groman for his support, advice, and guidance during my PhD.

Special thanks to Sebastian Moreno, Dan Zhang, Mohamed Yakout, Tao Wang, Hyokun Yun, Rongjing Xiang, Hoda Eldardiry, Timothy La Fond, Joel Pfeiffer, Hogun Park, Iman Alodah, Praveen Kumar, Camille Gaspard, Sait Celebi, Suvidha Kancharla, Ellen Lai, Giselle Zeno, Pablo Granda, Jiasen Yang, Jason Meng, Ahmed Aly, and many others, for the interesting discussions and the great times during conferences and/or meetings.

I feel very grateful to have had the opportunity to work closely with top researchers in a variety of industrial labs. I am especially fortunate and honored to have the opportunity

to work with Ayman Farahat at Adobe Advanced Technology Labs. He has been a great mentor to me and I am especially thankful for his often selfless support, encouragement, and advice over the years.

In addition, I am especially fortunate to Robert Kuhn, my mentor at Intel, for his kind nature, advice, and encouragement. Robert helped me to pursue new applications for data mining beyond the traditional ones. I would also like to thank Al Mamunur Rashid for his friendship, encouragement, and collaboration during my time at Intel.

During my time at Facebook, I am extremely fortunate to have had the opportunity to be mentored by Rajiv Krishnamurthy. I am especially thankful for his help and encouragement that gave me the strength to overcome many challenging problems that I encountered at Facebook and beyond.

I feel eternally indebted to my early research advisor, Amir Atiya, who not only introduced me to research in data mining and machine learning, but has significantly changed my life in the process. Amir has been a great source of inspiration and true role model as a researcher, mentor, and friend. I thank him for his relentless and selfless advice to tackle difficult problems. His caring nature always gave me the strength to move forward even during the tough times.

During my time at Purdue, I have built friendships that I hold closely to my heart. I thank Nilothpal Talukder, Balamurugan Anandan, Hani Jibrin, Sohayla Jibrin, Lamis Be, Ali Roumani, Muna Albasman, Sreen Al-Khalili, Ahmed Abdelhamid and many others. I am indebted to Ryan Rossi for his friendship, encouragement, and collaboration.

Most of all, I owe my deepest gratitude to my family for their endless love and encouragement. Everything was possible due to their strong support. I dedicate this work to my parents. My father has been always a continuous source of moral, joy, and support in my life. I believe he is still with me and I hope I make him proud. My mother has been so selfless in supporting me at all phases of my life and career. I am indebted to my sister Amany and brothers Mohamed and Mustafa for their continuous support and encouragement over the years. I have the best family anyone could ask for, and words will never be sufficient for expressing my gratitude to my family.

TABLE OF CONTENTS

	Page
LIST OF TABLES	ix
LIST OF FIGURES	xi
ABSTRACT	xiii
1 INTRODUCTION	1
1.1 Challenges of Network Analysis and Mining	1
1.2 A Taxonomy of Network Sampling Methods	3
1.2.1 Classes of Network Sampling Methods	3
1.2.2 Spectrum of Computational Models	4
1.3 Problem Statement	8
1.3.1 Sampling from Large Static Graphs	10
1.3.2 Sampling from Streaming Graphs	11
1.3.3 Big Graph Analytics and Unbiased Estimation	12
1.4 Contributions and Outline	13
2 BACKGROUND	15
2.1 Foundations and Notations	16
2.2 Goals, Units, and Population of Networks	17
2.3 Classes of Sampling Methods	20
2.4 Evaluation of Sampling Methods	22
2.5 Models of Computation	27
2.6 Review of Related work	31
2.6.1 Network Sampling in Social Science	31
2.6.2 Statistical Properties of Network Sampling	32
2.6.3 Network Sampling in Networked Systems	32
2.6.4 Network Sampling in Structured Data Mining	33

	Page
2.6.5 Graph Streams	34
3 SAMPLING FROM LARGE STATIC GRAPHS	36
3.1 Motivation	36
3.2 Two-Pass Stream Sampling	37
3.3 Analysis of Sampling Bias	39
3.3.1 Selection Bias Toward High Degree Nodes	39
3.3.2 Downward Bias Due to Sampling	41
3.4 Experiments	44
3.4.1 Distance Metrics	45
3.4.2 Statistical Distributions	46
3.4.3 Comparison to Metropolis Graph Sampling	48
3.5 Network Sampling Designs for Relational Classification	53
3.5.1 Impact on Parameter Estimation	54
3.5.2 Impact on Classification Accuracy	57
3.6 Summary	59
4 SAMPLING FROM STREAMING GRAPHS	61
4.1 Motivation	61
4.2 Streaming Node Sampling	62
4.3 Streaming Edge Sampling	63
4.4 Streaming Topology-Based Sampling	64
4.5 Partially-Induced Edge Sampling (PIES)	65
4.6 Experiments	68
4.6.1 Distance Metrics	68
4.6.2 Statistical Distributions	69
4.6.3 Analysis of Dense Versus Sparse Graphs	75
4.6.4 Analysis of Isolated Nodes	75
4.7 Sampling from Multigraph Streams	78
4.7.1 Kolmogorov-Smirnov Statistic	79

	Page
4.7.2	Statistical Distributions of Temporal Properties 80
4.7.3	Interplay between Graph Dynamics and Structure 82
4.7.4	Randomization Tests for Graph Streams 84
4.8	Summary 85
5	SAMPLE & HOLD: A FRAMEWORK FOR BIG GRAPH ANALYTICS 86
5.1	Motivation 86
5.2	Relation to Classic Sample and Hold 89
5.3	Framework for Graph Sampling 90
5.3.1	Graph Stream Model 90
5.3.2	Edge Sampling Model 90
5.3.3	Subgraph Estimation 91
5.4	Unbiased Estimation 93
5.4.1	General Estimation and Variance 94
5.4.2	Edges 95
5.4.3	Triangles 95
5.4.4	Connected Paths of Length 2 96
5.4.5	Clustering Coefficient 96
5.4.6	Nodes 97
5.5	Graph Sample and Hold 97
5.5.1	Algorithms 97
5.5.2	Illustration with $\text{gSH}(p,1)$ 99
5.6	Experiments 100
5.6.1	Performance Analysis 101
5.6.2	Confidence Bounds 102
5.6.3	Comparison to Previous Work 106
5.6.4	Effect of p, q on Sampling Rate 108
5.6.5	Implementation Issues 109
5.7	Related Work 111

	Page
5.8 Summary	113
6 FAST PARALLEL MOTIF COUNTING FOR LARGE GRAPHS	115
6.1 Motivation	115
6.2 Motifs, Scalability, Applications	116
6.3 Background	118
6.3.1 Notations and Definitions	118
6.3.2 Relation to Graph Complement	121
6.3.3 Relation to Graph & Matrix Reconstruction Theorems . . .	121
6.4 Framework	122
6.4.1 Searching Edge Neighborhoods	122
6.4.2 Counting Motifs of Size ($k = 3$) Nodes	123
6.5 Counting Motifs of Size ($k = 4$) Nodes	126
6.5.1 Motif State Transition Diagram	127
6.5.2 General Principle for Counting Motifs of size $k = 4$	128
6.5.3 Analysis & Combinatorial Arguments	131
6.5.4 Algorithm	144
6.6 Experiments & Applications	145
6.6.1 Scalability & Runtime	146
6.6.2 Large-Scale Graph Comparison & Classification	149
6.6.3 Finding Large Stars, Cliques, and Other Patterns	150
6.7 Summary	152
7 SUMMARY AND CONCLUSION	163
7.1 Contributions	164
7.2 Future Directions	166
REFERENCES	168
VITA	181

LIST OF TABLES

Table	Page
2.1 Description of Network Statistics	23
3.1 Characteristics of Network Data Sets	44
3.2 Comparison of the Maximum Kcore Number for Static Graphs	47
3.3 Comparison to Neighbor Reservoir Sampling	52
4.1 Comparison of the Maximum Kcore Number for Streaming Graphs	71
4.2 Comparison of Percentage of Isolated Nodes for All Sampling Algorithms	76
4.3 Average KS Distance for Stream Sampling Methods.	77
4.4 Average L1/L2 Distance for Stream Sampling Methods.	77
4.5 Characteristics of Multigraph Data Sets	78
5.1 Estimation on a Path of Length 3	99
5.2 Statistics of Data Sets	100
5.3 Estimated Properties using Graph Sample & Hold	103
5.4 Coverage Probability for 95% Confidence Interval	106
5.5 Comparison to Streaming Triangles	107
5.6 Runtime for Sampling and Estimation using gSH	110
6.1 Summary of Motif Notation and Properties	120
6.2 Runtime & Statistics for a Subset of 55 Networks	147
6.3 Accuracy of Graph Classification	150
6.4 Statistics of Facebook100 Networks	154
6.5 Statistics of Facebook100 Networks (Table 6.4 continued)	155
6.6 Statistics of Biological, Co-authorship & Interaction Networks	156
6.7 Statistics of Infrastructure, Strong Components & Social Networks	157
6.8 Statistics of Technological, Retweet, & Web Networks	158
6.9 Statistics of DIMACS Networks	159

Table	Page
6.10 Statistics of DIMACS Networks (Table 6.9 continued)	160
6.11 Statistics of Biological D& D Networks	161
6.12 Statistics of Chemical MUTAG Networks	162

LIST OF FIGURES

Figure	Page
1.1 Spectrum of Computational Models	6
2.1 Illustration of Graph Streams	29
3.1 Illustration of Original versus Sampled Degrees	43
3.2 Average Distance Across Six Static Graphs	46
3.3 Sampling Distribution of Facebook Graph	49
3.4 Sampling Distribution of HepPH Graph	49
3.5 Sampling Distribution of CondMAT Graph	49
3.6 Sampling Distribution of Twitter Graph	50
3.7 Sampling Distribution of Email-University Graph	50
3.8 Sampling Distribution of Flickr Graph	50
3.9 Sampling Distribution of LiveJournal Graph	51
3.10 Estimation of Class Prior	56
3.11 Classification Accuracy Versus Sampling Fraction	58
3.12 Classification Accuracy Versus Proportion of Labeled Nodes	59
4.1 Average Distance Across Six Streaming Graphs	69
4.2 Stream Sampling Distribution of Pokec Graph	71
4.3 Stream Sampling Distribution of Facebook Graph	71
4.4 Stream Sampling Distribution of HepPH Graph	72
4.5 Stream Sampling Distribution of CondMAT Graph	72
4.6 Stream Sampling Distribution of Twitter Graph	72
4.7 Stream Sampling Distribution of Email-Univ Graph	73
4.8 Stream Sampling Distribution of Flickr Graph	73
4.9 Stream Sampling Distribution of LiveJournal Graph	73
4.10 Stream Sampling Distribution of Youtube Graph	74

Figure	Page
4.11 Stream Sampling Distribution of Web-Stanford Graph	74
4.12 Average KS Statistics for Dense and Sparse Graphs	75
4.13 KS distance as a Function of Stream Length	80
4.14 Distributions of Node Strength	81
4.15 Distributions of Edge Weight	81
4.16 Analysis of Node Strength Versus Node Degree	83
4.17 Randomization Tests for Multigraph Streams	84
5.1 Convergence Analysis of Graph Sample & Hold	105
5.2 Analysis of Sample and Hold Probabilities	109
6.1 4-node Motif Transition Diagram	127
6.2 Illustration of Edge Neighborhood	129
6.3 Runtime of Exact Motif Counting	148
6.4 Anomaly Detection in Facebook University Networks	150
6.5 Visualization of the Human Diseasesome Network	151

ABSTRACT

Ahmed, Nesreen K. PhD, Purdue University, August 2015. Scaling Up Network Analysis and Mining: Statistical Sampling, Estimation, and Pattern Discovery. Major Professor: Jennifer Neville.

Network analysis and graph mining play a prominent role in providing insights and studying phenomena across various domains, including social, behavioral, biological, transportation, communication, and financial domains. Across all these domains, networks arise as a natural and rich representation for data. Studying these real-world networks is crucial for solving numerous problems that lead to high-impact applications. For example, identifying the behavior and interests of users in online social networks (e.g., viral marketing), monitoring and detecting virus outbreaks in human contact networks, predicting protein functions in biological networks, and detecting anomalous behavior in computer networks. A key characteristic of these networks is that their complex structure is massive and continuously evolving over time, which makes it challenging and computationally intensive to analyze, query, and model these networks in their entirety. In this dissertation, we propose sampling as well as fast, efficient, and scalable methods for network analysis and mining in both static and streaming graphs.

We develop a generic framework for statistical network stream sampling, called graph sample and hold. We formulate network sampling as a principled approach with two main functions: (1) the sampling function, and (2) the holding function, this approach allows tuning the sampling and estimation of graph properties more efficiently and accurately than the state-of-the-art. We develop a suite of algorithms to sample and estimate various graph properties, while processing the graph sequentially as a stream of edges. Finally, we develop a fast parallel algorithm for counting motifs, which is 460 times faster than the state-of-the-art. We show how these motif patterns can be used as features to benefit various machine learning tasks such as large-scale graph classification, prediction, anomaly detection, and visual analytics.

1. INTRODUCTION

Network analysis and graph mining play a prominent role in providing insights and studying phenomena across various domains, including social, behavioral, biological, transportation, communication, and financial domains. Across all these domains, networks arise as a natural and rich data representation. Studying these networks is crucial for solving numerous problems that lead to high-impact applications. For example, consider online activity and interaction networks formed from electronic communication (e.g., email, SMS, IMs), social media (e.g., Twitter, blogs, web pages), and content sharing (e.g., Facebook, Flickr, Youtube). These social tools produce a prolific amount of continuous and interaction data (e.g., Facebook users post 3.2 billion likes and comments every day [1]) that is naturally represented as a dynamic network—where the nodes are people or objects and the edges are the interactions among them.

Modeling and analyzing large dynamic networks have become increasingly important for many applications. For example, identifying the behavior and interests of users in online social networks (e.g., viral marketing, online advertising) [2, 3], monitoring and detecting virus outbreaks in human contact networks [4], predicting protein functions in biological networks [5], and detecting anomalous behavior in computer networks [6–8].

1.1 Challenges of Network Analysis and Mining

Many factors make it difficult and computationally intensive to study, analyze, query, or model these networks in their entirety [9–11]. First and foremost, the sheer size of many networks makes it computationally infeasible to study the entire network. Moreover, some networks are not completely visible to the public (e.g., Facebook),

can only be accessed through crawling (e. g., World Wide Web) [12], or their structure is dynamically changing over time (e. g., Twitter) [11, 13]. In other cases, the size of the full network may not be as large but the measurements required to observe the underlying network are costly (e. g., experiments in biological networks [14]). Thus, network sampling is at the heart and foundation of network analysis and mining—since researchers typically need to select a (tractable) subset of the nodes and edges from which to make inferences about the full network.

One key stumbling block for enabling large-scale graph analytics is the limitation in computational resources. Despite the recent advances in distributed and parallel processing frameworks for graph analytics (e.g. MapReduce), and the appearance of infinite resources in the cloud, running brute-force graph analytics is either too costly, too slow, or too inefficient in many practical situations [15–19]. This necessitates the development of fast, efficient, and approximation methods that exploit the characteristics of real-world networks to provide accurate, real-time analytics.

In many situations, finding an *approximate* answer is usually sufficient for many types of analysis tasks, in which the extra cost and time in finding the *exact* answer is often not worth the extra accuracy [20, 21]. In these cases, sampling provides an attractive approach to quickly and efficiently find an approximate answer to a query, or more generally, any graph analysis task.

Sampling has a long history for being efficient and useful to reduce storage requirements [22], speed up query execution times [23], and ensure data privacy by processing only a sample of the data from which to make inferences about the data population [24]. From peer-to-peer to social networks, sampling arises across many different settings [25–29]. For example, sampled networks may be used in simulations and experimentation—to measure performance before deploying new protocols and systems in the field, such as new Internet protocols, social/viral marketing schemes, A/B testing, and/or fraud detection algorithms.

Furthermore, many of the network data sets currently being analyzed as complete networks are themselves samples due to the above limitations in data collection [30,

31]. Thus, it is critical that researchers understand the impact of various sampling methods on the structure of the constructed networks. All of these factors motivate the need for a more refined and complete understanding of *network sampling*.

Although a large body of research has developed methods to sample from networks [25–29], much of the work is problem-specific, and there has been less work focused on developing a broader foundation for network sampling. More specifically, it is often not clear when and why particular sampling methods are appropriate. This is because the goals and population are often not explicitly defined or stated up front, which makes it difficult to evaluate the quality of the recovered samples for other applications. One of the primary aims of this work is to define the foundations of network sampling more explicitly, such as objectives/goals, population of interest, units, classes of sampling methods (i. e., node, edge, and topology-based), and techniques to evaluate a sample (e. g., network statistics and distance metrics).

In this chapter, we start in Section 1.2 by introducing a taxonomy for network sampling methods. Next, in Section 1.3, we highlight the key components and research questions that we address in this work. Finally, in Section 1.4, we summarize the main contributions and outline the structure of this dissertation.

1.2 A Taxonomy of Network Sampling Methods

Given a graph $G = (V, E)$ as an input, how to sample a subgraph $G_s = (V_s, E_s)$ with a subset of the nodes ($V_s \subset V$) and/or edges ($E_s \subset E$) from the population graph G ? The goal is to ensure that G_s is *representative*, in the sense that graph properties of interest are preserved in G_s (or can be estimated from G_s).

1.2.1 Classes of Network Sampling Methods

Network Sampling methods can be generally classified as node, edge, and topology-based sampling methods, based on whether nodes or edges are first selected from the graph G (node or edge-based sampling) or if the selection of nodes and edges depends

more on the existing topology of G (topology-based sampling). More precisely, we define the three classes as follows:

- **Node-based Sampling** – Nodes are sampled with some probability p . For example, in uniform node sampling, nodes are chosen independently and uniformly at random from G for inclusion in the sampled subgraph G_s , and subsequently edges that appear among these nodes are also added to G_s .
- **Edge-based Sampling** – Edges are sampled with some probability p . For example, in uniform edge sampling, edges are chosen independently and uniformly at random from G for inclusion in the sampled subgraph G_s .
- **Topology-based Sampling** – Nodes and edges in G are explored using some variation of breadth-first search (i. e., sampling without replacement) or random walk (i. e., sampling with replacement) methods for inclusion in the sampled subgraph G_s .

In the past years, most of the existing work has focused on studying topology-based sampling methods [9, 10, 26, 30, 32]. This trend was driven by the need of collecting data from the web (i. e., web crawling), and the limitations of applying node and edge-based sampling methods to collect data from distributed web and online social networks (e. g., Facebook),

1.2.2 Spectrum of Computational Models

While most of the previous work focused on the question of *how to sample*, in this dissertation, we discuss a spectrum of computational models for the design and implementation of network sampling methods. Then, we analyze sampling methods that generalize across this spectrum, going from the simplest and least constrained model focused on sampling from static graphs to the more difficult and most constrained model of sampling from streaming graphs. This spectrum provides various opportunities for applying sampling methods in a variety of scenarios.

Static Network Sampling. Traditionally, network sampling has been studied in the case of simple static graphs [9, 10, 26, 30, 32]— such as forest fire, random walk and snowball sampling methods. Given a disk-resident graph G , these works make the simplifying assumption that the graph size is moderate and has a static structure. Specifically, it is assumed that the graph can fit entirely in the main memory, and the graph structure can be traversed arbitrarily (i. e., algorithms assume that the full neighborhood of each node can be accessed randomly in constant time), while many of the intrinsic complexities of realistic networks, such as the massive size, the time-evolving nature, and the temporal characteristics of these graphs, are totally ignored.

While studying static graphs is indeed important, the assumption that the graph fits in memory is not always realistic for real-world scenarios (e. g., online social networks). When the graph is too large to fit in memory, sampling requires random disk accesses that incur large I/O costs [33–35]. These random accesses on disks are typically much slower than random accesses in main memory [33]. Therefore, a key disadvantage of these methods is that they don’t differentiate between a graph that can fit entirely in the main memory and a graph that cannot. Naturally, this raises the question: how can we sample from these large networks *sequentially*, one edge at a time, while *minimizing* the number of *passes* over the edges? Given that the edges can be stored in some arbitrary order, most of the topology-based sampling methods, such as breadth-first search, random walk, or forest-fire sampling, and node-based sampling methods are not appropriate as they require random access to a node’s neighbors (which would require many passes over the edges).

Streaming Network Sampling. In addition to their massive size, many real-world networks are also likely to be continuously streaming over time. A *streaming graph* is a rapid, continuous, and possibly unbounded, time-varying stream of *edges* that is clearly too large to fit in memory except for short windows of time (e. g., a single day, hour, etc). Streaming graphs occur frequently in the real-world and can

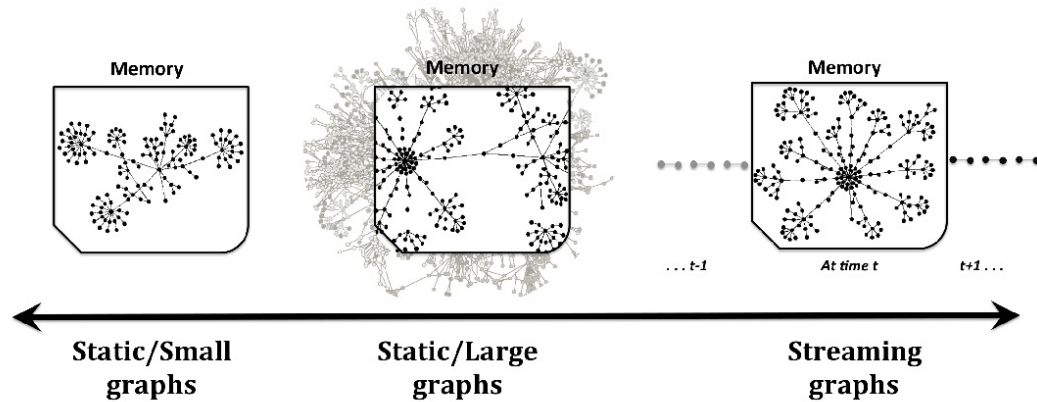


Fig. 1.1.: Spectrum of Computational Models for network sampling: from static to streaming.

be found in many modern online and communication applications such as: Twitter posts, Facebook likes/comments, email communications, network monitoring, sensor networks, among many other applications [36]. Although these applications are quite prevalent, there has been little focus on developing network sampling algorithms that address the complexities of streaming graphs. Generally, streaming graphs differ from static graphs in four main aspects:

1. The massive volume of edges is far too large to fit into main memory
2. The graph structure is not fully observable at any point in time (i. e., only sequential access is feasible, not random access)
3. Efficient real-time processing is critically important
4. The stream exhibits temporal characteristics (e. g., edge frequency) that don't appear in simple static graphs

Clearly, for such massive streaming graphs, sampling algorithms that process the edge stream in *single-pass* and maintains a small consulting state in memory, are more practical and efficient than those that process the data in an arbitrary order.

These specific sampling algorithms are typically classified as *graph stream sampling algorithms*.

The above discussion shows a natural progression of computational models for network sampling—from static to streaming. The majority of previous work has focused on sampling from static graphs, which is the simplest and least restrictive problem setup. In this work, however, we investigate the more challenging issues of sampling from massive disk-resident and streaming graphs. This leads us to propose a spectrum of computational models for network sampling as shown in Figure 1.1, where we outline the computational models for sampling from: (1) static graphs, (2) large graphs, and (3) streaming graphs. This spectrum not only provides insights into the complexity of the computational models (i. e., static vs. streaming), but also the complexity of the algorithms that are designed for each scenario. More complex algorithms are more suitable for the simplest computational model of sampling from static graphs. In contrast, as the complexity of the computational model increases to a streaming scenario, efficient algorithms become necessary. Thus, there is a trade-off between the complexity of the sampling algorithm and the complexity of the computational model (static \rightarrow streaming).

A subtle but important consequence is that any algorithm designed to work over streaming graphs is also applicable in the simpler computational models (i. e., static graphs). However, the converse is not true, algorithms designed for sampling a static graph that can fit in memory, may not be generally applicable for streaming graphs (as their implementation may require an intractable number of passes over the data). Clearly, node and topology-based sampling methods are not efficient for sampling sequentially from the edge stream, on the other hand, edge-based sampling methods are naturally amenable to streaming implementation.

1.3 Problem Statement

From the previous discussion, it is clear that network sampling must mediate between a variety of problem-specific constraints and opposing priorities [11,37]. These constraints are defined by the problem under study, such as the characteristics of the data (e. g., heavy-tailed data distribution), the data access constraints (e. g., streaming vs. distributed data), the available resources (e. g., memory, bandwidth), and the accuracy needs of queries. In this dissertation, we propose network sampling methods that can mediate between a variety of problem-specific constraints. One of the central questions in this dissertation is: given a graph G represented as an edge stream $\{e_1, e_2, \dots, e_t, \dots\}$, how to sample a subgraph $G_s = (V_s, E_s)$ sequentially, one edge at a time, from the population graph G , while maintaining a small state $|\Psi| \leq O(|G_s|)$?

A key challenge for any stream sampling algorithm is the need to decide whether to include an edge $e_t \in E$ in the sample or not as the edge is observed in the stream. The stream sampling algorithm may maintain a state $|\Psi|$ and consult the state to determine whether to sample an edge or not, but the total storage space (i. e., $|\Psi|$) is usually of the order of the size of the output: $|\Psi| = O(|G_s|)$. Note that this requirement is potentially larger than the $o(N)$ (and preferably $polylog(N)$) that streaming algorithms typically require [35]. But since any algorithm cannot require less space than its output, we relax this requirement.

In this work, we propose a framework that can be used to design network sampling algorithms in the streaming computational model. Generally, the process of network sampling in the streaming computational model can be decomposed into two key functions: a (1) Sampling function, and a (2) Holding function. For each observed edge e_t in the stream of edges $\{e_1, e_2, \dots, e_t, \dots\}$, the probability of selecting e_t can be formulated as a conditional probability as follows,

$$\mathcal{P}[e_t \text{ is selected} \mid \text{stored state } \Psi] = p_t$$

If e_t is independent of the stored state Ψ , for example, e_t is not adjacent to a previously selected edge, then $(\mathcal{P}[e_t \text{ is selected}] = P_S)$, which means the selection probability is a constant. We call this the *sampling function* and we use it for sampling and exploring new regions (unsampled) in the graph. On the other hand, if e_t is not independent of the stored state Ψ , for example, e_t is adjacent to one or more previously selected edges ($e_k \in \Psi$), then the selection probability is a function of the stored state ($\mathcal{P}[e_t \text{ is selected} \mid \text{stored state } \Psi] = f(\Psi)$, and $P_H = f(\Psi)$). We call this the *holding function* and we use it for holding on or exploring a previously sampled region in the graph. This holding function essentially involves a frequent update of the state for each sampled edge. One of the primary challenges in this case is making sure the state updates do not cause state increase. In this work, we show ways for controlling the size of the sampled state, such as reservoir sampling [38] to ensure the size of the reservoir remains constant.

A key property of the proposed framework is that any edge-based sampling algorithm is statistically biased towards sampling graph regions that are much denser than the rest of the graph regions. This is due to the bias of edge-based sampling algorithms to the selection of high degree nodes and network hubs (see Chapter 3 for statistical bias analysis). It has been shown by Karger in [39] that edge-based sampling algorithms have a higher likelihood of containing graph cuts with lower value [40].

Moreover, the proposed framework is quite generic and flexible. By varying the conditional dependence of the sampling probabilities on the stored state, one can tune the estimation of various properties of the original graph efficiently with arbitrary degrees of accuracy. For example, in uniform edge sampling, the sampling probability of edges is a constant uniform probability and totally independent of the stored state. Thus, in the case of uniform edge sampling, the sampling and holding functions are exactly the same, i. e., $(\mathcal{P}[e_t \text{ is selected}] = P_S = P_H = p)$. Furthermore, we can adapt the holding function to simply track or capture certain graph properties (e. g., triangles). Similarly, by carefully designing the sampling function, we can obtain

a uniform random sample of nodes that appeared in the stream so far (similar to the classical uniform node sampling). Therefore, the proposed framework does not only help to design new sampling methods, but also to extend existing work to the streaming computational model.

The central thesis statement of this work is formulated as:

Statistical network stream sampling can be approached as a principled approach with two main functions: (1) the sampling function, and (2) the holding function – By using this approach, we obtain a generic and flexible framework that allows tuning the sampling and estimation of various graph properties efficiently and accurately, while being applicable across the full spectrum of computational models.

Throughout this dissertation, we investigate sampling as well as fast and efficient methods to scale up network analysis and mining in static and streaming graphs. We formally study two primary network analysis tasks: (1) Sampling a representative subgraph, and (2) Unbiased estimation and efficient methods for subgraph counting.

1.3.1 Sampling from Large Static Graphs

Given a large static disk-resident graph $G = (V, E)$, the question of how to obtain a representative subgraph $G_s = (V_s, E_s)$ with $n = |V_s|$ nodes from G has been studied in previous work [9, 10, 26, 30, 32]. Most of the previous work assume that the graph G can fit entirely in main

In this dissertation, we propose a two-pass sampling method that runs sequentially with only *two passes* over the edge stream. The multi-pass computational model takes a middle position in the memory spectrum by relaxing the $\text{polylog}(N)$ storage requirement of the streaming model [35, 41]. In addition to relaxing the memory requirement, the multi-pass model allows multiple passes over the input stream [35, 41]. In some applications, a small number of sequential passes over the edge stream

would be more efficient than many random disk accesses to the graph [41]. The proposed method is used to investigate three main research challenges:

- Given a large static disk-resident graph G , how can we sample from G *sequentially*, one edge at a time, while *minimizing* the number of *passes* over the edges?
- Analysis of the bias coming from sampling sequentially from the edge stream, and the effect of the bias on the sampled subgraph.
- What is impact of different sampling methods on the performance of data mining tasks, such as relational classification?

1.3.2 Sampling from Streaming Graphs

Today’s networks are not only large in size but also continuously streaming over time, and therefore, their structure can be viewed as a dynamical system. The normal operation of any dynamical system can be described by a process that transitions between different states over time. In this component, we study how to use network sampling to analyze the temporal and structural properties of streaming graphs.

Previous work has focused primarily on streaming uniform edge sampling, such as the work in [40], which is useful for some analysis tasks but not all of them. Therefore, we propose methods that extend traditional sampling algorithms from the various classes (e. g., node, edge, etc) into the streaming computational model. Then, We propose a novel graph stream sampling method that efficiently sample from the graph stream in a single pass. We evaluate these methods on a variety of multigraph streams and show their performance on both temporal and structural properties. Our work in this part is focused on exploring two main research challenges:

- Given a streaming graph $e_t \in E$, how to sample a representative subgraph in a *single-pass* over the edge stream, while maintaining a small state in memory $|\Psi| \leq O(|S|)$?

- How to extend traditional sampling algorithms to streaming implementation in a single pass?
- What is the impact of single-pass stream sampling on structural and temporal graph properties?

1.3.3 Big Graph Analytics and Unbiased Estimation

In this part, we focus on exploring fast, efficient approximation and exact methods to obtain counts of frequent patterns/subgraphs in the edge stream of the graph. One goal of this part is to develop methods useful for answering various graph queries accurately, quickly, and efficiently. For this goal, we propose a generic stream sampling framework for big graph analytics, called Graph Sample and Hold (gSH). gSH is a parametric framework that can be used to estimate subgraph counts, such as the number of edges, triangles, etc.

While previous work focused particularly on sampling schemes used to estimate a certain graph property [17, 19, 42–44], gSH is generic and can be used to estimate various graph properties with the same sampling scheme. Our framework starts by sampling from massive graphs sequentially in a single pass, one edge at a time, while maintaining a small state in memory. We also develop statistical estimators based on the construction of Horvitz-Thompson [45], and we apply them to obtain unbiased estimates of subgraph counts.

Another goal of this part is to extract useful structural features, such as motif frequencies, that would benefit important machine learning tasks. For example, large-scale graph classification, prediction, anomaly detection, among others. For this goal, we propose a fast efficient algorithm for motif counting that take only a fraction of the time to compute when compared with the current methods used. The proposed motif counting algorithm leverages a number of combinatorial arguments that we show for the different motif patterns. For each edge, we count a few of the patterns, and with these counts along with combinatorial arguments, we derive the exact counts of the

others in constant time. The combinatorial arguments we show enable us to obtain significant improvement on the scalability of motif counting.

In summary, we investigate the following research questions:

- How to quickly and efficiently estimate various subgraph counts, such as the number of edges, triangles, etc?
- How to efficiently count all possible motifs (or graphlets) of size $k = \{2, 3, 4\}$ nodes?

1.4 Contributions and Outline

This dissertation is positioned to extend the range, applicability, and performance gains of network sampling as a tool that is not only useful for web crawling and data collection, but also for the analysis and mining of massive disk-resident and streaming graphs efficiently. We show how network sampling can be used as a mediator of various problem-specific constraints, such as the characteristics of the data (e. g., heavy-tailed data distribution), the data access constraints (e. g., streaming vs. distributed data), the available resources (e. g., memory, bandwidth), and the accuracy needs of queries.

We propose a flexible framework for designing statistical graph stream sampling. The potential benefits of the proposed framework are two-fold. First, it will lead to more interpretable sampling designs, that efficiently capture the specific graph properties of interest. This should benefit big graph analytics and data mining applications in general, since interpretability is a quality that is often important for domain experts to design useful sampling methods. Second, it will lead to samples with better quality that efficiently mirror the properties of the population graph.

In addition, we propose a fast efficient algorithm for motif counting that is significantly faster than the current methods used, while also scaling to much larger networks with millions of nodes and edges. The proposed motif counting algorithm leverages a number of combinatorial arguments, which enable us to obtain significant improvement on the scalability of motif counting. Thus, this brings new opportu-

nities to investigate the use of motifs on much larger networks and newer applications. Furthermore, a number of important machine learning applications are likely to benefit from such algorithm, including graph-based anomaly detection [7, 46], entity resolution [47], as well as features for improving community detection [48], role discovery [49], and relational classification [50, 51].

The rest of this dissertation is organized as follows: Chapter 2 describes the foundations of network analysis and sampling, highlighting the different objectives of network sampling, the population and units with respect to the specific goals, evaluation and classes of network sampling methods. In Chapter 3, we introduce our approach to sampling a subgraph from large static graphs. Next, in Chapter 4, we introduce our approach to sampling from streaming graphs. Then, we describe our framework for unbiased estimation of counts of frequent subgraphs in Chapter 5. In Chapter 6, we introduce our fast efficient algorithm for motif counting (counting all possible subgraphs of size 2, 3, 4 nodes). Finally, Chapter 7 concludes the dissertation and points out for future directions.

Parts of this dissertation have been published in peer-reviewed conferences and journals. In particular, the work in Chapter 2 and Chapter 3 is published in the ACM journal of TKDD [11] and WIN [52]. Parts of the work in Chapter 3 is published in ICWSM [53], and MLG [54]. Also, the work in Chapter 4 is published in the ACM journal of TKDD [11] and BigMine [55]. Moreover, the work in Chapter 5 is published in SIGKDD [23]. Finally, the work in Chapter 6 is published in [56], and other parts of this chapter are published in ICWSM [57].

2. BACKGROUND

In the context of statistical data analysis, a number of issues need to be considered carefully before collecting data and making inferences based on them. First, we need to identify the relevant *population* to be studied. Then, if sampling is necessary then we need to decide how to sample from that population. Generally, the term *population* is defined as the full set of representative units that one wishes to study (e. g., individuals in a particular city). In some instances, the population may be relatively small and therefore easy to study in its entirety (i. e., without sampling). For instance, it is fairly easy to study the *full* set of graduate students in a particular academic department. However, in many situations the population is large, unbounded, or difficult and/or costly to access in its entirety (e. g., the complete set of Facebook users). In this case, for efficiency reasons, a sample of units can be collected and characteristics of the population can then be estimated from the sampled units.

Network sampling is of interest to a variety of researchers in a range of distinct fields (e. g. statistics, social science, databases, data mining, machine learning) due to the numerous complex data sets that can be represented as graphs. While each area may investigate different types of networks, they have all considered *how* to sample. For example, in social science, snowball sampling is used extensively to run survey sampling in populations that are difficult-to-access (e. g., the set of drug users in a city) [58]. Similarly, in Internet topology measurements, breadth first search is used to *crawl* distributed, large-scale online social networks [30]. In structured data mining and machine learning, the focus has been on developing algorithms to sample small(er) subgraphs from a single large network [9]. These sampled subgraphs are further used to learn models (e. g., relational classification models [59]), evaluate and compare the performance of algorithms (e. g., different classification methods [60,61]),

and study complex network processes (e. g., information diffusion [62]). Section 2.6 provides a more detailed discussion of related work.

While this large body of research has developed methods to sample from networks, much of the work is problem-specific and there has been less work focused on developing a broader foundation for network sampling. More specifically, it is often not clear *when* and *why* particularly sampling methods are appropriate. This is because the goals and population are often not explicitly defined or stated up front, which makes it difficult to evaluate the quality of the recovered samples for other applications. One of the primary aims of this work is to define and discuss the foundations of network sampling more explicitly, such as: objectives/goals, population of interest, units, classes of sampling algorithms (i. e., node, edge, and topology-based), and techniques to evaluate a sample (e. g., network statistics and distance metrics). In this chapter, we outline a solid methodological framework for network sampling. The framework will facilitate the comparison of various network sampling algorithms, and help to understand their relative strengths and weaknesses with respect to particular sampling goals.

2.1 Foundations and Notations

Formally, we consider an input network represented as a graph $G = (V, E)$ with the node set $V = \{v_1, v_2, \dots, v_N\}$ and edge set $E = \{e_1, e_2, \dots, e_M\}$, such that $N = |V|$ is the number of nodes, and $M = |E|$ is the number of edges. We denote $\eta(\cdot)$ as any topological graph property. Therefore, $\eta(G)$ could be a *point statistic* (e. g., average degree of nodes in V) or a *distribution* (e. g., degree distribution of V in G).

Further, we define $\Lambda = \{a_1, a_2, \dots, a_k\}$ as the set of k attributes associated with the nodes describing their properties. Each node $v_i \in V$ is associated with an attribute vector $[a_1(v_i), a_2(v_i), \dots, a_k(v_i)]$ where $a_j(v_i)$ is the j^{th} attribute value of node v_i . For instance, in a Facebook network where nodes represent users and edges represent

friendships, the node attributes may include age, political view, and relationship status of the user.

Similarly, we denote $\beta = \{b_1, b_2, \dots, b_l\}$ as the set of l attributes associated with the edges describing their properties. Each edge $e_{ij} = (v_i, v_j) \in E$ is associated with an attribute vector $[b_1(e_{ij}), b_2(e_{ij}), \dots, b_l(e_{ij})]$. In the Facebook example, edge attributes may include relationship type (e. g., friends, married), relationship strength, and type of communication (e. g., wall post, photo tag).

Now, we define the network sampling process. Let σ be any sampling algorithm that selects a random sample S from G (i. e., $S = \sigma(G)$). The sampled set S could be a subset of the nodes ($S = V_s \subset V$), or edges ($S = E_s \subset E$), or a subgraph ($S = (V_s, E_s)$) where $V_s \subset V$ and $E_s \subset E$). The size of the sample S is defined relative to the graph size with respect to a sampling fraction ϕ ($0 \leq \phi \leq 1$). In most cases the sample size is defined as a fraction of the nodes in the input graph, e. g., $|S| = \phi \cdot |V|$. But in some cases, the sample size is defined relative to the number of edges ($|S| = \phi \cdot |E|$).

2.2 Goals, Units, and Population of Networks

While the explicit aim of many network sampling algorithms is to select a smaller subgraph from G , there are often other more implicit goals of the process that are left unstated. Here, we formally outline a range of possible goals for network sampling:

(Goal 1) ESTIMATE NETWORK PARAMETERS

Let $S \subset V$ or $S \subset E$. Then for a property η of G , if $\eta(S) \approx \eta(G)$, S is considered a *good* sample of G .

For example, let $S = V_s \subset V$ be the subset of sampled nodes, we can estimate the average degree of nodes G using S :

$$\widehat{deg}_{avg} = \frac{1}{|S|} \sum_{v_i \in S} deg(v_i \in G)$$

where $\text{deg}(v_i \in G)$ is the degree of node v_i as it appears in G , and a direct application of statistical estimators helps to correct sampling bias in $\widehat{\text{deg}}_{avg}$ [63].

(Goal 2) SAMPLE A REPRESENTATIVE SUBGRAPH

Let $S = G_s$ refer to a subgraph $G_s = (V_s, E_s)$ sampled from G . Then for a set of topological properties η_A of G , if $\eta_A(S) \approx \eta_A(G)$, S is considered a *good* sample of G .

Generally, subgraph representativeness is evaluated by selecting a set of graph topological properties that are important for a wide range of applications. This ensures that the sample subgraph S can be used in place of G for testing algorithms, systems, and/or models in an application. For example, [9] evaluated sample quality using topological properties like degree, clustering, and eigenvalues.

(Goal 3) ESTIMATE NODE ATTRIBUTES

Let $S \subset V$. Then for a function f_a of node attribute a , if $f_a(S) \approx f_a(V)$, S is considered a *good* sample of V (where V is the set of nodes in G).

For example, if a represents the age of users, we can estimate the average in G using S :

$$\widehat{a}_{avg} = \frac{1}{|S|} \sum_{v_i \in S} a(v_i)$$

Similar to goal 1, statistical estimators can be used to correct for bias.

(Goal 4) ESTIMATE EDGE ATTRIBUTES

Let $S \subset E$. Then for a function f_b of edge attribute b , if $f_b(S) \approx f_b(E)$, S is considered a *good* sample of E (where E is the set of edges in G).

For example, if b represents the relationship type of friends (e. g., married, coworkers), we can estimate the proportion of married relationships in G using S :

$$\hat{P}_{\text{married}} = \frac{1}{|S|} \sum_{e_{ij} \in S} 1_{(b(e_{ij})=\text{married})}$$

Clearly, the first two goals (1 and 2) focus on characteristics of entire networks, while the last two goals (3 and 4) focus on characteristics of nodes or edges in isolation. Therefore, these goals may be difficult to satisfy simultaneously—i.e., if the sampled data enable accurate study for one, it may not allow accurate study for others. For instance, a representative subgraph sample could produce a biased estimate of node attributes.

Once the goal is outlined, the population of interest can be defined relative to the goal. In many cases, the definition of the population may be obvious (e. g., nodes in the network). The main challenge is then to select a representative subset of units in the population in order to make the study cost efficient and feasible. Other times, the population may be less tangible and difficult to define. For example, if one wishes to study the characteristics of a system or *process*, there is not a clearly defined set of items to study. Instead, one is often interested in the overall behavior of the system. In this case, the population can be defined as the set of possible outcomes from the system (e.g., measurements over all settings) and these *units* should be sampled according to their underlying probability distribution.

In the first two goals outlined above, the objective of study is an entire network (either for structure or parameter estimation). In goal 1, if the objective is to estimate local properties from the nodes (e.g. degree distribution of G), then the elementary units are the nodes, and then the population would be the set of all nodes V in G . However, if the objective is to estimate global properties (e.g. diameter of G), then the elementary units correspond to subgraphs (any $G_s \subset G$) rather than nodes and the population should be defined as the set of subgraphs of a particular size that

could be drawn from G . In goal 2, the objective is to select a subgraph G_s , thus the elementary units correspond to subgraphs, rather than nodes or edges (goal 3 and 4). As such, the population should also be defined as the set of subgraphs of a particular size that could be drawn from G .

2.3 Classes of Sampling Methods

Once the population has been defined, a sampling algorithm σ must be chosen to sample from G . Sampling algorithms can be categorized as node, edge, and topology-based sampling, based on whether nodes or edges are locally selected from G (node and edge-based sampling) or if the selection of nodes and edges depends more on the existing topology of G (topology-based sampling).

Graph sampling algorithms have two basic steps:

- (1) *Node selection*: used to sample a subset of nodes $S = V_s$ from G , (i. e., $V_s \subset V$).
- (2) *Edge selection*: used to sample a subset of edges $S = E_s$ from G , (i. e., $E_s \subset E$)

When the objective is to sample only nodes or edges (e. g., goals 3, 4 or 1), then either step 1 or step 2 is used to form the sample S . When the objective is to sample a subgraph G_s from G (e. g., goals 2 or 1), then both step 1 and 2 from above are used to form S , (i. e., $S = (V_s, E_s)$). In this case, the edge selection is often conditioned on the selected node set in order to form an *induced subgraph* by sampling a subset of the edges incident to V_s (i. e. $E_s = \{e_{ij} = (v_i, v_j) | e_{ij} \in E \wedge v_i, v_j \in V_s\}$). This process is called graph induction. We distinguish between two approaches of graph induction—*total* and *partial* graph induction—which differ by whether *all* or *some* of the edges incident on V_s are selected. The resulting sampled graphs are referred to as the *induced subgraph* and *partially induced subgraph* respectively.

While the discussion of the algorithms in the next chapters focuses more on sampling a subgraph G_s from G , they can easily generalize to sampling only nodes or edges.

Node sampling (NS). In classic node sampling, nodes are chosen independently and uniformly at random from G for inclusion in the sampled graph G_s . For a target fraction ϕ of nodes required, each node is simply sampled with a probability of ϕ . Once the nodes are selected for V_s , the sampled subgraph is constructed to be the *induced subgraph* over the nodes V_s , i. e., all edges among the $V_s \in G$ are added to E_s . While node sampling is intuitive and relatively straightforward, the work in [64] shows that it does not accurately capture properties of graphs with power-law degree distributions. Similarly, [65] shows that although node sampling appears to capture nodes of different degrees well, due to its inclusion of all edges for a chosen node set only, the original level of connectivity is not likely to be preserved.

Edge sampling (ES). In classic edge sampling, edges are chosen independently and uniformly at random from G for inclusion in the sampled graph G_s . Since edge sampling focuses on the selection of edges rather than nodes to populate the sample, the node set is constructed by including both incident nodes in V_s when a particular edge is sampled (and added to E_s). The resulting subgraph is partially induced, which means no extra edges are added over and above those that were chosen during the random edge selection process. Unfortunately, ES fails to preserve many desired graph properties. Due to the independent sampling of edges, it does not preserve clustering and connectivity. It is however more likely to capture path lengths, due to its bias towards high degree nodes and the inclusion of both end points of selected edges.

Topology-based sampling. Due to the known limitations of NS [64, 65] and ES (bias toward high degree nodes), researchers have also considered many other topology-based sampling methods (also referred to as exploration sampling), which use breadth-first search (i. e., sampling without replacement) or random walks (i. e., sampling with replacement) over the graph to construct a sample.

One example is snowball sampling, which adds nodes and edges using breadth-first search from a randomly selected seed node, but stops early once it reaches a

particular size. Snowball sampling accurately maintains the network connectivity within the snowball, but it suffers from *boundary bias* in that many peripheral nodes (i. e., those sampled on the last round) will be missing a large number of neighbors [65].

Another example is the Forest Fire Sampling (FFS) method [9], which uses *partial* breadth-first search where only a fraction of neighbors are followed for each node. The algorithm starts by picking a node uniformly at random and adding it to the sample. It then “burns” a random proportion of its outgoing links, and adds those edges, along with the incident nodes, to the sample. The fraction is determined by sampling from a geometric distribution with mean $(p_f/(1 - p_f))$. The authors recommend setting $p_f = 0.7$, which results in an average burn of 2.33 edges per node. The process is repeated recursively for each burned neighbor until no new node is selected, then a new random node is chosen to continue the process until the desired sample size is obtained. There are other algorithms such as respondent-driven sampling [66] and expansion sampling [28] that we give more details on in Chapter 2.6.

In general, such topology-based sampling approaches form the sampled graph out of the explored nodes and edges, and usually perform better than simple algorithms such as NS and ES.

2.4 Evaluation of Sampling Methods

When the goal is to approximate the entire input network—either for estimating parameters (goal 1) or to select a representative subgraph structure (goal 2)—the accuracy of network sampling methods is often measured by comparing structural network statistics (e. g., degree). We first define a suite of common network statistics and then discuss how they can be used to quantitatively compare sampling methods.

Network Statistics. The commonly considered network statistics can be compared along two dimensions: *local* vs. *global* statistics, and *point* statistic vs. *distribution*. A local statistic is used to describe a characteristic of a local graph element (e. g., node, edge, subgraph). For example, node degree and node clustering coefficient.

Table 2.1.: Description of Network Statistics

Network Statistic.	Description
DEGREE DIST.	Distribution of degrees for all nodes in the network
PATH LENGTH DIST.	Distribution of (finite) shortest path lengths between all pairs of nodes in the network
CLUSTERING COEFFICIENT DIST.	Distribution of local clustering for all nodes in the network
K-CORE DIST.	Distribution of k -core decomposition of the network
EIGENVALUES	Distribution of the eigenvalues of the network adjacency matrix vs. their rank
NETWORK VALUES	Distribution of eigenvector components vs. their rank, for the largest eigenvalue of the network adjacency matrix

On the other hand, a global statistic is used to describe a characteristic of the entire graph. For example, global clustering coefficient and graph diameter. Similarly, there is also the distinction between point statistics and distributions. A point-statistic is a single value statistic (e. g., diameter) while a distribution is a multi-valued statistic (e. g., distribution of path length for all pairs of nodes). Clearly, a range of network statistics are important to investigate the full graph structure.

In this work, we focus on the goal of sampling a representative subgraph G_s from G , by using distributions of network characteristics calculated on the level of nodes, edges, sets of nodes or edges, and subgraphs. Table 2.1 provides a summary for the six network statistics we use and we formally define the statistics below:

- (1) *Degree distribution*: The fraction of nodes with degree k , for all $k > 0$

$$p_k = \frac{|\{v \in V | \text{deg}(v) = k\}|}{N}$$

Degree distribution has been widely studied by many researchers to understand the connectivity in graphs. Many real-world networks were shown to have a

power-law degree distribution, for example in the Web [67], citation graphs [68], and online social networks [69].

- (2) *Path length distribution*: Also known as the *hop plot* distribution and denotes the fraction of pairs $(u, v) \in V$ with a shortest-path distance ($dist(u, v)$) of h , for all $h > 0$ and $h \neq \infty$

$$p_h = \frac{|\{(u,v) \in V | dist(u,v)=h\}|}{N^2}$$

The path length distribution is essential to know how the number of paths between nodes expands as a function of distance (i. e., number of hops).

- (3) *Clustering coefficient distribution*: The fraction of nodes with clustering coefficient ($cc(v)$) c , for all $0 \leq c \leq 1$

$$p_c = \frac{|\{v \in V' | cc(v)=c\}|}{|V'|}, \text{ where } V' = \{v \in V | deg(v) > 1\}$$

Here the clustering coefficient of a node v is calculated as the number of triangles centered on v divided by the number of pairs of neighbors of v (e. g., the proportion of v 's neighbor that are linked). In social networks and many other real networks, nodes tend to cluster. Thus, the clustering coefficient is an important measure to capture the transitivity of the graph [70].

- (4) *K-core distribution*: The fraction of nodes in graph G participating in a k -core of order k . The k -core of G is the largest induced subgraph with minimum degree k . Formally, let $U \subseteq V$, and $G_{[U]} = (U, E')$ where $E' = \{e_{u,v} \in E | u, v \in U\}$. Then $G_{[U]}$ is a k -core of order k if $\forall v \in U \ deg_{G_{[U]}}(v) \geq k$.

Studying k -cores is an essential part of social network analysis as they demonstrate the connectivity and community structure of the graph [71–73]. We denote the *maximum core number* as the maximum value of k in the k -core distribution. The *maximum core number* can be used as a lower bound on the degree of the nodes that participate in the largest induced subgraph of G . Also, the core sizes can be used to demonstrate the localized density of subgraphs in G [74].

- (5) *Eigenvalues*: Let A be the corresponding adjacency matrix of a graph G , then the decomposition of A into eigenvalues ($\Lambda = [\lambda_1, \dots, \lambda_n]$) and eigenvectors ($\mathbf{X} = [X_1, \dots, X_n]$) satisfies the equation $(A - \Lambda I)X = 0$, where I is the $N \times N$ identity matrix. We number the set of eigenvalues (and associated eigenvectors) in descending order $\lambda_1 \geq \dots \geq \lambda_N$, then we compare the largest 25 eigenvalues of the sampled graphs to their counterparts in G . Note that eigenvalues are the basis of spectral graph analysis [75].
- (6) *Network values*: The distribution of the principal eigenvector components (i.e., the components $x_i \in X_1$ associated with the largest eigenvalue λ_1 of the graph adjacency matrix). We compare the largest 100 components of the principal eigenvector of the sampled graphs to their counterparts in G .

Next, we describe the use of these statistics for comparing sampling methods.

Distance Measures for Quantitatively Comparing Sampling Methods. A *good* sample has properties that approximate the full graph G (i.e., $\eta(S) \approx \eta(G)$). Thus, the distance between the property in G and the property in G_s is often used to evaluate sample *representativeness* quantitatively (i.e., $dist[\eta(G), \eta(G_s)]$) and sampling algorithms that minimize the distance are considered superior. When the goal is to provide estimates of global network parameters (e.g., average degree), then $\eta(\cdot)$ may return *point statistics*. However, when the goal is to provide a representative subgraph sample, then $\eta(\cdot)$ may return *distributions* of network properties (e.g., degree distribution). These distributions reflect how the graph structure is distributed across nodes and edges. The *dist* function used for evaluation could be typically any distance measure (e.g., absolute difference). In this chapter, since we focus on using distributions to characterize graph structure, we use four different distributional distance measures for evaluation.

- (1) *Kolmogorov-Smirnov (KS) statistic*: Used to assess the distance between two cumulative distribution functions (CDF). The KS-statistic is a widely used measure of the agreement between two distributions, including in [9] where it is used

to illustrate the accuracy of FFS. It is computed as the maximum vertical distance between the two distributions, where x represents the range of the random variable and F_1 and F_2 represent two CDFs:

$$KS(F_1, F_2) = \max_x |F_1(x) - F_2(x)|$$

- (2) *Skew divergence (SD)*: Used to assess the difference between two probability density functions (PDF) [76]. Skew divergence is used to measure the Kullback-Leibler (KL) divergence between two PDFs P_1 and P_2 that do not have continuous support over the full range of values (e. g., discrete degrees). KL measures the average number of extra bits required to represent samples from the original distribution when using the sampled distribution. However, since KL divergence is not defined for distributions with different areas of support, skew divergence *smooths* the two PDFs before computing the KL divergence:

$$SD(P_1, P_2, \alpha) = KL[\alpha P_1 + (1 - \alpha)P_2 || \alpha P_2 + (1 - \alpha)P_1]$$

The results shown in [76] indicate that using SD yields better results than other methods to approximate KL divergence on non-smoothed distributions. In this work, as in [76], we use $\alpha = 0.99$.

- (3) *Normalized L_1 distance*: In some cases, for evaluation we will need to measure the distance between two positive m -dimensional real vectors p and q such that p is the true vector and q is the estimated vector. For example, to compute the distance between two vectors of eigenvalues. In this case, we use the normalized L_1 distance:

$$L_1(p, q) = \frac{1}{m} \sum_{i=1}^m \frac{|p_i - q_i|}{p_i}$$

- (4) *Normalized L_2 distance*: In other cases, when the vector components are fractions (less than one), we use the normalized euclidean distance L_2 distance (e. g., to compute the distance between two vectors of network values):

$$L_2(p, q) = \frac{\|p - q\|}{\|p\|}$$

2.5 Models of Computation

In this section, we discuss the different models of computation that can be used to implement network sampling methods. At first, let us assume the network $G = (V, E)$ is given (e. g., stored on a large storage device). Then, the goal is to select a sample S from G .

Traditionally, network sampling has been explored in the context of a *static model of computation*. This simple model makes the fundamental assumption that it is easy and fast (i. e., constant time) to randomly access any location of the graph G . For example, random access may be used to query the entire set of nodes V or to query the neighbors $\mathcal{N}(v_i)$ of a particular node v_i (where $\mathcal{N}(v_i) = \{v_j \in V | e_{ij} = (v_i, v_j) \in E\}$). However, random accesses on disks are much slower than random accesses in main memory. A key disadvantage of the static model of computation is that it does not differentiate between a graph that can fit entirely in the main memory and a graph that cannot. Conversely, the primary advantage of the static model is that it is a natural extension of how we *understand* and *view* the graph—thus, it is a simple framework within which to design algorithms.

Although designing sampling algorithms with a static model of computation in mind is indeed appropriate for some applications, this approach assumes the input graphs are relatively small, can fit entirely into main memory, and have static structure (i. e., not changing over the time). This is unrealistic for many domains. For instance, many social, communication, and information networks naturally change over time and are massive in size (e. g., Facebook, Twitter, Flickr). The sheer size and dynamic nature of these networks make it difficult to load the full graph entirely in the main memory. Therefore, the static model of computation cannot realistically capture all the intricacies of many real world graphs that we study today.

In addition, many real-world networks that are currently of interest are *too large* to fit into memory. In this case, sampling methods that require random disk access can incur large I/O costs for loading and reading the data. Naturally, this raises

a question as to how we can sample from large networks more efficiently (i.e., in a sequential fashion rather than assuming random access). In this context, most of the topology based sampling procedures such as breadth-first search and random-walk sampling are no longer appropriate as they require the ability to randomly access a node’s neighbors $\mathcal{N}(v_i)$. If access is restricted to sequential passes over the edges, a large number of passes over the edges would be needed to repeatedly query $\mathcal{N}(\cdot)$. In a similar way, node sampling would no longer be appropriate as it not only requires random access for querying a node’s neighbors but it also requires random access to the entire node set V in order to obtain a uniform random sample.

A *streaming model of computation* in which the graph can only be accessed sequentially as a stream of edges, is therefore more preferable for these situations [77]. The streaming model completely discards the possibility of random access to G and the graph can only be accessed through an ordered scan of the edge stream. A sampling algorithm in this context may use the main memory for holding a portion of the edges temporarily and perform random accesses on that subset. In addition, the sampling algorithm may access the edges repeatedly by making multiple passes over the graph stream. Formally, for any input network G , we assume G is represented as a graph stream (e.g., as in Figure 2.1).

Definition 2.5.1 (Graph Stream) *A graph stream is an ordered sequence of edges $e_{\pi(1)}, e_{\pi(2)}, \dots, e_{\pi(M)}$, where π is any arbitrary permutation on the edge indices $[M] = \{1, 2, \dots, M\}$, $\pi : [M] \rightarrow [M]$.*

Definition 2.5.1 is usually called the “adjacency stream” model in which the graph is presented as a stream of edges in an arbitrary order. In contrast, the “incidence stream” model assumes all edges incident to a vertex are presented in order successively [43]. In this work, we use the adjacency stream model because it is more reflective of the temporal ordering we observe in real world data sets.

While most real-world networks are too large to fit into main memory, many are also likely to occur naturally as *streaming* data. A streaming graph is a rapid, con-

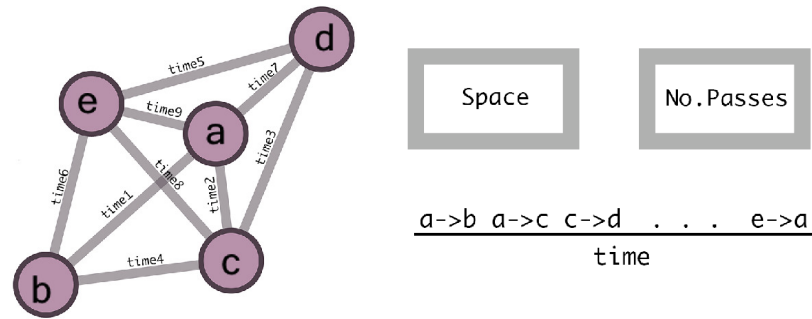


Fig. 2.1.: Illustration of *graph streams*—a sequence of edges ordered by time, and the complexity constraints of streaming algorithms (space and no. passes).

tinuous, and possibly unbounded, time-varying stream of edges that is both too large and too dynamic to fit into memory. These types of streaming graphs occur frequently in real-world communication and information domains. For example real-time tweets between users in Twitter, email logs, IP traffic, sensor networks, web search traffic, and many other applications. While sampling from these streaming networks is clearly challenging, their characteristics preclude the use of static models of computation and thus a more in depth investigation of streaming models of computation is warranted. This naturally raises a follow up question: how can we sample from large graph streams in a *single pass* over the edges? Generally streaming graphs differ from static graphs in three main aspects:

- (1) The massive volume of edges streaming over the time is far too large to fit in main memory.
- (2) The graph can only be accessed sequentially in a single pass (i. e., random access to neighboring nodes or to the entire graph is not possible).
- (3) Efficient, real-time processing is of critical importance.

In a streaming model, as each edge $e \in E$ arrives, the sampling algorithm σ needs to decide whether to include the edge or not as the edge *streams* by. The sampling

algorithm σ may also maintain state Ψ and consult the state to determine whether to sample e or not.

The complexity of a streaming sampling algorithm is measured by:

- (1) Number of passes over the stream ω .
- (2) Space required to store the state Ψ and the output.
- (3) Representativeness of the output sample S .

Multiple passes over the stream (i. e., $\omega > 1$) may be allowed for massive disk-resident graphs but prohibitive due to the massive volumes of stored graph data. However, multiple passes are generally not realistic for data sets where the graph is continuously streaming over time. In this case, a requirement of a single pass is more suitable (i. e., $\omega = 1$). The total storage space (i. e., Ψ) is usually of the order of the size of the output: $|\Psi| = O(|G_s|)$. Note that this requirement is potentially larger than the $o(N, t)$ (and preferably *polylog*(N, t)) that streaming algorithms typically require [35]. But, since any algorithm cannot require less space than its output, we relax this requirement in our definition as follows.

Definition 2.5.2 (Streaming Graph Sampling) *A streaming graph sampling algorithm is any sampling algorithm σ that produces a sampled graph G_s by sampling edges of the input graph G in a sequential order, preferably in one pass (i. e., $\omega = 1$), while maintaining state Ψ such that $|\Psi| \leq O(|G_s|)$.*

Clearly, it is more difficult to design sampling algorithms for the *graph stream* model, but it is critical to address the fundamental intricacies of, and implementation requirements for, real-world graphs that we see today.

We now have what can be viewed as a complete spectrum of computational models for network sampling, which ranges from the simple, yet less realistic, static graph model to the more complex, but more realistic, streaming model (see Figure 1.1). In Chapters 3-4, we will evaluate algorithms for representative subgraph sampling in each computation model from the spectrum.

We note that our assumption in this work is that the population graph G is visible in its entirety (*collected and stored on disk*). In many domains this assumption is valid, but in some cases the full structure of the population graph may be unknown prior to the sampling process (e. g., the deep Web or distributed information in peer-to-peer networks). Web/network crawling is used extensively to sample from graphs that are not fully visible to the public but naturally allow methods to explore the neighbors of a given node (e. g., hyperlinks in a web page). Topology-based sampling methods (e. g., breadth-first search, random walk) have been widely used in this context. However, these methods typically assume the graph G is *well connected* and remains *static* during crawling, as discussed in [78].

2.6 Review of Related work

Generally speaking, there are two bodies of work related to this work: (i) network sampling methods, which investigate and evaluate sampling methods with different goals for collecting a sample and (ii) graph stream methods, including work on mining and querying streaming data. In this section, we describe related work and put it in the perspective of the framework we discussed earlier in this chapter.

The problem of sampling graphs has been of interest in many different fields of research. Most of this body of research has focused on *how* to sample and each project evaluates the “goodness” of the resulting samples relative to its research goal(s).

2.6.1 Network Sampling in Social Science

In social science, the classic work done by [79] (also see the review papers [25,80]) provides basic solutions to the initial problems that arise when only a sample of the actors in a social network is available. [81] introduced the first snowball sampling method and originated the concept of “chain-referral” sampling. Further, Granovetter introduced the network community to the problem of making inferences about the entire population from a sample (e. g., estimation of network density) [82]. Later,

respondent-driven sampling was proposed in [66] and analyzed in [83] to reduce the biases associated with chain referral sampling of hidden populations. For an excellent survey about estimation of network properties from samples, we refer the reader to [84]. Generally, work in this area focuses on either the estimation of global network parameters (e. g., density) or the estimation of actors (node) attributes, i. e., goals 1 and 3.

2.6.2 Statistical Properties of Network Sampling

Another important trend of research focused on analyzing the statistical properties of sampled subgraphs. For example, the work in [65] and [85] studied the statistical properties of sampled subgraphs produced by classical node, edge and random walk sampling methods and discussed the bias in estimates of topological properties. Similarly, the work in [64] showed that the sampled subgraph of a scale free network is far from being scale free. Conversely, the work in [86] shows that under traceroute sampling, the resulting degree distribution follows a power law even when the original distribution is Poisson. Clearly, work in this area has focused on representative subgraph sampling (i. e., goal 2), considering how the topological properties of samples differ from those of the original network.

2.6.3 Network Sampling in Networked Systems

A large body of research in networked systems has focused on Internet *measurement*, which targets the problem of topology measurements in large-scale networks, such as peer-to-peer networks (P2P), the world wide web (WWW), and online social networks (OSN). The sheer size and distributed structure of these networks make it hard to measure the properties of the entire network. Network sampling, via *crawling*, has been used extensively in this context. In OSNs, sampling methods that do not revisit nodes are widely used (e. g., breadth-first search [30, 87, 88]). Breadth-first search has been shown to be biased towards high degree nodes [89], but the

work in [90] suggested analytical solutions to correct the bias. Random walk sampling has also been used to sample a uniform sample from users in Facebook and Last.fm (see e. g., [26]) . For a recent survey covering assumptions and comparing different methods of crawling, we refer the reader to [78]. Similar to OSNs, random walk sampling and its variants were used extensively to sample the WWW [91, 92], and P2P networks [27]. Since the classical random walk is biased towards high degree nodes, some improvements were applied to correct the bias. For example, the work in [93] applied Metropolis-Hastings random walks (MHRW) to sample peers in Gnutella network, and the work in [94] applied re-weighted random walk (RWRW) to sample P2P networks. Other work used m -dependent random walks and random walks with jumps [32, 95].

Overall, the work done in this area has focused extensively on sampling a uniform subset of nodes from the graph, to estimate topological properties of the entire network from the set of sampled nodes (i. e., goal 1).

2.6.4 Network Sampling in Structured Data Mining

Network sampling is a core part of data mining research. Representative subgraph sampling was first defined in [9]. [96] then proposed a generic Metropolis algorithm to optimize the representativeness of a sampled subgraph—by minimizing the distance of several graph properties from the sample to the original graph. Unfortunately, the number of steps until convergence is not known in advance and is usually quite large in practice. In addition, each step requires the computation of a complex distance function, which may be costly. In contrast to sampling, [97] explored reductive methods to shrink the existing topology of the graph. At the same time, other work discussed the difficulty of constructing a “universally representative” subgraph that preserves *all* properties of the original network. For example, our past work discussed the correlations between network properties and showed that accurately preserving some properties leads to under- or over-estimation of other properties (e. g., preserving the

average degree of the original network leads to a sampled subgraph with overestimated density) [52]. Also, [10] investigated the connection between the biases of topology-based sampling methods (e. g., breadth-first search) and some topological properties of the network (e. g., degree). These findings led to work focused on obtaining samples for specific applications, where the goal is to reproduce specific properties of the target network—for example, to preserve the community structure [28], to preserve the pagerank between all pairs of sampled nodes [98], or to visualize the graph [99].

Other network sampling goals have been considered as well. For example, sampling nodes to perform A/B testing of social features [29], sampling connected subgraphs [100], sampling nodes to analyze the fraction of users with a certain property [101], (i. e., goal 3), and sampling tweets (edges) to analyze the language used in Twitter (i. e., goal 4). In addition, [102] and [103] sample the output space of graph mining algorithms (e. g., graphlets), [104] collected information from social peers to enhance the information needs of users, and [105] studied the impact of sampling on the discovery of information diffusion.

Much of this work has focused on sampling in the context of a static model of computation—where algorithms assume that the graph can be loaded entirely into main memory, or the graph is distributed in a manner which allows exploration of a node’s neighborhood in a crawling fashion.

2.6.5 Graph Streams

Data stream querying and mining has garnered a lot of interest over the past few years [35, 106–108]. For example, sampling sequences (e. g., reservoir sampling) [38, 109, 110], computing frequency counts [111, 112] and load shedding [113], mining concept drifting data streams [114–117], clustering evolving data streams [36, 118], active mining and learning in data streams [119, 120], and other related mining tasks [121–124].

Recently, as a result of the proliferation of graph data (e.g., social networks, emails, IP traffic, Twitter hashtags), there has been an increased interest in mining and querying *graph streams*. Following the earliest work on graph streams [125], various problems were explored in the field of mining graph streams. For example, counting triangles [43, 126], finding common neighborhoods [127], estimating pagerank values [128], and characterizing degree sequences in multi-graph streams [129]. More recently, there is work on clustering graph streams [130], outlier detection [40], searching for subgraph patterns [131], and mining dense structural patterns [132].

Graph stream sampling was utilized in some of the work mentioned above. For example, [128] performed short random walks from uniformly sampled nodes to estimate pagerank scores. Also, [43] used sampling to estimate number of triangles in the graph stream. Moreover, [129] used a min-wise hash function to sample nearly uniformly from the set of all edges that have been at any time in the stream. The sampled edges were later used to maintain cascaded summaries of the graph stream. More recently, [40] designed a structural reservoir sampling approach (based on min-wise hash sampling of edges) for structural summarization. For an excellent survey on mining graph streams, we refer the reader to [133] and [77].

The majority of this work has focused on sampling a subset of nodes uniformly from the stream to estimate parameters such as the number of triangles or pagerank scores of the graph stream (i.e., goal 1). Also, as discussed above, other work has focused on sampling a subset of edges uniformly from the graph stream to maintain summaries (i.e., goal 2). These summaries can be further pruned (by lowering a threshold on the hash value [40]) to satisfy a specific stopping constraint (e.g., specific number of nodes in the summary). In this work, since we focus primarily on sampling a representative subgraph $G_s \subset G$ from the graph stream, we compare to some of these methods in Chapter 4.

3. SAMPLING FROM LARGE STATIC GRAPHS

In this chapter, we focus on how to sample a representative subgraph $G_s = (V_s, E_s)$ from $G = (V, E)$ (i. e., goal 2 from Section 2.2). A representative sample G_s is essential for many applications in machine learning, data mining, and network simulations. As an example, it can be used to drive realistic simulations and experimentation before deploying new protocols and systems in the field [97]. We evaluate the representativeness of G_s relative to G , by comparing distributions of six topological properties calculated over nodes, edges, and subgraphs (as summarized in Table 2.1).

3.1 Motivation

We distinguish between the degree of the sampled nodes before and after sampling. For any node $v_i \in V_s$, we denote k_i to be the node degree of v_i in the input graph G . Similarly, we denote k_i^s to be the node degree of v_i in the sampled subgraph G_s . Note that $k_i = |\mathcal{N}(v_i)|$, where $\mathcal{N}(v_i) = \{v_j \in V \mid e_{ij} = (v_i, v_j) \in E\}$ is the set of neighbors of node v_i . Clearly, when a node is sampled, it is not necessarily the case that all its neighbors are sampled as well, and therefore $0 \leq k_i^s \leq k_i$.

We propose a simple and efficient two-pass sampling algorithm: *2-pass induced edge sampling* (for brevity ES-i). ES-i has several advantages over current sampling methods as we show later in this chapter:

- (1) ES-i preserves the topological properties of G better than many of current sampling algorithms.
- (2) ES-i can be easily implemented as a *two-pass* sampling algorithm using only two passes over the edges of G (i. e., $\omega = 2$).
- (3) ES-i is suitable for sampling large graphs that cannot fit into main memory.

3.2 Two-Pass Stream Sampling

We formally specify ES-i in Algorithm 3.1. Initially, ES-i selects the nodes in pairs by sampling edges uniformly (i. e., $p(e_{ij} \text{ is selected}) = 1/|E|$) and adds them to the sample (V_s). Then, ES-i augments the sample with all edges that exist between any of the sampled nodes ($E_s = \{e_{ij} = (v_i, v_j) \in E \mid v_i, v_j \in V_s\}$). These two steps together form the sample subgraph $G_s = (V_s, E_s)$. For example, suppose edges $e_{12} = (v_1, v_2)$ and $e_{34} = (v_3, v_4)$ are sampled in the first step, which leads to the addition of the vertices v_1, \dots, v_4 into the sampled graph. In the second step, ES-i adds all the edges that exist between the sampled nodes—for example, edges $e_{12} = (v_1, v_2)$, $e_{34} = (v_3, v_4)$, $e_{13} = (v_1, v_3)$, $e_{24} = (v_2, v_4)$, and any other possible combinations involving v_1, \dots, v_4 that appear in G .

Algorithm 3.1: ES-I(ϕ, E)

Input : Sample fraction ϕ , Edge set E
Output: Sampled Subgraph $G_s = (V_s, E_s)$

```

1  $V_s = \emptyset, E_s = \emptyset$ 
2 // Node selection step
3 while  $|V_s| < \phi \times |V|$  do
4    $r = \text{random}(1, |E|)$ 
5   // uniformly random
6    $(u, v) = e_r$ 
7    $V_s = V_s \cup \{u, v\}$ 
8 // Edge selection step
9 for  $k = 1 : |E|$  do
10   $e_k = (u, v)$ 
11  if  $u \in V_s$  AND  $v \in V_s$  then
12     $E_s = E_s \cup \{e_k\}$ 

```

Algorithm 3.1 can also be generalized to control the total number of edges in the sampled subgraph. For example, instead of adding an edge e_k that exists between two sampled nodes, we select e_k with a probability $p(e_k) = \left(\frac{1}{\text{deg}_{G_s}(e_k)}\right)^\beta$, such that $\text{deg}_{G_s}(e_k)$ represents the number of adjacent edges to e_k in the sampled subgraph, and β is a sparsification parameter ($0 \leq \beta \leq 1$). To simplify the analysis, we use $\beta = 0$ in the rest of this chapter, which corresponds to taking the induced subgraph on the nodes sampled in the first step.

Since any sampling algorithm (by definition) selects only a subset of the nodes/edges in the graph G , it naturally produces subgraphs with underestimated degrees in the degree distribution of G_s . We refer to this as a *downward bias* and note that it is a property of all network sampling methods, since only a fraction of a node’s neighbors may be selected for inclusion in the sample (i. e., $k_i^s \leq k_i$ for any sampled node v_i).

Our proposed sampling methods exploits two key observations. First, by selecting nodes via edge sampling, the method is inherently biased towards the selection of nodes with high degrees, resulting in an *upward bias* in the (original) degree distribution if it is only measured from the sampled nodes (i. e., using the degree of the sampled nodes as observed in G). The upward bias resulting from edge sampling can help offset the downward bias of the sampled degree distribution of G_s . Furthermore, in addition to improving estimates of the sampled degree distribution, selecting high degree nodes also helps to produce a more connected sampled subgraph that preserves the topological properties of the graph G . This is due to the fact that high degree nodes often represent *hubs* in the graph, which serve as good navigators through the graph (e. g., many shortest paths usually pass through these hub nodes).

However, while the upward bias of edge sampling can help offset some issues of sample selection bias, it is not sufficient to use it in isolation to construct a good sampled subgraph. Specifically, since the edges are each sampled independently, edge sampling is unlikely to preserve much structure *surrounding* each of the selected nodes. This leads us to our second observation, that a simple *graph induction* step over the sampled nodes (where we sample all the edges between any sampled nodes from G) is crucial to recover much of the connectivity in the sampled subgraph—offsetting the downward degree bias as well as increasing local clustering in the sampled graph. More specifically, graph induction increases the likelihood that triangles will be sampled among the set of selected nodes, producing higher clustering coefficients and shorter path lengths in G_s .

These observations, while simple, make the sampled subgraph G_s approximate the characteristics of the original graph G more accurately, even better than topology-based sampling methods.

Since many real networks are now too large to fit into main memory, this raises the question of how to sample from G sequentially, one edge at a time, while minimizing the number of passes over the edges? Section 2.5 discussed how most of the topology-based sampling methods are no longer applicable in this scenario, since they require many passes over the edges. In contrast, ES-i can be implemented to run sequentially and requires only two passes over the edges of G (i. e., $\omega = 2$). Note that if the graph can fit in main memory, ES-i can be implemented with run time linear in the number of edges ($O(2E)$) by flipping coins with the corresponding probability. Next, we analyze the characteristics of ES-i and after that, in the evaluation, we show how it accurately preserves many of the properties of the graph G .

3.3 Analysis of Sampling Bias

In this section, we analyze the bias of ES-i's node selection analytically by comparing to the unbiased case of uniform sampling in which all nodes are sampled with a uniform probability (i. e., $p = \frac{1}{N}$). First, we denote f_D to be the degree sequence of G where $f_D(k)$ is the number of nodes with degree k in graph G . Let $n = |V_s|$ be the target number of nodes in the sample subgraph G_s (i. e., $\phi = \frac{n}{N}$).

3.3.1 Selection Bias Toward High Degree Nodes

We start by analyzing the upward bias to select high degree nodes by calculating the expected value of the number of nodes with original degree k that are added to the sample set of nodes V_s . Let $E_{UN}[f_D(k)]$ be the expected value of $f_D(k)$ for the sampled set V_s when n nodes are sampled uniformly with probability $p = \frac{1}{N}$:

$$\begin{aligned}
E_{UN}[f_D(k)] &= f_D(k) \cdot n \cdot p \\
&= f_D(k) \cdot \frac{n}{N}
\end{aligned}$$

Since, ES-i selects nodes proportional to their degree, the probability of sampling a node v_i with degree $k_i = k$ is $p' = \frac{k}{\sum_{j=1}^N k_j}$. Note that we can also express the probability as $p' = \frac{k}{2 \cdot |E|}$. Then we let $E_{ES_i}[f_D(k)]$ denote the expected value of $f_D(k)$ for the sampled set V_s when nodes are sampled with ES-i:

$$\begin{aligned}
E_{ES_i}[f_D(k)] &= f_D(k) \cdot n \cdot p' \\
&= f_D(k) \cdot n \cdot \frac{k}{2 \cdot |E|}
\end{aligned}$$

This leads us to Lemma 3.3.1, which shows ES-i's bias towards high degree nodes.

Lemma 3.3.1 *ES-i is biased to select high degree nodes. Let $k_{avg} = \frac{2 \cdot |E|}{N}$ be the average degree in G . Then for $k \geq k_{avg}$, $E_{ES_i}[f_D(k)] \geq E_{UN}[f_D(k)]$.*

Proof Consider the threshold k at which the expected value of $f_D(k)$ using ES-i sampling is greater than the expected value of $f_D(k)$ using uniform random sampling:

$$\begin{aligned}
E_{ES_i}[f_D(k)] - E_{UN}[f_D(k)] &= f_D(k) \cdot n \cdot \frac{k}{2 \cdot |E|} - f_D(k) \cdot \frac{n}{N} \\
&= f_D(k) \cdot n \left[\frac{k}{2 \cdot |E|} - \frac{1}{N} \right] \\
&= f_D(k) \cdot \frac{n}{2 \cdot |E|} \left[k - k_{avg} \right] \\
&\geq 0 \quad \text{when } k \geq k_{avg}
\end{aligned}$$

■

3.3.2 Downward Bias Due to Sampling

Next, instead of focusing on the *original* degree k as observed in the graph G , we focus on the *sampled* degree k^s as observed in the sample subgraph G_s , where $0 \leq k^s \leq k$. Let k_i^s be a random variable that represent the sampled degree of node v_i in G_s , given that the original degree of node v_i in G was k_i . We compare the expected value of k_i^s when using uniform sampling to the expected value of k_i^s when using ES-i. Generally, the degree of the node v_i in G_s depends on how many of its neighbors in G are sampled. When using uniform sampling, the probability of sampling one of the node's neighbors is $p = \frac{1}{N}$:

$$E_{UN}[k_i^s] = \sum_{j=1}^{k_i} p \cdot n = k_i \cdot \frac{n}{N}$$

Now, let us consider the variable $k_{\mathcal{N}} = \sum_{k'} k' \cdot P(k'|k)$, where $k_{\mathcal{N}}$ represents the average degree of the neighbors of a node with degree k as observed in G . The function $k_{\mathcal{N}}$ has been widely used as a global measure of the *assortativity* in a network [134]. If $k_{\mathcal{N}}$ is increasing with k , then the network is assortative—indicating that nodes with high degree connect to, on average, other nodes with high degree. Alternatively, if $k_{\mathcal{N}}$ is decreasing with k , then the network is disassortative—indicating that nodes of high degree tend to connect to nodes of low degree.

When using ES-i, the probability of sampling any of the node's neighbors is proportional to the degree of the neighbor. Let v_j be a neighbor of v_i (i. e., $e_{ij} = (v_i, v_j) \in E$), then the probability of sampling v_j is $p' = \frac{k_j}{2|E|}$. Then we define $k_{\mathcal{N}i} = \frac{\sum_{j=1}^{k_i} k_j}{k_i}$ to be the average degree of the neighbors of node v_i . Note that $k_{\mathcal{N}i} \geq 1$. In this context, $k_{\mathcal{N}i}$ represents the local assortativity of node v_i , so then:

$$E_{ES_i}[k_i^s] = \sum_{j=1}^{k_i} p' \cdot n$$

$$\begin{aligned}
&= \frac{n}{N} \cdot \frac{\sum_{j=1}^{k_i} k_j}{k_{avg}} \\
&= \frac{n}{N} \cdot \frac{k_i}{k_{avg}} \cdot \frac{\sum_{j=1}^{k_i} k_j}{k_i} \\
&= k_i \cdot \frac{n}{N} \cdot \frac{k_{\mathcal{N}i}}{k_{avg}}
\end{aligned}$$

This leads us to Lemma 3.3.2, which shows when the sampled degrees using ES-i are higher than uniform random sampling.

Lemma 3.3.2 *The expected sampled degree in V_s using ES-i is greater than the expected sampled degree based on uniform node sampling when the local assortativity of the node is high. Let $k_{avg} = \frac{2|E|}{N}$ be the average degree in G and let $k_{\mathcal{N}i} = \frac{\sum_{j=1}^{k_i} k_j}{k_i}$ be the average degree of v_i 's neighbors in G . For any node $v_i \in V_s$, if $k_{\mathcal{N}i} \geq k_{avg}$, then $E_{ES_i}[k_i^s] \geq E_{UN}[k_i^s]$.*

Proof Consider the threshold k at which the expected value of k^s using ES-i sampling is greater than the expected value of k^s using uniform random sampling:

$$\begin{aligned}
E_{ES_i}[k_i^s] - E_{UN}[k_i^s] &= k_i \cdot \frac{n}{N} \cdot \frac{k_{\mathcal{N}i}}{k_{avg}} - k_i \cdot \frac{n}{N} \\
&= k_i \cdot \frac{n}{N} \left[\frac{k_{\mathcal{N}i}}{k_{avg}} - 1 \right] \\
&\geq 0 \quad \text{when } k_{\mathcal{N}i} \geq k_{avg}
\end{aligned}$$

■

Generally, for any sampled node v_i , if the average degree of v_i 's neighbors is greater than the average degree of G , then its expected sampled degree under ES-i is higher than it would be if uniform sampling was used. This is often the case in many real networks where high degree nodes are connected with other high degree nodes.

In Figure 3.1, we empirically investigate the difference between the sampled degrees k^s and original degrees k for an example network—the CONDMAT graph.

Specifically, in Figure 3.1a, we compare the cumulative degree distribution (CDF) of G when measured from the full set of nodes V to the CDF of G when measured only from the set of sampled nodes V_s . To contrast this with the *sampled* degrees, in Figure 3.1b we compare the CDF of G_s to the CDF of G (measured from the full set of nodes V). Note that in Figure 3.1, the x-axis denotes the degree (k in Figure 3.1a, and k^s in Figure 3.1b), and the y-axis denotes the cumulative probability ($P(X < x)$).

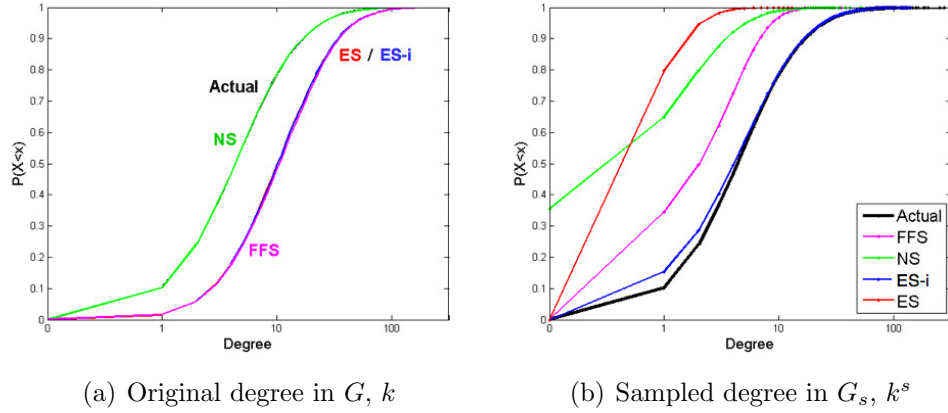


Fig. 3.1.: Illustration of original degrees (in G) vs. sampled degrees (in G_s) for subgraphs selected by NS, ES, ES-i, and FFS on the CondMAT network.

In Figure 3.1a, the NS curve is directly on top of the *Actual* CDF, indicating that NS accurately estimates the true degree distribution as observed in G . However, in Figure 3.1b, the NS curve is skewed upwards, indicating that NS underestimates the sampled degree distribution in G_s . On the other hand, in Figure 3.1a ES, FFS, and ES-i all overestimate the true degree distribution in G , which is expected since they are biased to selecting higher degree nodes. At the same time, in Figure 3.1b, ES and FFS both underestimate the sampled degree distribution G_s . In contrast to other sampling methods, ES-i comes closer to replicating the original degree distribution of G in G_s . This is from the combination of node selection bias (toward high degree nodes) with further augmentation through graph induction, which help ES-i to compensate for the underestimation caused by sampling subgraphs.

Table 3.1.: Characteristics of Network Data Sets

Graph	Nodes	Edges	Weak Comps.	Avg. Path	Density	Clustering Coeff.
HEPPH	34,546	420,877	61	4.33	7×10^{-4}	0.146
CONDMAT	23,133	93,439	567	5.35	4×10^{-4}	0.264
TWITTER	8,581	27,889	162	4.17	7×10^{-4}	0.061
FACEBOOK	46,952	183,412	842	5.6	2×10^{-4}	0.085
FLICKR	820,878	6,625,280	1	6.5	1.9×10^{-5}	0.116
LIVEJOURNAL	4,847,571	68,993,773	1876	6.5	5.8×10^{-6}	0.288
YOUTUBE	1,134,890	2,987,624	1	5.29	2.3×10^{-6}	0.006
POKEC	1,632,803	30,622,564	1	4.63	1.2×10^{-5}	0.047
WEB-STANFORD	281,903	2,312,497	1	6.93	2.9×10^{-5}	0.008
EMAIL-UNIV	214,893	1,270,285	24	3.91	5.5×10^{-5}	0.002

3.4 Experiments

In this section, we present results evaluating the various sampling methods on static graphs. We compare the performance of our proposed algorithm ES-i to other algorithms from each class (as discussed in section 2.3): node sampling (NS), edge sampling (ES) and forest fire sampling (FFS).

We compare the algorithms on seven different real-world networks. We use online social networks from FACEBOOK in the city of New Orleans [12] and FLICKR [135]. We use a social media network drawn from TWITTER, corresponding to users tweets surrounding the United Nations Climate Change Conference in Copenhagen, December 2009 (*#cop15*) [54]. Also, we use a citation graph from ArXiv HEPPH and a collaboration graph from CONDMAT [136]. We consider an email network EMAIL-UNIV that corresponds to a month of email communication collected from Purdue university mail-servers [55]. Finally, we compare the methods on a large social network from LIVEJOURNAL [136] with 4 million nodes (included only at the 20% sample size). Table 3.1 provides a summary of the global statistics of the network data sets.

Below we discuss the experimental results. For each experiment, we applied the sampling methods to the full network and sampled subgraphs over a range of sampling fractions $\phi = [5\%, 40\%]$. For each sampling fraction, we report the average results over ten different trials.

3.4.1 Distance Metrics

Figures 3.2(a)–3.2(d) show the average KS statistic for degree, path length, clustering coefficient, and k-core distributions on the six data sets. Generally, ES-i outperforms the other methods for each of the four distributions. FFS performs similar to ES-i in the degree distribution, however, it does not perform well for path length, clustering coefficient, and k-core distributions. This implies that FFS can capture the degree distribution but not connectivity between the sampled nodes. NS performs better than FFS and ES for path length, clustering coefficient, and k-core statistics but not for the degree statistics. This is due to the uniform sampling of the nodes that makes NS more likely to sample fewer high degree nodes (as discussed in 3.3). Clearly, as the sample size increases, NS is able to select more nodes and thus the KS statistic decreases. ES-i and NS perform similarly for path length distribution. This is because they both form a fully induced subgraph out of the sampled nodes. Since induced subgraphs are more connected, the distance between pairs of nodes is generally smaller.

In addition, we also used skew divergence as a second evaluation measure. Figures 3.2(e)–3.2(h) show the average skew divergence statistic for degree, path length, clustering coefficient, and k-core distributions on the six data sets. Note that skew divergence computes the divergence between the sampled and the real distributions on the entire support of the distributions. While the skew divergence is similar to KS statistic in that it still shows ES-i outperforming the other methods, it also shows significant gains in some cases, indicating that ES-i produces samples that capture the entire distribution more accurately.

Finally, Figures 3.2(i) and 3.2(j) show the L1 and L2 distances for eigenvalues and network values respectively. Clearly, ES-i outperforms all the other methods that fail to improve their performance even when the sample size is increased up to 40% of the full graph.

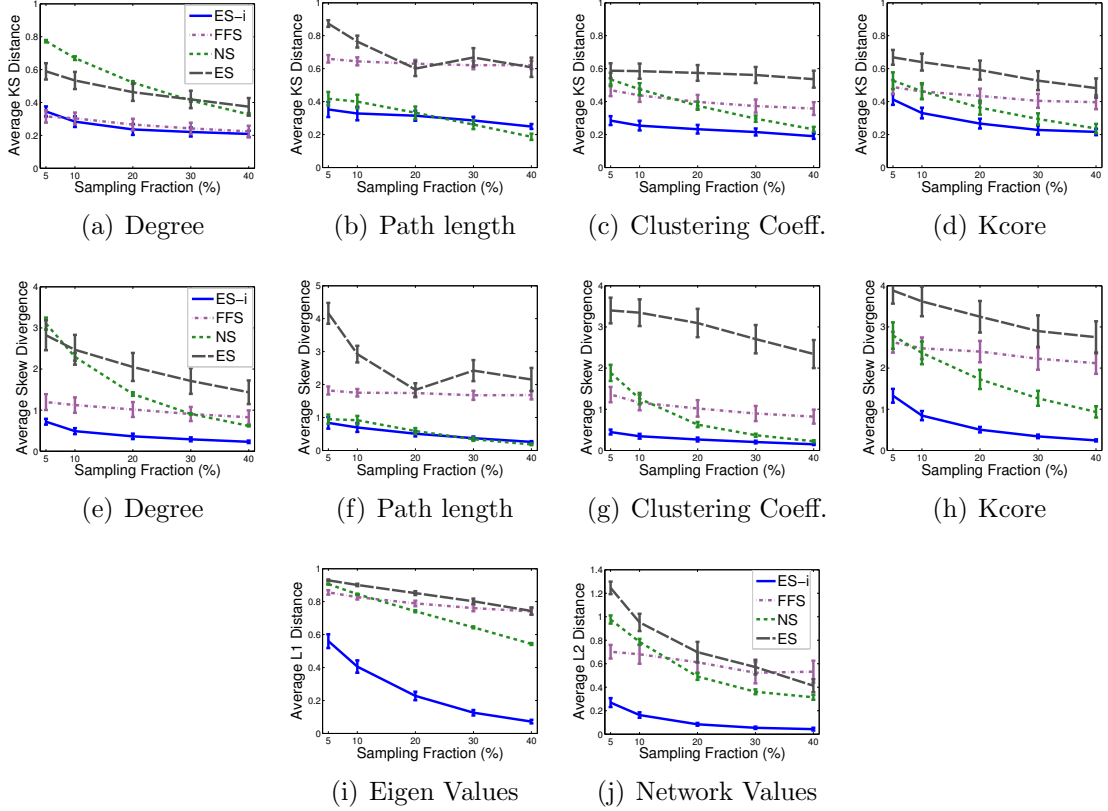


Fig. 3.2.: (a-d) Average KS distance, (e-h) average skew divergence, and (i-j) average L1 and L2 distance respectively, across 6 data sets.

3.4.2 Statistical Distributions

While the distance measures are important to quantify the divergence between the sampled and the real distributions, by analyzing only the distance measures it is unclear whether the sampled statistics are an over-estimate or under-estimate of the original statistics. Therefore, we plot the distributions for all networks at the 20% sample size. We choose the 20% as a representative sample size, however, we note that same conclusions hold for the other sample sizes. Note that we plot the *complementary CDF (CCDF)* for degree and k-core distributions, *CDF* for path length and clustering coefficient distribution, and we plot eigenvalues and network values versus their rank. Figures 3.3, 3.5, 3.4, 3.6, 3.7, 3.8, and 3.9, show the

Table 3.2.: Comparison of the maximum kcore number at the 20% sample size for ES-i, NS, ES, FFS versus the original value in G

Graph	True	ES-i	NS	ES	FFS
HEPPH	30	23*	8	2	4
CONDMAT	25	20*	7	2	6
TWITTER	18	18*	5	2	3
FACEBOOK	16	14*	4	2	3
FLICKR	406	406*	83	21	7
LIVEJOURNAL	372	372*	84	6	7
EMAIL-UNIV	47	46*	15	3	7

distributions for all networks. Next, we discuss the main findings we can observe from inspecting the distributions.

Degree Distribution. Across all networks, ES-i captures the tail of the degree distribution (high degree nodes) better than NS, ES, and FFS. However, ES-i under-estimates the low degree nodes for TWITTER, EMAIL-UNIV, and FLICKR. FFS and NS capture a large fraction of low degree nodes but they fail to capture the high degree nodes.

Path length Distribution. ES-i preserves the path length distribution of HEPH, CONDMAT, and LIVEJOURNAL, however, it underestimates the distributions of TWITTER, EMAIL-UNIV, and FLICKR. Conversely, NS over-estimates the distributions of HEPH, CONDMAT, and LIVEJOURNAL but successfully preserves the distributions of the other data sets.

Clustering Coefficient Distribution. ES-i generally captures the clustering coefficient more accurately than other methods. While ES-i under-estimates the low clustering coefficients, particularly in EMAIL-UNIV, and FLICKR, the other methods fail to capture the clustered structures in almost all the data sets.

K-Core Distribution. Similar to the previous statistics, ES-i nicely preserves the distribution of core sizes for HEPH, CONDMAT, and FACEBOOK, but it over-estimates the core structures of the other data sets. On the other hand, NS,

ES, and FFS generally fail to capture the core structures for the majority of the data sets (with the exception of FLICKR). In addition to the distribution of the core sizes, we compared the *max-core* number in the sampled graphs to their real counterparts for the 20% sample size (Table 3.2). Note that the *max-core number* is the maximum value of k in the k -core distribution. In contrast to ES-i, the *max-core number* in samples for NS, ES, and FFS is consistently an order of magnitude smaller than the real *max-core number*. This indicates that NS, ES, and FFS do not preserve the local density in the sampled subgraph structures.

Eigen Values. The NS, ES, and FFS methods generally fail to approximate the eigenvalues of the original graph in the sample. In contrast, ES-i accurately approximates the eigenvalues of TWITTER, EMAIL-UNIV, FLICKR, and LIVE-JOURNAL and closely approximates the eigenvalues of HEPH, CONDMAT, and FACEBOOK (at 20% sample size). By the *interlacing* theorem of the eigenvalues of induced subgraphs [137], the eigenvalues of ES-i in G_s can be used to estimate bounds on their counterparts in the input graph G : $\lambda_i \leq \mu_i \leq \lambda_{i+(N-n)}$ such that μ_i is the i^{th} eigenvalue of G_s , and λ_i is the i^{th} eigenvalue of G .

Network Values. Similar to the other graph measures, ES-i more accurately approximates the network values of the graph compared to other methods.

3.4.3 Comparison to Metropolis Graph Sampling

Metropolis sampling has been used frequently to improve the quality of collected samples. The main idea of Metropolis graph sampling is to draw a sample from the sample space X with a particular density, such that *good* samples will be sampled more frequently than bad ones. In [96], the authors used Markov chain sampling to improve the quality of uniform node sampling (NS). Their algorithm starts with an initial sample collected by NS, then the sampled nodes are replaced randomly

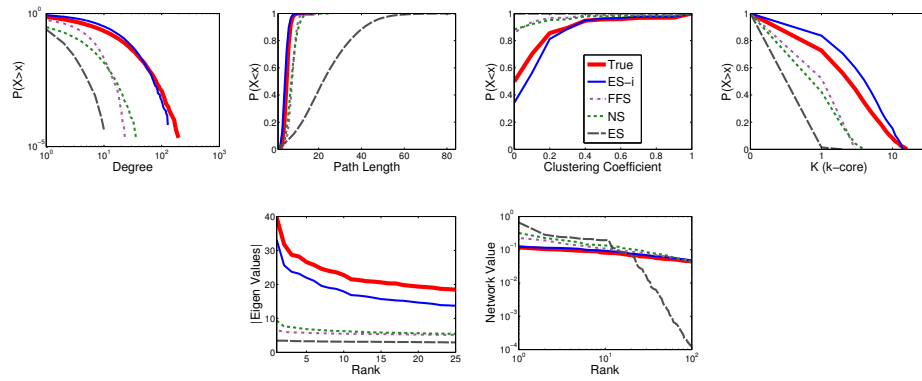


Fig. 3.3.: Sampling Distribution of FACEBOOK Graph

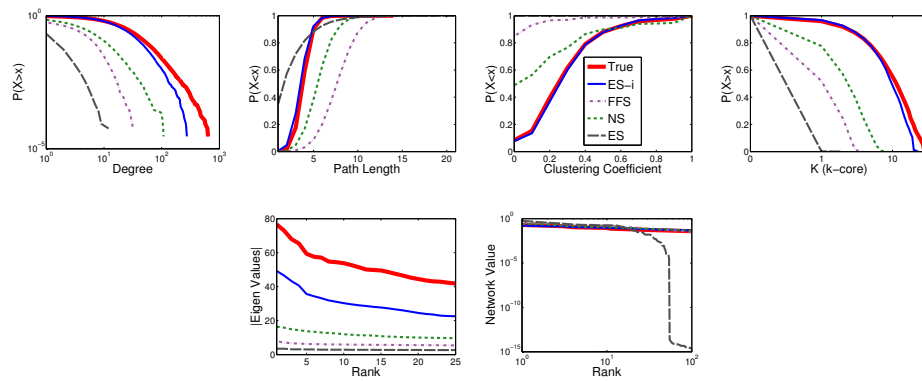


Fig. 3.4.: Sampling Distribution of HEPPI Graph

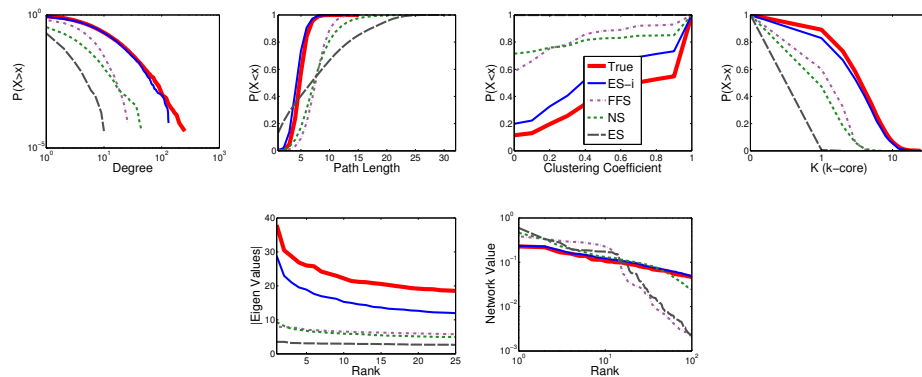


Fig. 3.5.: Sampling Distribution of CONDMAT Graph

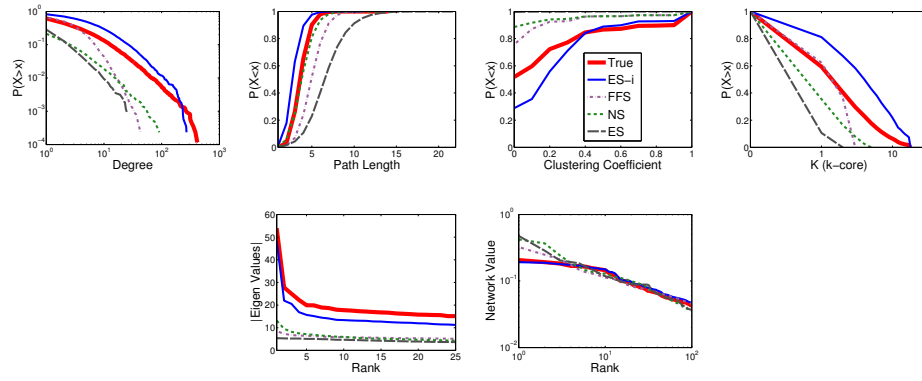


Fig. 3.6.: Sampling Distribution of TWITTER Graph

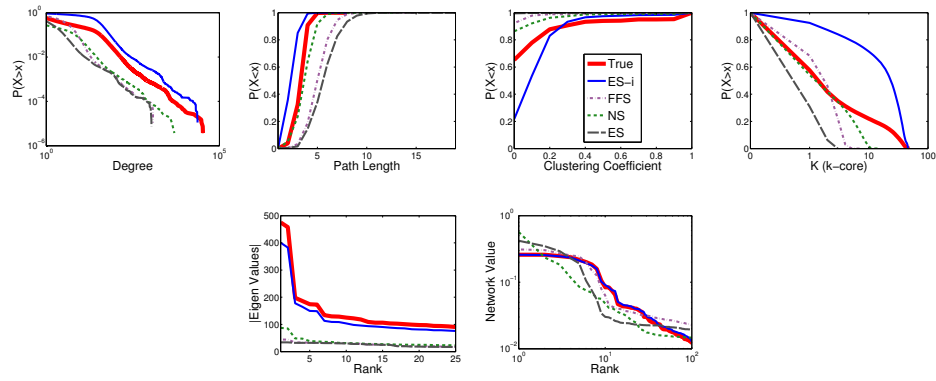


Fig. 3.7.: Sampling Distribution of EMAIL-UNIV Graph

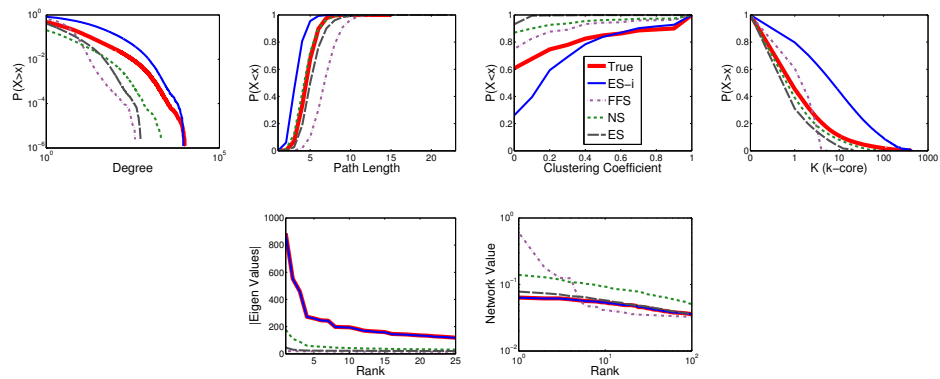


Fig. 3.8.: Sampling Distribution of FLICKR Graph

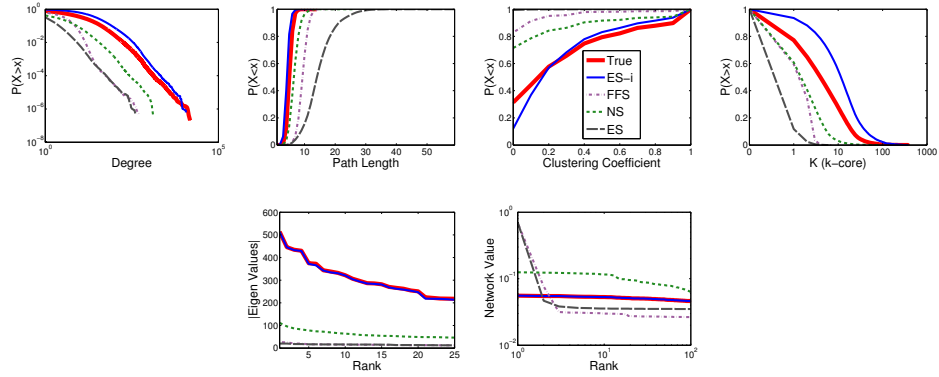


Fig. 3.9.: Sampling Distribution of LIVEJOURNAL Graph

such that that the new sampled subgraph G_s will better match the properties of the full graph G (e.g. degree distribution). Clearly, this method requires computing the properties of the full graph G in advance, and this requirement might be hard to satisfy when graphs are too large. In [138], we found that “bad” nodes (i.e., ones that do not improve the match) are sampled more frequently than “good” ones as the sample size increases (typically ≥ 1000 nodes), which makes it difficult for the method to converge. Consequently, this method produces graphs with properties that are comparable to node sampling.

Recently, neighbor reservoir sampling (NRS) has been proposed in [100] to sample *connected* subgraphs. The key idea is to start by an initial sample G_s (chosen by random-walk approaches), then nodes in G_s are randomly replaced by other potential nodes (chosen from the neighbors of G_s), such that the new sample subgraph is connected (i.e., G_s has a single component). Random-walk methods are generally biased to high degree nodes, and thus NRS helps to reduce their bias by randomly replacing nodes that were initially selected by random-walk methods.

In Table 3.3, we compare ES-i to neighbor reservoir sampling (NRS) for TWITTER, FACEBOOK, HEPH, and CONDMAT. Note that the results are the average of the 20% and 30% sample sizes. We show the average KS distance computed over the distributions of degree, path length, clustering, and kcore. We observe that NRS out-

Table 3.3.: Comparison of ES-i to Neighbor Reservoir (NRS) and Forest Fire Sampling (FFS-i) with graph induction

Data	ES-i	NRS	FFS-i
Average KS dist. (DEG, PL, CLUST, KCORE)			
TWITTER	0.2801	0.0631	0.3009
FACEBOOK	0.1918	0.1054	0.2650
HEPPH	0.0895	0.3321	0.1229
CONDMAT	0.1125	0.1918	0.1892
	0.1685	0.1731	0.2195
Average L1 dist. (EIGENVAL)			
TWITTER	0.1923	0.3323	0.2080
FACEBOOK	0.1647	0.4676	0.1671
HEPPH	0.3459	0.6512	0.3051
CONDMAT	0.2479	0.4812	0.1689
	0.2377	0.4831	0.2123
Average L2 dist. (NETVAL)			
TWITTER	0.0481	0.1029	0.0549
FACEBOOK	0.0787	0.2801	0.0759
HEPPH	0.1804	0.3637	0.2482
CONDMAT	0.0944	0.1750	0.0852
	0.1004	0.2304	0.1161

performs ES-i for sparse graphs (TWITTER and FACEBOOK). This shows that NRS is indeed effective to reduce the bias of random-walk sampling approaches. However, ES-i outperforms NRS for dense graphs (HEPPH and CONDMAT). We also compare the L1/L2 distances computed for the distribution of eigenvalues and network values respectively and we observe that ES-i outperforms NRS for the four data sets. We tried to use NRS for sampling very large graphs, but the method was slow and did not converge for any of our larger data sets (within a limit of 2 hours of running time). Further, we test the effect of graph induction on forest fire sampling (FFS-i) in Table 3.3. ES-i outperforms FFS-i on the average KS distance computed over the distributions of degree, path length, clustering, and kcore. For the L1/L2 distances, ES-i and FFS-i performs comparably.

3.5 Network Sampling Designs for Relational Classification

In Chapter 2, we discussed how network sampling arises in many different applications (e. g., social science). Most data mining research on network sampling has focused on how to collect a sample that closely match *topological* properties of the network [9,10]. However, since the topological properties are never entirely preserved, it is also important to study how the sampling process impacts applications overlaid on the sampled networks.

One such study recently investigated the impact of sampling methods on the investigation of information diffusion [105]. The study shows that sampling methods which considers both topology and user context improves discovery compared to other naive methods. In this chapter, we consider the impact of sampling on relational learning. Network sampling is a core part of relational learning since relational data is often represented as attributed graphs. Sampled relational data is used in many different contexts—including parameter estimation, active learning, collective inference, and algorithm evaluation.

Network sampling can produce samples with imbalance in class membership and/or bias in topological features (e. g., path length, clustering) due to missing nodes/edges—thus the sampling process can significantly impact the accuracy of relational classification methods. Biases may result from the size of the sample, the applied sampling method, or both. While, most previous work in relational learning has focused on analyzing a single *input network* and research has considered how to further split the input network into training and testing networks for evaluation [51,139,140], the fact that the input network is often itself sampled from an unknown target network has largely been ignored. There has been little focus on how the construction of the input networks may impact the performance of relational algorithms and evaluation methods [53].

In this chapter, we study the question of how the choice of the sampling method can impact *parameter estimation* and *performance evaluation* of relational classifica-

tion algorithms. We aim to evaluate the impact of network sampling on relational classification using two different goals:

1. *Parameter estimation:* we study the impact of network sampling on the estimation of class priors, i. e., goal 3.
2. *Performance evaluation:* we study the impact of network sampling on the estimation of classification accuracy of relational learners, i. e., goal 2.

Conventional classification algorithms focus on the problem of identifying the unknown class (e. g., group) to which an entity (e. g., person) belongs. Classification models are learned from a training set of entities, which are assumed to be independent and identically distributed (i.i.d.) and drawn from the underlying population of instances. However, relational classification problems differs from this conventional view in that entities violate the i.i.d. assumption. In relational data, entities (e. g., social network users) can exhibit complex dependencies. For example, friends often share similar interests (e. g., political views).

Recently, there have been a great deal of research in relational learning and classification. For example, the work in [59] and [141] outline probabilistic relational learning algorithms that search the space of relational attributes and neighbor correlations to improve classification accuracy. The work in [140] proposed a simple relational neighbor classifier (weighted-vote relational neighbor wvRN) that requires no learning and iteratively classifies the entities in a relational network based on the relational structure. Macskassy in [140] showed that wvRN often performs competitively when compared to other more complex relational learning algorithms.

3.5.1 Impact on Parameter Estimation

Let a be the node attribute representing the class label of any node $v_i \in V$ in graph G . We denote $\mathcal{C} = \{c_1, c_2, \dots\}$ as the set of possible class labels, where c_l is the class label of node v_i (i. e., $a(v_i) = c_l$).

We study the impact of network sampling on the estimation of class priors in G (i. e., the distribution of class labels), using the following procedure:

1. Choose a set of nodes S from V using a sampling algorithm σ .
2. For each node $v_i \in S$, observe v_i 's class label.
3. Estimate the class label distribution \hat{p}_{c_l} from S , using the following equation,

$$\hat{p}_{c_l} = \frac{1}{|S|} \sum_{v_i \in S} 1_{(a(v_i)=c_l)}$$

In the experiments, we consider four real networks: two citation networks CORA with 2708 nodes and CITESEER with 3312 nodes [142], FACEBOOK collected from Facebook Purdue network with 7315 users with their political views [143], and a single day snapshot of 1490 political blogs that shows the interactions between liberal and conservative blogs [144].

We sample a subset of the nodes using NS, ES, ES-i, FFS, and expansion sampling (XS). Expansion sampling (XS) is a snowball sampling method which samples nodes deterministically to maximize the sample expansion [28]. We varied the sample size from 10 – 80% of the size of the graph G . For each sample size, we take the average of ten different experiments. Then, we compare the estimated class prior to the actual class prior in the full graph G using the average KS distance measure. As shown in Figures 3.10(a)–3.10(d), node sampling (NS) estimates the class priors more accurately than other methods. In contrast, we note that FFS produces a large bias in most of the graphs at small sample sizes (ie, 10%).

While XS performs similar to ES-i in CORA, CITESEER, and FACEBOOK, it performs significantly worse when estimating the class priors for the political blogs network (at 10% sample size). In [144], the authors show there is a clear division between the liberal and conservative political blogs, and that 91% of the links originating within either the conservative or liberal communities stay within that com-

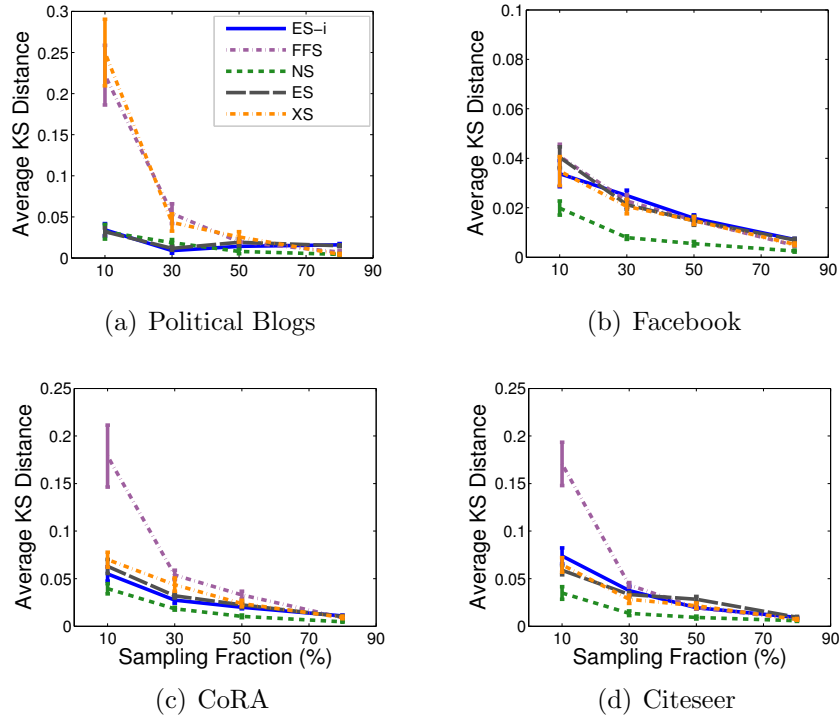


Fig. 3.10.: Average KS distance of class priors for NS, ES, FFS, XS, and ES-i.

munity. According to these observations, the political blogs network is structurally divided into two dense communities, and the nodes are homogeneously labeled within each community. We find in networks with such a dense community structure, XS (and FFS) are likely to be largely biased towards exploring only a small fraction of the communities. For instance, in the political blogs network, we find that at the 10% sample size, XS (and FFS) only sample from one of the two communities (either conservatives or liberals), effectively ignoring the other. This indicates that topology-based sampling methods (such as XS, FFS) may produce biased estimates of class priors from networks that have dense, homogeneously-labeled communities.

To solve this problem, we can modify the topology-based methods (like XS and FFS) to randomly jump with a small probability α to a new node in the graph (which can be chosen uniformly at random). We note that α should be set relative to the density within the communities and the sparsity of edges between communities.

3.5.2 Impact on Classification Accuracy

Let \mathcal{R} be a relational classifier which takes a graph G as input. The goal is to predict the class labels of nodes in G . Therefore, \mathcal{R} uses a proportion of nodes in graph G with known class labels as a *training set* to learn a model. Afterwards, \mathcal{R} is used to predict the label of the remaining (unlabeled) nodes in G (i.e., *test set*). Generally, the performance of \mathcal{R} can be evaluated based on the accuracy of the predicted class labels. In this work, we calculate the accuracy using area under the ROC curve (AUC) measure.

We study the impact of network sampling on the accuracy of relational classification using the following procedure:

1. Sample a subgraph G_s from G using a sampling algorithm σ .
2. Estimate the classification accuracy of a classifier \mathcal{R} on G_s : $\widehat{auc} = \mathcal{R}(G_s)$

We compare the actual classification accuracy on G to the estimated classification accuracy on G_s . Formally, we compare $auc = \mathcal{R}(G)$ to $\widehat{auc} = \mathcal{R}(G_s)$ and G_s is said to be representative to G , if $\widehat{auc} \approx auc$.

In our experiments, we use the weighted-vote relational neighbor classifier (wvRN) as our base classifier \mathcal{R} [140]. In wvRN, the class membership probability of a node v_i belonging to class c_l is defined as:

$$P(c_l|v_i) = \frac{1}{Z} \sum_{v_j \in \mathcal{N}(v_i)} w(v_i, v_j) * P(c_l|v_j)$$

where $\mathcal{N}(v_i)$ is the set of neighbors of node v_i , $w(v_i, v_j)$ is the weight of the edge $e_{ij} = (v_i, v_j)$, and $Z = \sum_{v_j \in \mathcal{N}(v_i)} w(v_i, v_j)$ is the normalization term.

We follow the common methodology used in [140] to compute the classification accuracy. First, we vary the proportion of randomly selected labeled nodes from 10–80% and use 5-fold cross validation to compute the average AUC. Then, we repeat this procedure for both the graph G and the sample subgraph G_s (at different sample

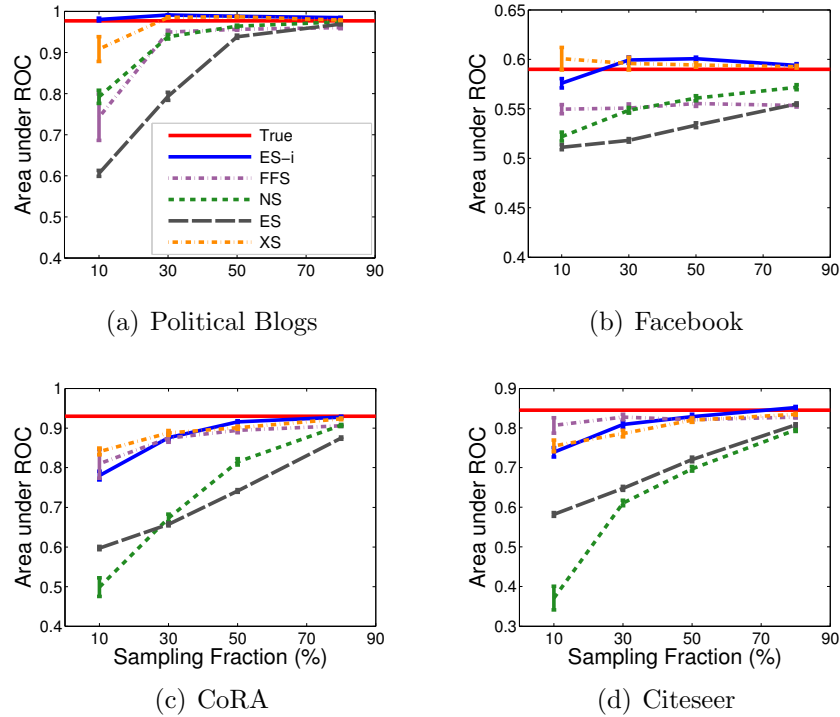


Fig. 3.11.: Classification accuracy versus sampling fraction, where 10% of nodes in the graph are initially labeled for prediction.

sizes). Note that AUC is calculated for the most prevalent class. Figures 3.11(a)-3.11(d) show the plots of AUC versus the sample size ($\phi = [10\%, 80\%]$) with 10% labeled nodes. Figures 3.12(a)-3.12(d) show the plots of AUC versus the proportion of labeled nodes, where the AUC is averaged over all sample sizes (10 – 80%). We observe that AUC of G is generally underestimated for sample sizes $< 30\%$ in the case of NS, ES, and FFS. However, generally ES-i and XS perform better than other sampling methods and converges to the “True” AUC in G . In Figures 3.11(b) and 3.12(b), ES-i and XS slightly overestimate the true AUC. These results show that in some cases when the graph is noisy (with low autocorrelation between labels of neighboring nodes), sampling of nodes and edges can enhance the performance of relational classification by reducing noise and overfitting.

We observe that many sampling methods fail to simultaneously satisfy the two different goals even though both are needed for relational learning applications (i. e.,

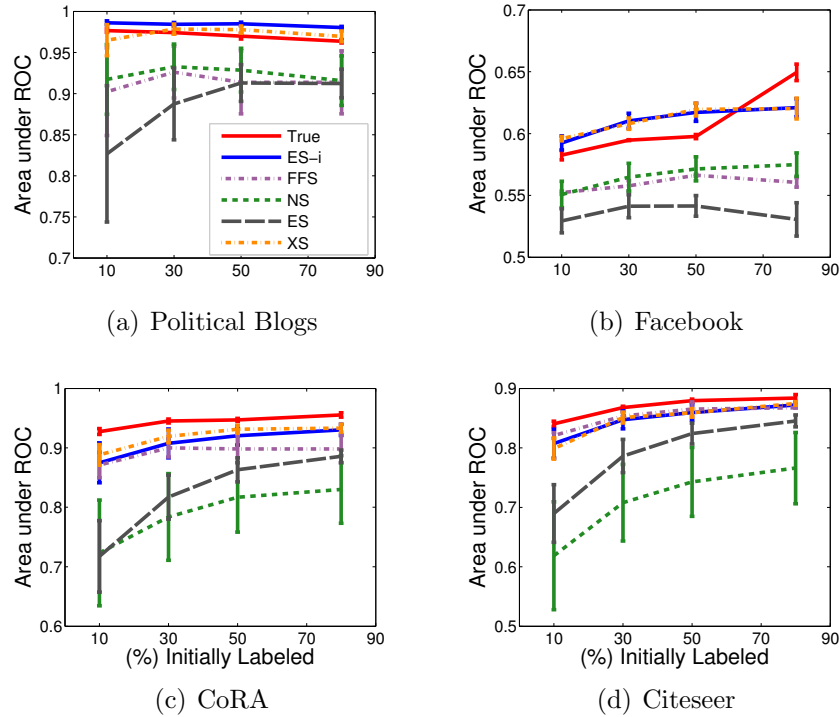


Fig. 3.12.: Classification accuracy versus proportion of nodes in the graph that are initially labeled for prediction, averaged over sample sizes 10 – 80%.

parameter estimation and accuracy evaluation). For example, while NS estimates the class priors better than other methods, it cannot accurately estimate classification accuracy due to lack of connectivity in the samples. Edge sampling performs similar to node sampling. On the other hand, while FFS and XS methods may be more accurate for estimating classification accuracy, they are generally not robust for estimating unbiased class priors (particularly for networks with homogeneously-labeled dense communities). ES-i provides a good balance for satisfying the two goals with a small bias at the smaller sample sizes.

3.6 Summary

In this chapter, we proposed a novel *2-pass* streaming network sampling algorithm. In addition, we studied the impact of various network sampling algorithms on the

performance of relational classification. Our contributions can be summarized in the following points:

- (1) Sampled subgraphs collected and constructed by our proposed algorithm (ES-i) accurately preserve a range of network statistics that capture both local and global distributions of the graph structure.
- (2) Due to its bias to selecting high degree nodes, ES-i generally favors dense and clustered areas of the graph, which results in connected sample subgraphs—in contrast with other methods.
- (3) uniform node/edge sampling, and forest fire sampling methods generally construct more sparsely connected sampled subgraphs.
- (4) Most sampling methods with the exception of our proposed algorithm (ES-i) fail to simultaneously satisfy the two different goals needed for relational classification tasks (i. e., parameter estimation and accuracy evaluation).

4. SAMPLING FROM STREAMING GRAPHS

In this chapter, we focus on how to sample a representative subgraph G_s from a streaming graph G in a single-pass. Note that in this work we focus on space-efficient sampling methods. Using the definition of a streaming graph sampling algorithm, as discussed in Section 2.5, we now present streaming variants of different sampling algorithms from Section 2.3.

4.1 Motivation

We live in a vastly connected world. A large percentage of world's population routinely use online applications (e.g., Facebook and instant messaging) that allow them to interact with their friends, family, colleagues and anybody else that they wish to. Analyzing various properties of these interconnection networks is a key aspect in managing these applications; for example, uncovering interesting dynamics often prove crucial for either enabling new services or making existing ones better. Since these interconnection networks are often modeled as graphs, and these networks are huge in practice (e.g., Facebook has more than a billion nodes), efficient *streaming methods* have recently become extremely important.

In addition, many interesting graphs in the online world naturally evolve over time, as new nodes join or new edges are added to the network. A natural representation of such graphs is in the form of a stream of edges. Clearly, in such a streaming graph model, sampling algorithms that process the data in one-pass are more efficient than those that process data in an arbitrary order. Even for static graphs, the streaming model is still applicable, with a one-pass algorithm for processing arbitrary queries over this graph, which is typically more efficient than those that involve arbitrary traversals through the graph.

4.2 Streaming Node Sampling

One key problem with traditional uniform node sampling (discussed in 2.3) is that the algorithm assumes that nodes can be accessed uniformly at random. In our stream setting, new nodes arrive into the system only when an edge that contains the new node is observed in the system. It is therefore difficult to identify which n nodes to select *a priori*. To address this, we utilize the idea of reservoir sampling [38] to implement a streaming variant of node sampling (see Algorithm 4.1).

Algorithm 4.1: STREAMING NODE SAMPLING $NS(n, S)$

Input : Sample Size n , Graph Stream S
Output: Sampled Subgraph $G_s = (V_s, E_s)$

```

1  $V_s = \emptyset, E_s = \emptyset$ 
2  $h$  is fixed uniform random hash function
3  $t = 1$ 
4 for  $e_t$  in the graph stream  $S$  do
5    $(u, v) = e_t$ 
6   if  $u \notin V_s$  &  $h(u)$  is top- $n$  min hash then
7      $V_s = V_s \cup u$ 
8     Let  $w \in V_s$  be the node s.t.  $h(w)$  is no longer a top- $n$  min hash
9      $V_s = V_s - \{w\}$ ; Remove all edges incident on  $w$  from  $E_s$ 
10  if  $v \notin V_s$  &  $h(v)$  is top- $n$  min hash then
11     $V_s = V_s \cup v$ 
12    Let  $w \in V_s$  be the node s.t.  $h(w)$  is no longer a top- $n$  min hash
13     $V_s = V_s - \{w\}$ ; Remove all edges incident on  $w$  from  $E_s$ 
14  if  $u, v \in V_s$  then
15     $E_s = E_s \cup e_t$ 
16   $t = t + 1$ 

```

The main idea is to select nodes uniformly at random with the help of a uniform random hash function – $h(v_i) \sim \text{Uniform}(0, 1)$. A uniform random hash function defines a true random permutation on the nodes in the graph, meaning that any node is equally likely to be the node with the minimum value. Specifically, we keep track of the nodes with the n smallest hash values in the graph.

Nodes are only added to the sample if their hash values are among the top- n minimum hashes seen thus far in the stream. Any edge that has both vertices already in the reservoir is automatically added to the original graph. Since the reservoir is finite, a node with smaller hash value may arrive late in the stream and replace a node that was sampled earlier. In this case, all edges incident to the replaced/dropped

node will be removed from the sampled subgraph. Once the reservoir is filled up to n nodes, it will remain at n nodes, but since selection is based on the hash values, nodes will be dropped and added as the algorithm samples from all portions of the stream (not just the front). Therefore, it guarantees a uniformly sampled set of nodes from the graph stream.

4.3 Streaming Edge Sampling

Streaming edge sampling can be implemented similar to streaming node sampling. Instead of hashing individual nodes, we focus on using hash-based selection of edges (as shown in Algorithm 4.2). We use the approach that was first proposed in [40].

Algorithm 4.2: STREAMING EDGE SAMPLING $ES(n, S)$

Input : Sample Size n , Edge Selection Size m , Graph Stream S
Output: Sampled Subgraph $G_s = (V_s, E_s)$

```

1  $V_s = \emptyset, E_s = \emptyset$ 
2  $h$  is fixed uniform random hash function
3  $t = 1$ 
4 for  $e_t$  in the graph stream  $S$  do
5    $(u, v) = e_t$ 
6   if  $h(e_t)$  is in top- $m$  min hash then
7      $E_s = E_s \cup e_t$ 
8      $V_s = V_s \cup \{u, v\}$ 
9     Let  $e_k \in E_s$  be the edge s.t.  $h(e_k)$  is no longer a top- $m$  min hash
10     $E_s = E_s - \{e_k\}$ ; Remove any nodes from  $V_s$  that have no incident edges
11    Iteratively remove edges in  $E_s$  in decreasing order until  $|V_s| = n$  nodes
12     $t = t + 1$ 

```

More precisely, if we are interested in sampling m edges at random from the stream, we can simply keep a reservoir of the m edges with minimum hash values. Thus, if a new edge streams into the system, we check if its hash value is less than the top- m minimum hash value. If it is not, the edge is not selected, otherwise it is added to the reservoir, replacing the edge with the previous highest hash value. One problem with this approach is that our goal is often in terms of sampling a certain number of nodes n .

Since we use a reservoir of edges, determining the value of m that provides n nodes is difficult. The value may also vary throughout the stream, depending on

which edges the algorithm ends up selecting. Note that the sampling fraction could also be specified in terms of fraction of edges; the choice of defining it in terms of nodes is somewhat arbitrary in that sense. For our comparison purposes, we ensured that we choose a large enough m such that the number of nodes was much higher than n , but later iteratively pruned out sampled edges with the maximum hash values until the target number of nodes n was reached.

4.4 Streaming Topology-Based Sampling

We also consider a streaming variant of a topology-based sampling algorithm. Specifically, we consider a simple BFS-based algorithm (shown in Algorithm 4.3) that works as follows. This algorithm essentially implements a simple breadth-first search on a sliding window of w edges in the stream.

In many respects, this algorithm is similar to the forest-fire sampling (FFS) algorithm. Just as in FFS, it starts at a random node in the graph and selects an edge, among all edges incident on that node within the sliding window, to burn (as in FFS parlance). For every edge burned, let v be the incident node at the other end of the burned edge.

We enqueue v onto a queue Q in order to get a chance to burn its incident edges within the window. For every new streaming edge observed, the sliding window moves one step, which means the oldest edge in the window is dropped and a new edge is added. (If that oldest edge was sampled, it will still be part of the sampled graph.) If as a result of the sliding window moving one step, the node has no more edges left to burn, then the burning process will dequeue a new node from Q . If the queue is empty, the process jumps to a random node within the sliding window (just as in FFS). This way, it does BFS as much as possible within a sliding window, with random jumps if there is no more edges left to explore. Note that there may be other ways to implement a one-pass streaming approach to topology-based sampling, but

since to our knowledge, there are no streaming methods in the literature, we include this as a reasonable approximation for comparison.

Algorithm 4.3: STREAMING BREADTH FIRST SAMPLING BFS($n, S, wsize$)

Input : Sample Size n , Graph Stream S , Window Size= $wsize$
Output: Sampled Subgraph $G_s = (V_s, E_s)$

```

1  $V_s = \emptyset; E_s = \emptyset; W = \emptyset$ 
2 Add the first  $wsize$  edges to  $W$ 
3  $t' = wsize$ 
4 Create a queue  $Q$ 
5 // uniformly sample a seed node from  $W$ 
6  $u = Uniform(V_W); V_s = V_s \cup \{u\}$ 
7 for  $e_t$  in the graph stream  $S$  starting at  $t'$  do
8   if  $W.incident\_edges(u) = \emptyset$  then
9     if  $Q \neq \emptyset$  then  $u = Q.dequeue()$ 
10    else  $u = Uniform(V_W)$ 
11    if  $u \notin V_s$  then  $V_s = V_s \cup \{u\}$ 
12  else
13    Sample  $e_s = (u, v)$  from  $W.incident\_edges(u)$ 
14     $E_s = E_s \cup e_s$ 
15     $V_s = V_s \cup \{v\}$ 
16     $W = W - \{e_s\}$ 
17    Enqueue  $v$  onto  $Q$ 
18  // Move the window  $W$ 
19   $W = W - \{e_{t-wsize}\}; W = W \cup \{e_t\}$ 
20  if  $|V_s| > n$  then
21    Retain  $[e] \subset E_s$  such that  $[e]$  has  $n$  nodes
22    Output  $G_s = (V_s, E_s)$ 
23   $t = t + 1$ 

```

This algorithm has a similar problem as the edge sampling variant in that it is difficult to control the exact number of sampled nodes and hence some additional pruning needs to be done at the end (see Algorithm 4.3).

4.5 Partially-Induced Edge Sampling (PIES)

We finally present our main algorithm called PIES that outperforms the above implementations of stream sampling algorithms. Our ES-i approach discussed in Chapter 3 outlines a sampling algorithm based on edge sampling concepts. A key advantage of using edge sampling is its bias towards high degree nodes. This upward bias helps offset the downward bias (caused by subgraph sampling) to some extent. Afterwards, forming the induced graph will help capture the connectivity among the sampled nodes.

Unfortunately, full graph induction in a streaming fashion is difficult, since node selection and graph induction requires at least two passes (when implemented in the obvious, straightforward way). Thus, instead of full induction of the edges between the sampled nodes, in a streaming environment we can utilize *partial* induction and combine edge-based node sampling with the graph induction (as shown in Algorithm 4.4) in a single pass. The partial induction step induces the sample in the forward direction only. In other words, it adds an edge among a pair of sampled nodes if it occurs *after* both the two nodes were added to the sample.

PIES aims to maintain a dynamic sample while the graph is streaming by utilizing the same reservoir sampling idea we have used before. In brief, we add the first set of edges in the stream to a *reservoir* and then the rest of the stream is processed by randomly replacing existing records in the reservoir. PIES runs over the stream in a single pass and adds deterministically the first m edges incident to n nodes to the sample. Once it achieves the target sample size of n , then for any streaming edge, it (probabilistically) adds the incident nodes to the sample by replacing other sampled nodes from the node set (selected uniformly at random). At each step, the algorithm will also add the edge to the sample if its two incident nodes are already in the sampled node set—producing a partial induction effect.

Algorithm 4.4: PIES(Sample Size n , Stream S)

Input : Sample Size n , Graph Stream S
Output: Sampled Subgraph $G_s = (V_s, E_s)$

```

1  $V_s = \emptyset, E_s = \emptyset$ 
2  $t = 1$ 
3 while graph is streaming do
4    $(u, v) = e_t$ 
5   if  $|V_s| < n$  then
6     if  $u \notin V_s$  then  $V_s = V_s \cup \{u\}$ 
7     if  $v \notin V_s$  then  $V_s = V_s \cup \{v\}$ 
8      $E_s = E_s \cup \{e_t\}$ 
9      $m = |E_s|$ 
10  else
11     $p_e = \frac{m}{t}$ 
12    draw  $r$  from continuous Uniform(0,1)
13    if  $r \leq p_e$  then
14      draw  $i$  and  $j$  from discrete Uniform[1,  $|V_s|$ ]
15      if  $u \notin V_s$  then  $V_s = V_s \cup \{u\}$ , drop node  $V_s[i]$  with all its incident edges
16      if  $v \notin V_s$  then  $V_s = V_s \cup \{v\}$ , drop node  $V_s[j]$  with all its incident edges
17    if  $u \in V_s$  AND  $v \in V_s$  then  $E_s = E_s \cup \{e_t\}$ 
18   $t = t + 1$ 

```

Next we discuss the properties of PIES to illustrate its characteristics.

1. *PIES is a two-phase sampling method.* A two-phase sampling method is a method in which an initial sample of units is selected from the population (e. g., the graph stream), and then a second sample is selected as a subsample of the first. PIES can be analyzed as a two-phase sampling method. The first phase samples edges (i. e., edge sampling) from the graph stream with probability $p_e = \frac{m}{t}$ if the edge is incident to at least one node that does not belong to the reservoir, where t is the variable representing the time of the stream and m is the number of initial edges in the reservoir. Also, an edge is sampled with probability $p_e = 1$ if the edge is incident to two nodes that belong to the reservoir. After that, the second phase samples a subgraph uniformly (i. e., node sampling) to maintain only n nodes in the reservoir (i. e., all nodes in the reservoir are equally likely to be sampled).
2. *PIES has a selection bias to high degree nodes.* PIES is biased to high degree nodes due to its first phase that relies on edge sampling. Naturally, edge sampling is biased towards high degree nodes since they tend to have more edges compared to lower degree nodes.
3. *PIES samples an induced subgraph uniformly from the sub-sampled edge stream $E'(t)$ at any time t in the stream.* At any time t in the graph stream E , PIES sub-samples $E(t)$ to $E'(t)$ (such that $|E'(t)| \leq |E(t)|$). Then, PIES samples a uniform induced subgraph from $E'(t)$, such that all nodes in $E'(t)$ have an equal chance to be selected. Now, that we have discussed the main properties of PIES, it can be easily adapted depending on a specific choice of the network sampling goal.

4.6 Experiments

In this section, we present results of sampling from streaming graphs observed as an arbitrarily ordered sequence of edges. The experimental setup is similar to what we used in section 3.4. We compare the performance of our proposed algorithm PIES to the streaming implementation of node (NS), edge (ES), and breadth-first search sampling (BFS) methods. Note that we implement breadth first search using a sliding window of 100 edges.

Similar to the experiments in section 3.4, we report average performance over ten different runs. To assess algorithm variation based on the edge sequence ordering, we randomly permute the edges in each run (while ensuring that all sampling methods use the same sequential order). Note that all streaming algorithms run in $O(|E|)$, and therefore, in addition to the data sets we used in section 3.4, we also compare the methods on large networks with millions of nodes/edges from YOUTUBE, POKEC, and WEB-STANFORD [136]. Note that large data sets are included only for 20% sample size as they take longer running time.

4.6.1 Distance Metrics

Figures 4.1(a)–4.1(d) show the average KS statistic for degree, path length, clustering coefficient, and k-core distributions as an average over the six data sets (similar to section 3.4). PIES outperforms all other methods for capturing the degree distribution. NS performs almost as well as PIES for path length, clustering coefficient, and k-core distributions. As we explained earlier in this Chapter, PIES is biased to high degree nodes (due to its first phase) compared to NS. Both BFS and ES perform the worst among the four methods. This shows that the limited observability of the graph structure using a window of 100 edges does not facilitate effective breadth-first search. While increasing the window size may help improve the performance of BFS, we did not explore this as our focus was primarily on space-efficient sampling.

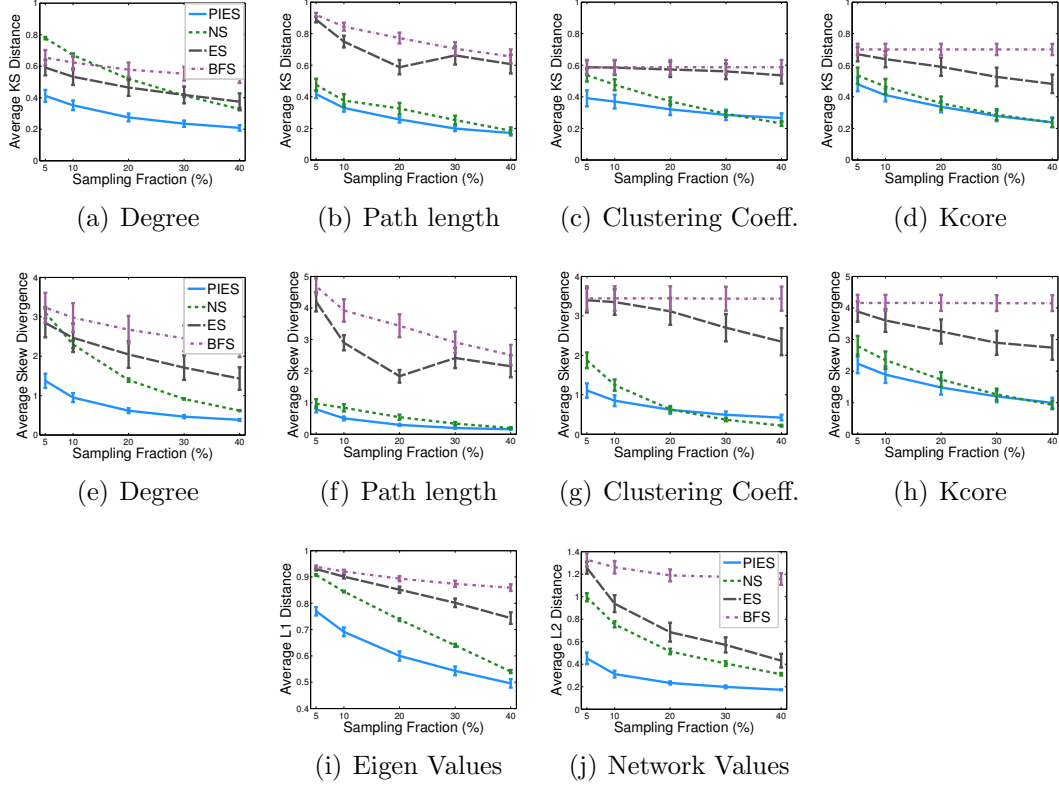


Fig. 4.1.: (a-d) KS distance, (e-h) average skew divergence, and (i-j) average L1 and L2 distance respectively, averaged across 6 graphs represented as edge streams.

Figures 4.1(e)–4.1(h) show the skew divergence results are similar to that of the KS statistic.

Finally, Figures 4.1(i)–4.1(j) show the L1 and L2 distance for eigenvalues and network values respectively. PIES outperforms all other methods. However, even though PIES performs the best, the distance is almost 50% for the eigenvalues. This implies PIES is not suitable for capturing the eigenvalues of the graph.

4.6.2 Statistical Distributions

We plot the distributions of the network statistics at the 20% sample size. Figures 4.2– 4.11 show the distributions across all data sets.

Degree Distribution. We observe across the all datasets, PIES outperforms the other methods for FACEBOOK, TWITTER, EMAIL-UNIV, FLICKR, LIVEJOURNAL, YOUTUBE, POKEC, and WEB-STANFORD. However, PIES only performs slightly better than NS for HEPH and CONDMAT. This behavior appears to be related to the specific properties of the network datasets themselves. HEPH and CONDMAT are more clustered and denser compared to other graphs used in the evaluation. We will discuss the behavior of the sampling methods for dense versus sparse graphs later in this section.

Path Length Distribution. PIES preserves the path length distribution of FACEBOOK, TWITTER, EMAIL-UNIV, FLICKR, LIVEJOURNAL, YOUTUBE, POKEC, and WEB-STANFORD, however, it overestimates the shortest path for HEPH and CONDMAT.

Clustering Distribution. PIES generally underestimates the clustering coefficient in the graph by missing some of the clustering surrounding the sampled nodes. This behavior is more clear in HEPH, CONDMAT, and WEB-STANFORD since they are more clustered initially.

K-Core Distribution. Similarly, PIES outperforms the other methods for FACEBOOK, TWITTER, LIVEJOURNAL, POKEC, YOUTUBE, and WEB-STANFORD. For HEPH and CONDMAT, PIES performs almost as good as NS. In addition to the distribution of the core sizes, we compared the *max-core number* in the sampled subgraphs to their real counterparts for the 20% sample size (Table 4.1).

Eigen Values. While PIES captures the eigenvalues better than ES and BFS, its eigenvalues are orders of magnitude smaller than the real graph’s eigenvalues. This shows that none of the streaming algorithms accurately captures the eigenvalues of the full graph (compared to ES-i in Chapter 3).

Network Values. PIES accurately estimates the network values of most graphs.

Table 4.1.: Comparison of *max-core-number* at the 20% sample size for PIES, NS, ES, BFS versus the original value in G

Graph	True	PIES	NS	ES	BFS
HEPPH	30	8*	8*	2	1
CONDMAT	25	7*	7*	2	1
TWITTER	18	7*	4	3	1
FACEBOOK	16	6*	4	2	1
FLICKR	406	166*	81	19	1
LIVEJOURNAL	372	117*	82	5	1
YOUTUBE	51	22*	12	5	2
POKEC	47	19*	13	2	1
WEB-STANFORD	71	33*	19	3	3
EMAIL-UNIV	47	22*	15	3	1

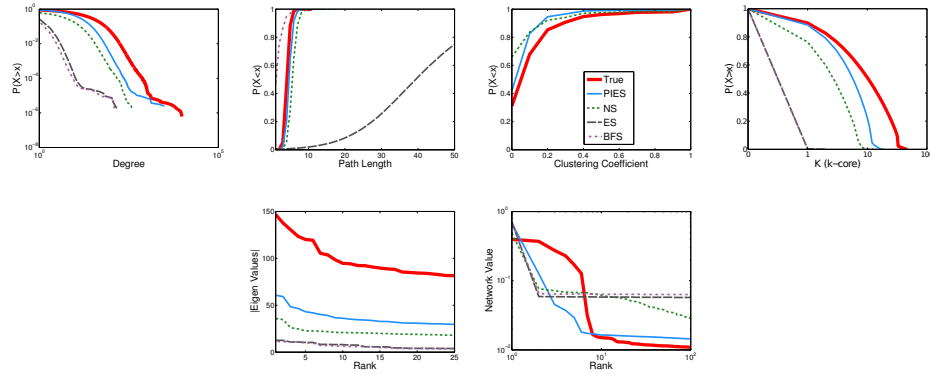


Fig. 4.2.: Stream Sampling Distribution of POKEC Graph

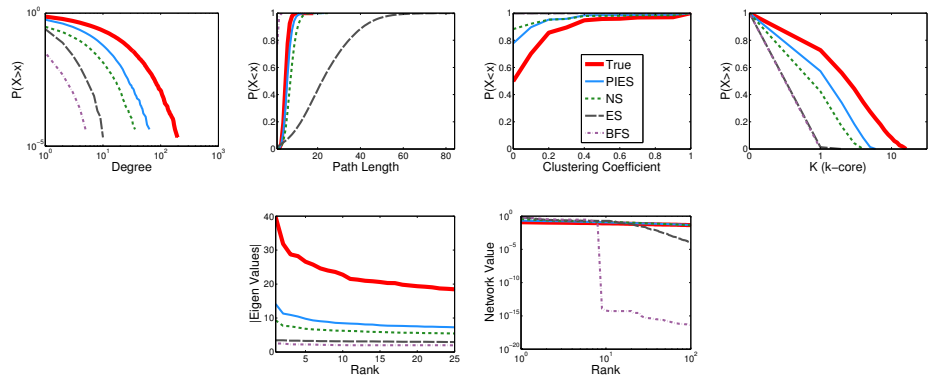


Fig. 4.3.: Stream Sampling Distribution of FACEBOOK Graph

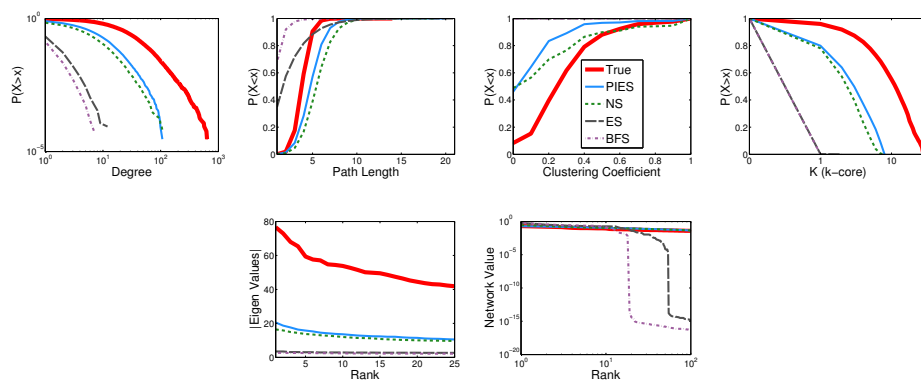


Fig. 4.4.: Stream Sampling Distribution of HEPH Graph

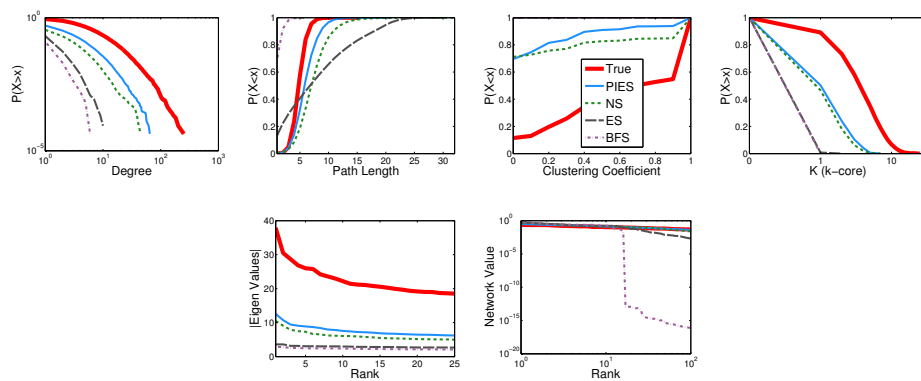


Fig. 4.5.: Stream Sampling Distribution of CONDMAT Graph

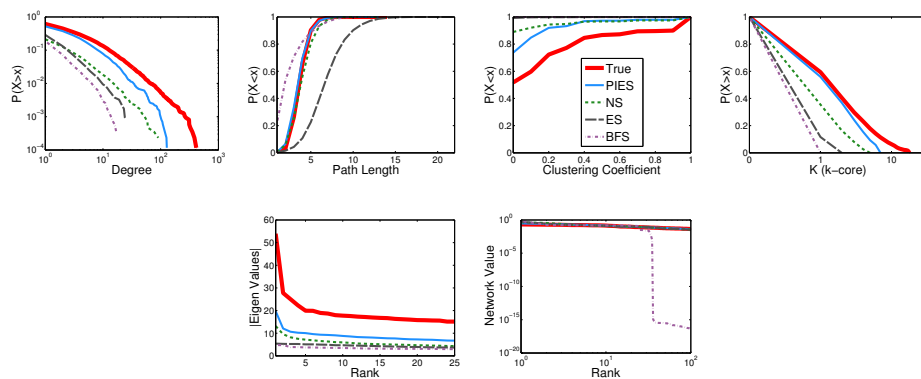


Fig. 4.6.: Stream Sampling Distribution of TWITTER Graph

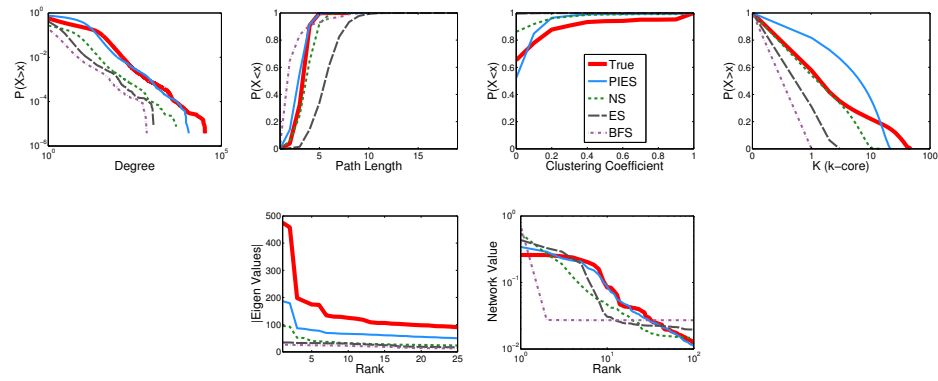


Fig. 4.7.: Stream Sampling Distribution of EMAIL-UNIV Graph

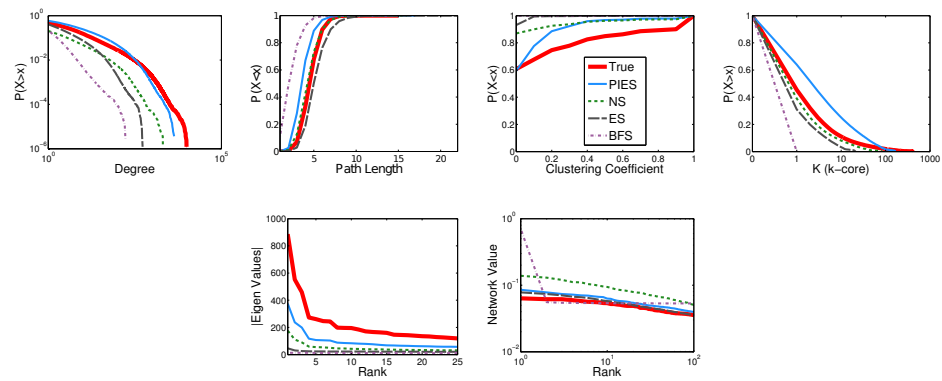


Fig. 4.8.: Stream Sampling Distribution of FLICKR Graph

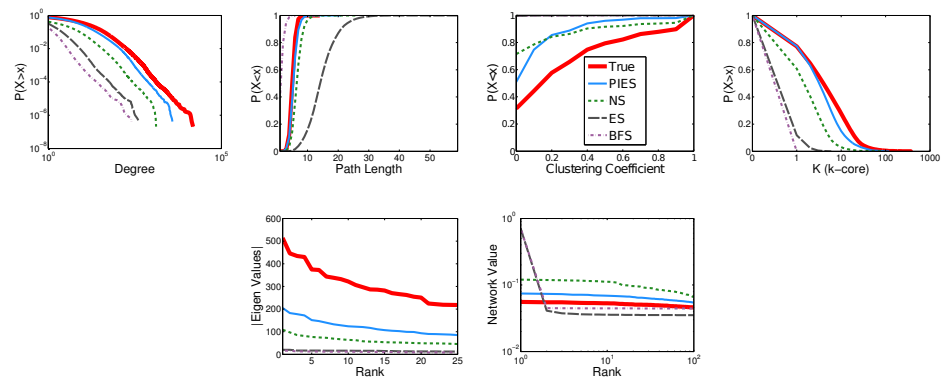


Fig. 4.9.: Stream Sampling Distribution of LIVEJOURNAL Graph

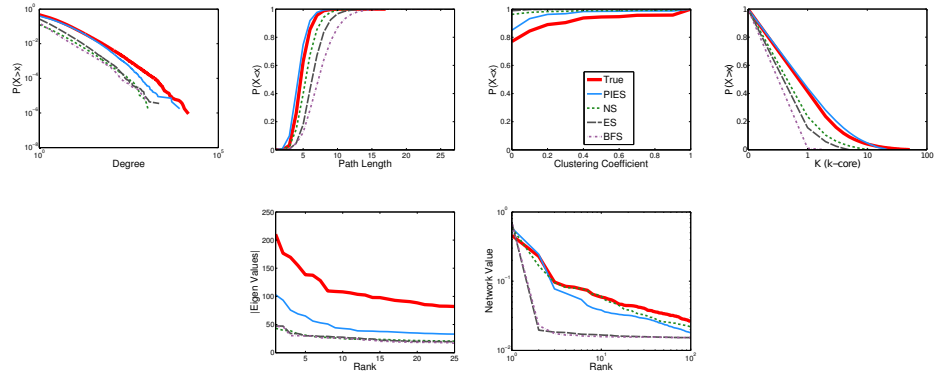


Fig. 4.10.: Stream Sampling Distribution of YOUTUBE Graph

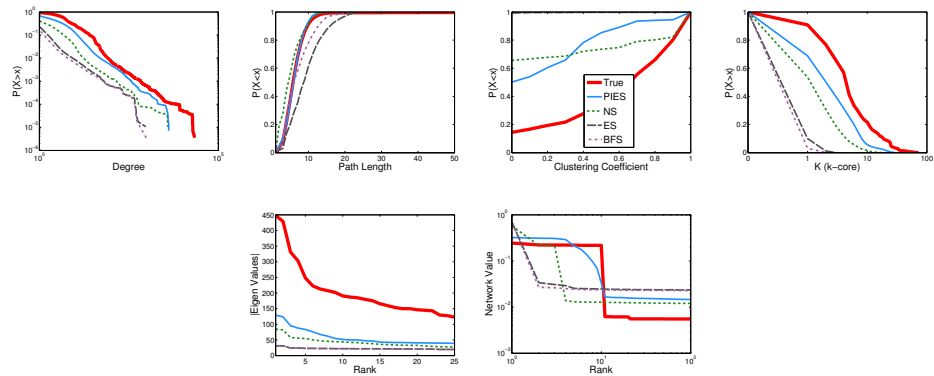


Fig. 4.11.: Stream Sampling Distribution of WEB-STANFORD Graph

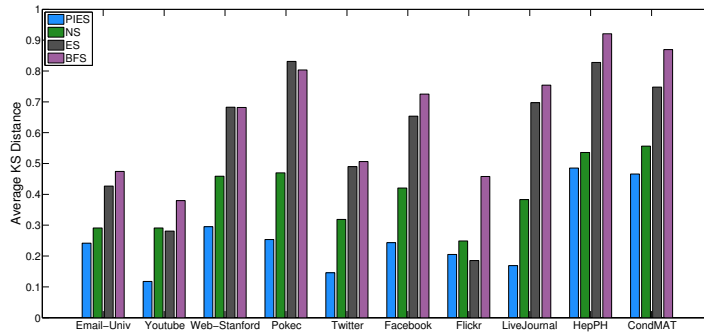


Fig. 4.12.: Average KS Statistics for different networks (sorted in increasing order of clustering/density from left to right).

4.6.3 Analysis of Dense Versus Sparse Graphs

Further to the discussion of the distributional results, we note that PIES is more accurate for sparse, less clustered graphs. To illustrate this, we report the performance of the stream sampling methods for each network in Figure 4.12, sorted from left to right in ascending order by clustering coefficient and density. Note that the bars represent the KS statistic (averaged over degree, path length, clustering, and k-core) for the 20% sample size.

Clearly, the KS statistic for all methods increases as the graph becomes more dense and clustered. PIES maintains a KS distance of approximately $\leq 26\%$ for eight out of ten networks. These results indicate that PIES performs better in networks that are generally sparse and less clustered. This interesting result shows that PIES will be more suitable to sample rapidly changing graph streams that have lower density and less clustering—which is likely to be the case for many large-scale dynamic communication and activity networks.

4.6.4 Analysis of Isolated Nodes

We also analyzed the number of isolated nodes for both NS and PIES in Table 4.2. Since both NS and PIES sample nodes independently, it is expected that their sampled

Table 4.2.: Average percentage of isolated nodes for NS & PIES at 20% sample.

Graph	PIES	NS
HEPPH	0.05	0.15
CONDMAT	0.14	0.36
TWITTER	0.15	0.51
FACEBOOK	0.13	0.43
FLICKR	0.14	0.56
LIVEJOURNAL	0.06	0.36
YOUTUBE	0.22	0.63
POKEC	0.01	0.21
WEB-STANFORD	0.08	0.32
EMAIL-UNIV	0.07	0.51

subgraph contains some nodes with zero degree (i. e., isolated nodes). This implies that PIES carries isolated nodes in the reservoir as the graph streams by. Clearly, each time a new edge is sampled from the stream, its incident nodes replace randomly selected nodes from the reservoir. This approach could replace high degree nodes while other isolated nodes still remain in the reservoir.

With this observation in mind, we propose a modification for PIES such that a newly added node replaces the node with minimum degree which has stayed in the reservoir the longest amount of time without acquiring more edges. This strategy favors retaining high degree nodes over isolated and/or low degree nodes in the sample. We show the results of this modification in Tables 4.3 and 4.4, which compare the KS distance, and L1/L2 distances respectively for each data set (averaged over the two reasonable sample sizes 20% and 30%). Note that we refer to the modification of PIES as “PIES (MIN)”.

The results show that modifying PIES in this manner achieves better results for dense graphs such as HEPPH and CONDMAT. Note that there is a trade-off between preventing nodes from being replaced (based on recency of adding nodes in the sample) and degree. We handle the trade-off between degree and recency by keeping two sets, one for the recent (not to be replaced) nodes, and the other for the oldest (can be replaced) nodes. When the number of the oldest nodes becomes

Table 4.3.: Average KS Distance for Stream Sampling Methods.

Data		PIES	PIES (MIN)	NS	ES	BFS
EMAIL-UNIV	<i>Deg</i>	0.2348	0.5803	0.4547	0.2186	0.3724
	<i>PL</i>	0.204	0.5621	0.19	0.6989	0.5114
	<i>Clust</i>	0.1108	0.4728	0.188	0.3302	0.3473
	<i>KCore</i>	0.2819	0.5821	0.1985	0.3219	0.5759
		0.2079*	0.5493	0.2578	0.3924	0.4518
TWITTER	<i>Deg</i>	0.1521	0.2598	0.4667	0.3052	0.4194
	<i>PL</i>	0.0528	0.3941	0.1243	0.617	0.4811
	<i>Clust</i>	0.2462	0.2269	0.346	0.4673	0.482
	<i>KCore</i>	0.1001	0.2886	0.2271	0.4393	0.5929
		0.1378*	0.2923	0.291	0.4572	0.4938
FACEBOOK	<i>Deg</i>	0.1848	0.2357	0.3804	0.4912	0.6917
	<i>PL</i>	0.2121	0.3171	0.4337	0.8762	0.9557
	<i>Clust</i>	0.2594	0.2314	0.3496	0.4975	0.5017
	<i>KCore</i>	0.2375	0.2447	0.3569	0.661	0.7275
		0.2234*	0.2572	0.3802	0.6315	0.7192
FLICKR	<i>Deg</i>	0.1503	0.399	0.514	0.0924	0.2706
	<i>PL</i>	0.2845	0.4936	0.0789	0.1487	0.6763
	<i>Clust</i>	0.1426	0.3754	0.2404	0.3156	0.3931
	<i>KCore</i>	0.1654	0.4289	0.0595	0.1295	0.4541
		0.1857	0.4242	0.2232	0.1716*	0.4485
HEPPH	<i>Deg</i>	0.4103	0.1304	0.483	0.8585	0.8923
	<i>PL</i>	0.306	0.1959	0.431	0.749	0.8676
	<i>Clust</i>	0.4636	0.0393	0.3441	0.9156	0.9171
	<i>KCore</i>	0.592	0.1674	0.6233	0.9402	0.9592
		0.443	0.1332*	0.4704	0.8658	0.909
CONDMAT	<i>Deg</i>	0.4042	0.1259	0.5006	0.6787	0.7471
	<i>PL</i>	0.2944	0.2758	0.5211	0.6981	0.9205
	<i>Clust</i>	0.5927	0.3285	0.5341	0.878	0.8853
	<i>KCore</i>	0.4692	0.1512	0.4955	0.858	0.8909
		0.4401	0.2203*	0.5128	0.7782	0.8609
Average for all Data sets		0.2730*	0.3128	0.3559	0.5494	0.6472

Table 4.4.: Average L1/L2 Distance for Stream Sampling Methods.

Data		PIES	PIES (MIN)	NS	ES	BFS
EMAIL-UNIV	<i>Eigen</i>	0.4487	0.074*	0.7018	0.7588	0.838
	<i>NetVal</i>	0.199	0.0201*	0.5785	0.3799	1.007
TWITTER	<i>Eigen</i>	0.4981	0.1851*	0.6411	0.7217	0.7964
	<i>NetVal</i>	0.1431	0.043*	0.3108	0.4271	0.8385
FACEBOOK	<i>Eigen</i>	0.591	0.1143*	0.6771	0.8417	0.9018
	<i>NetVal</i>	0.306	0.0617*	0.5383	1.0984	1.5027
FLICKR	<i>Eigen</i>	0.5503	0.0049*	0.7227	0.8491	0.9298
	<i>NetVal</i>	0.1657	0.0005*	0.5626	0.0574	1.5193
HEPPH	<i>Eigen</i>	0.7083	0.2825*	0.7232	0.9373	0.95
	<i>NetVal</i>	0.3	0.1817*	0.3198	1.0821	1.2477
CONDMAT	<i>Eigen</i>	0.6278	0.1475*	0.6843	0.8507	0.8875
	<i>NetVal</i>	0.2254	0.06*	0.3235	0.7514	0.9853

Table 4.5.: Characteristics of Multigraph Data Sets

Graph	Nodes	Edges	Avg. Node Strength	Avg. Edge Weight	Density	Global Clust.
FACEBOOK-CITY	46,952	855,542	37.4	4.7	4×10^{-4}	0.085
FACEBOOK-UNIV	49,825	1,518,155	60.9	4.2	6×10^{-4}	0.060
TWITTER-COP15	8,578	45,771	10.7	1.6	6×10^{-4}	0.061
RETWEET-POL	18,470	61,157	6.6	1.3	2×10^{-4}	0.027
INFECTIOUS-SOCIO	10,972	415,912	75.8	9.3	3×10^{-3}	0.44

less than 50% of the total sample size, we merge the two sets, and consider all nodes amenable to replacement, and replace the node with minimum degree.

4.7 Sampling from Multigraph Streams

In the previous sections, we focused on sampling a representative subgraph from *simple* graph streams, i. e., graph streams where edges appear only once, similar to the problem definition studied in [43]. In time-evolving graph streams, there are often two characteristics to study:

1. Graph structure
2. Graph dynamics

Studying simple graph streams is particularly focused on the *structure* of the streaming graph G (i. e., topological properties such as degree, clustering), while ignoring the *dynamics* that take place on top of the graph structure. These dynamics describe the strength (frequency) of communications between pairs of nodes, as well as the strength of individual nodes over time. Therefore, we also study the problem of sampling from *multigraph* streams, i. e., graph streams where edges may appear more than once, similar to the problem definition studied in [129].

Formally, let $G = (V, E)$ be a snapshot of streaming undirected graph of time length T , such that $E = \{e_t, \forall t \in [1, T]\}$. We denote by w the weight of an edge $e = (u, v)$, i. e., the number of times an edge e appeared in the stream, and we denote

by s the strength of a node v , equal to the sum of the weights of the incident edges to node v , i. e., $s = \sum_{(u,v) \in E} w$. We also denote by k the degree of node v , i. e., the number of unique neighbors of v . These functions previously used by [145].

Table 4.5 describes the characteristics of five real multigraph datasets. We use graph streams (with real timestamps), from Facebook wall communications in the city network of New Orleans—FACEBOOK-CITY graph [30], from Facebook wall communications in the university network of Purdue University—FACEBOOK-UNIV graph [54], from the Twitter hashtag *#cop15*—TWITTER-COP15 graph [54], from the retweets collected from Twitter hashtags with political content—RETWEET-POL graph [146], and from face-to-face interactions at the INFECTIOUS exhibition in Dublin, Ireland—INFECTIOUS-SOCIO graph [147].

In the experimental setup, we used the *real* timestamps of the datasets and we ran ten experiments for each data set. In each experiment, we sample 20% of the total number of nodes that appear in the stream as it is progressing. We evaluate the quality of the sample at different time points in the stream, from 40% to 100% of the stream length. Note that the stream length is the number of edges in the streaming graph ordered by timestamps. For example, if the total number of edges in the stream is 1000 edges, we evaluate using the graph sampled from the first 400, 600, 800, and 1000 edges respectively.

4.7.1 Kolmogorov-Smirnov Statistic

Figure 4.13 shows the average Kolmogorov-Smirnov (KS) statistics over the five datasets, while the stream is progressing. Figures 4.13(a) and 4.13(b) show the KS statistics of the distributions of node strength and edge weight respectively. The results show that PIES performs consistently well for both node strength and edge strength. In contrast, NS and ES each only perform well in one of the two properties.

In addition to the dynamics properties, we also evaluate the quality of the sample using the structural properties. Figures 4.13(c)-4.13(f) show the KS statistics for the

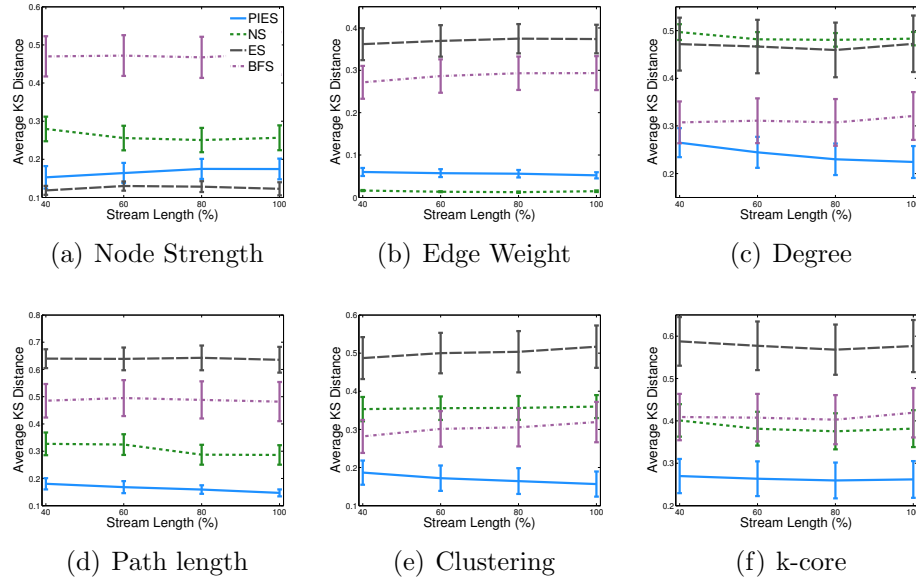


Fig. 4.13.: Average KS distance of across 5 datasets vs. percentage of stream length.

distributions of degree, path length, clustering, and kcore. These results show that PIES captures both the structure and dynamic properties as the stream is progressing.

4.7.2 Statistical Distributions of Temporal Properties

Figures 4.14 and 4.15 show the CCDF distributions of node strength and edge weight respectively at the 20% sample, constructed using the entire stream (i. e., 100% of the stream length). The results of FACEBOOK-CITY and INFECTIOUS-SOCIO show that ES captures the distribution of node strength but over-estimates the distribution of edge weight. Also, the results show that NS captures the distribution of edge weight but under-estimates the distribution of node strength. However, PIES again balances the estimation of both node strength and edge weight, producing a more accurate sample overall. Notably, BFS performs significantly worse than the other methods.

These results are not surprising since ES (unlike NS) is naturally biased towards edges with high weights as they appear more frequently in the stream. We found that in all the datasets, there is a strong positive correlation between the node strength

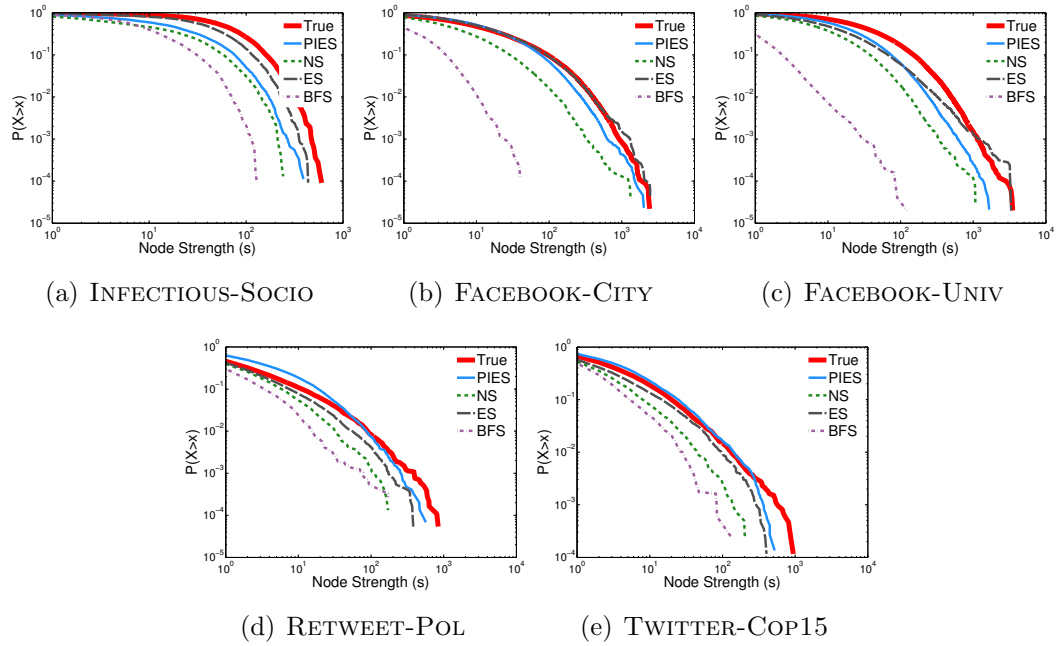


Fig. 4.14.: Distributions of node strength in multigraph streams at 20% sample.

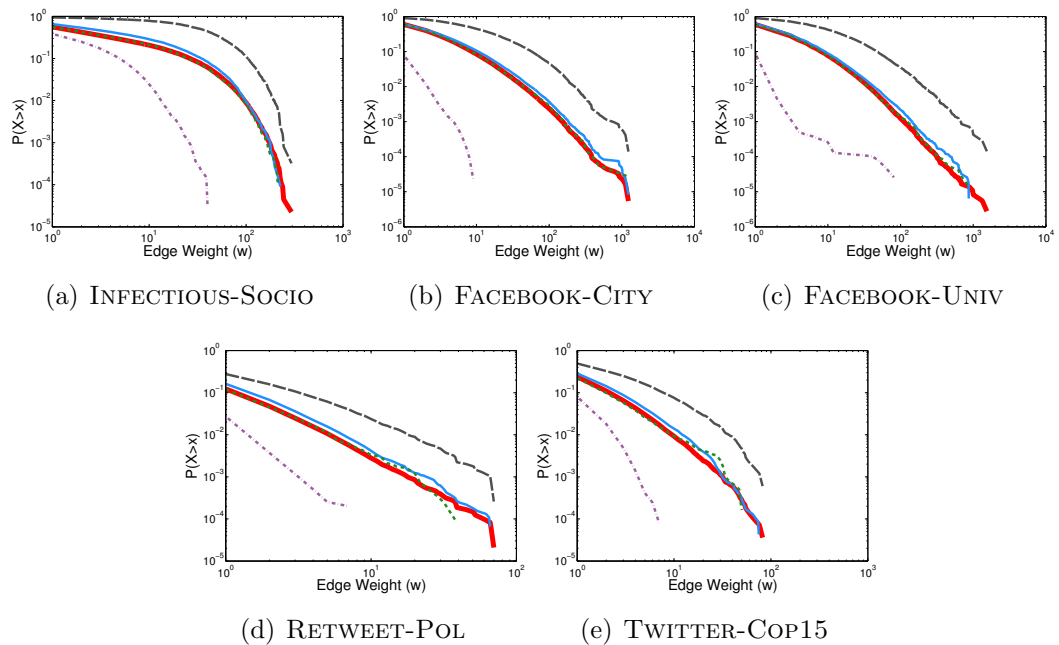


Fig. 4.15.: Distributions of edge weight in multigraph streams at 20% sample.

and its degree (e. g., $\text{corr}(k, s) = 0.76$ in FACEBOOK-CITY). Due to this correlation, sampling methods that are biased to high degree nodes also have a greater chance of capturing node strength compared to methods that are biased to low degree nodes. On the other hand, the bias towards high degree nodes also may result in over-estimation of the edge weights. This is clearly the case for ES. Similar observations can be seen in the other graph data sets as shown in Figures 4.14 and 4.15. We also note that these results are consistent at different points in the stream.

4.7.3 Interplay between Graph Dynamics and Structure

We further investigate the interplay between graph dynamics and structure by plotting node degree versus node strength, comparing the samples to the original data. Figure 4.16 shows scatter plots for a variety of multigraph streams (for a 20% sample from the total stream).

The results show that PIES outperforms all other methods when it comes to capturing the correlation between node degree and strength. NS performs similar to PIES but generally fails to capture high degree nodes. Clearly, ES is biased to sampling frequent edges with incident nodes that may or may not have a high degree. Therefore, we observe ES that captures the strength of the nodes but fails to capture their degree. In contrast, BFS captures the degree but fails to capture the strength of the nodes.

From this experiment, we conclude that the choice of stream sampling method may have a significant impact on the accuracy of sampling multigraph streams. This is because the *probability* of sampling nodes depends not only on the graph structure (i. e., node degree) but also on graph dynamics (i. e., node strength and edge weight). Therefore, methods that are biased to sampling frequent edges (e. g., ES) will capture more of the dynamics but less of the structure. At the same time, methods that sample nodes (nearly) uniformly (e. g., NS, PIES) are able to capture both the graph structure and its dynamics. We also found that graphs that are more clustered (e. g.,

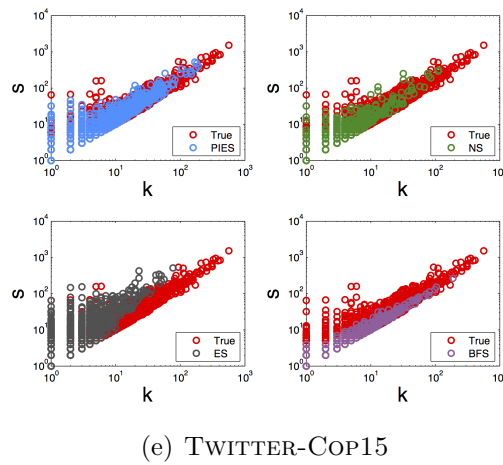
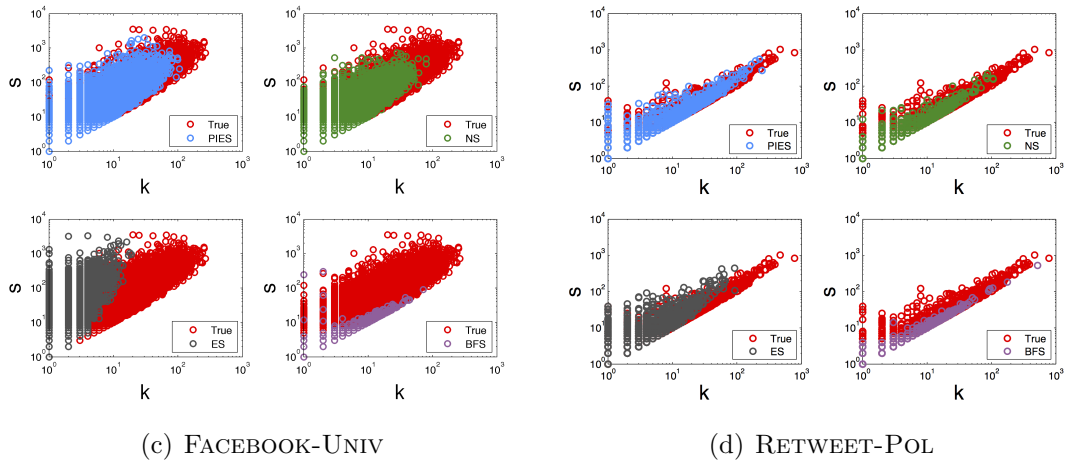
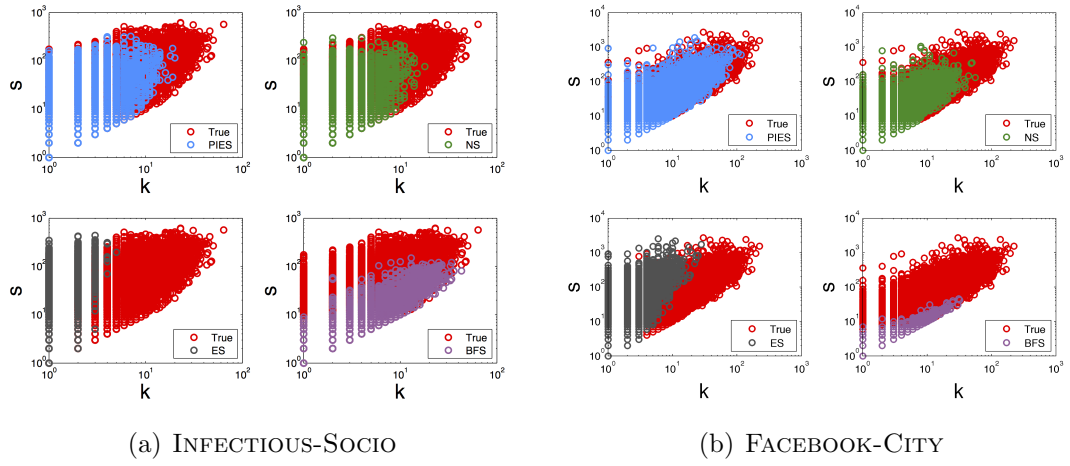


Fig. 4.16.: Node strength (s) vs. degree (k) at 20% sample of multigraph streams.

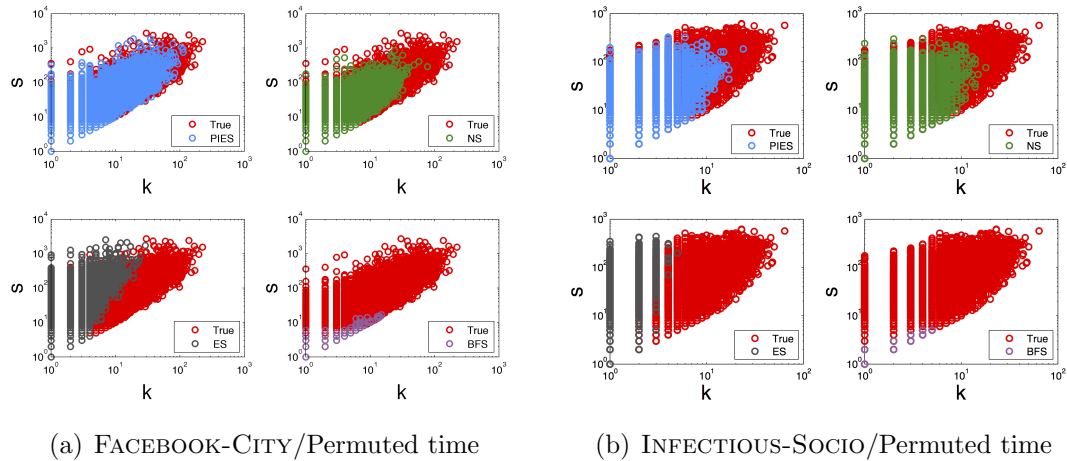


Fig. 4.17.: Node strength (s) versus degree (k) for PIES, NS, ES, and BFS at 20% sample size for graph streams with permuted ordering.

INFECTIOUS-SOCIO) are generally more difficult to sample accurately compared to less clustered graphs (e. g., TWITTER-COP15).

4.7.4 Randomization Tests for Graph Streams

Randomization tests provide a way to test the effects of time on the graph structure and degree/strength correlation. To investigate the effects of time-clustering on the various sampling methods, we permuted the timestamps in the original stream before sampling. If node interactions are grouped together in small intervals of time, then the permutation will destroy this aspect of the stream. Figure 4.17 shows plots of node degree versus node strength for FACEBOOK-CITY and INFECTIOUS-SOCIO when sampled from time-permuted graphs. The results show no impact on the accuracy of PIES, NS, and ES. However, BFS is significantly impacted by the time permutation, which shows that BFS performance is highly dependent on the temporal clustering of edges in the stream. We also found that the effect of time clustering is more significant in INFECTIOUS-SOCIO and less significant in the Twitter communication graphs we studied.

4.8 Summary

In this chapter, we proposed a novel single-pass streaming network sampling algorithm (PIES), and we extended traditional algorithms from the three classes (node, edge, and topology-based sampling). Our contributions can be summarized in the following points:

- (1) PIES runs in a single pass scan over the stream, and maintains only a stored state of the order of the required sample size
- (2) Sampled subgraphs constructed by PIES accurately preserve many network statistics and distributions (e. g., degree, path length, and k-core).
- (3) PIES produces better samples when the graph is more sparse and less clustered (e. g., TWITTER and LIVEJOURNAL datasets).
- (4) PIES can be easily adapted to reduce the number of isolated nodes in the sample by modifying the reservoir replacement mechanism (i. e., PIES(MIN)).
- (5) PIES(MIN) more accurately preserves the properties of dense graphs as well as certain statistics (e. g., eigenvalues) that are difficult to be capture with PIES.
- (6) PIES is better able to capture both the dynamics and structure of multigraph streams compared to other methods.
- (7) The results show that the structure of the sampled subgraph G_s depends on the manner in which the topology of the graph G , the nature of the target property η (e. g., degree distribution), and the characteristics of the sampling method interact.

5. SAMPLE & HOLD: A FRAMEWORK FOR BIG GRAPH ANALYTICS

Sampling is a standard approach in big-graph analytics; the goal is to efficiently estimate the graph properties by consulting a sample of the whole population. A perfect sample is assumed to mirror every property of the whole population. Unfortunately, such a perfect sample is hard to collect in complex populations such as graphs (e.g. web graphs, social networks), where an underlying network connects the units of the population. Therefore, a good sample will be representative in the sense that graph properties of interest can be estimated with a known degree of accuracy.

While previous work focused particularly on sampling schemes to estimate certain graph properties (e.g. triangle count), much less is known for the case when we need to estimate various graph properties with the same sampling scheme. In this chapter, we propose a generic stream sampling framework for big-graph analytics, called Graph Sample and Hold (gSH), which samples from massive graphs sequentially in a single pass, one edge at a time, while maintaining a small state in memory.

5.1 Motivation

One key stumbling block for enabling big graph analytics is the limitation in computational resources. Despite advances in distributed and parallel processing frameworks such as MapReduce for graph analytics and the appearance of infinite resources in the cloud, running brute-force graph analytics is either too costly, too slow, or too inefficient in many practical situations. Further, finding an *approximate* answer is usually sufficient for many types of analyses; the extra cost and time in finding the exact answer is often not worth the extra accuracy. *Sampling* therefore

provides an attractive approach to quickly and efficiently finding an approximate answer to a query, or more generally, any analysis objective.

In this chapter, we propose a new sampling framework for big-graph analytics, called Graph Sample and Hold (gSH). gSH essentially maintains a small amount of state and passes through all edges in the graph in a streaming fashion. The sampling probability of an arriving edge can in general be a function of the stored state, such as the adjacency properties of the arriving edge with those already sampled. (This can be seen as an analog of the manner in which standard Sample and Hold [148] samples packets with a probability depending on whether their key matches one already sampled). Since the algorithm involves processing only a sample of edges (and thus, nodes), it keeps run time complexity under check.

gSH provides a generic framework for unbiased estimation of the counts of arbitrary subgraphs. This uses the Horvitz-Thompson construction [45] in which the count of any sampled object is weighted by dividing by its sampling probability. In gSH this is realized by maintaining along with each sampled edge, the sampling probability that was in force when it was sampled. The counts of subgraphs of sampled edges are then weighted according to the product of the selection probabilities of their constituent edges. Since the edge sampling probabilities are determined conditionally with respect to the prior sampling outcomes, this product reflects the dependence structure of edge selection.

The sampling framework also provide the means to compute the accuracy of estimates, since the unbiased estimator of the variance of the count estimator can be computed from the sampling probabilities of selected edges alone. More generally, the covariance between the count estimators of any pair of subgraphs can be estimated in the same manner.

The framework itself is quite generic. By varying the dependence of sampling probabilities on previous history, one can tune the estimation of various properties of the original graph efficiently with arbitrary degrees of accuracy. For example, simple uniform sampling of edges at random may naturally lead to selecting a large number

of higher-degree nodes since higher-degree nodes appear in more number of edges. For each of these sampled nodes, we can choose the holding function to simply track the size of the degree for these specific nodes, of course accounting for the loss of the count before the node has been sampled in an unbiased manner. Similarly, by carefully designing the sampling function, we can obtain a uniformly random sample of nodes (similar to the classic node sampling), for whom we can choose to hold an accurate count of number of triangles each of these nodes is part of.

In this chapter, we demonstrate applications of the gSH framework in two directions. Firstly, we formulate a parameterized family $\text{gSH}(p,q)$ of gSH sampling schemes, in which an arriving edge with no adjacencies with previously sampled edges is selected with probability p ; otherwise it is sampled with probability q . Secondly, we consider four specific quantities of interest to estimate within the framework. These are counts of links, triangles, connected paths of length two, and the derived global clustering coefficient. We also provide an unbiased estimator of node counts based on edge sampling. Note that we do not claim that these lists of examples are by any means exhaustive or that the framework can accommodate arbitrary queries efficiently.

In Section 5.3, we describe the general framework for graph sampling, and show how it can be used to provide unbiased estimates of the counts of arbitrary selections of subgraphs. We also show how unbiased estimates of the variance of these estimators can be efficiently computed within the same framework. In Section 5.4, we show how counts of specific types of subgraph (links, triangles, paths of length 2) and the global clustering coefficient can be estimated in this framework. In Section 5.5, we describe the specific $\text{gSH}(p,q)$ graph Sample and Hold algorithms, and illustrate the application of $\text{gSH}(p,1)$ on a simple graph. In Section 5.6, we describe a set of evaluations based on a number of real network topologies. We apply the estimators described in Section 5.3.2 to the counts described in Section 5.4, and compare empirical confidence intervals with those estimated directly from the samples. We also

compare accuracy with prior work. We discuss the general relation of our work to existing literature in Section 5.7 and conclude in Section 5.8.

5.2 Relation to Classic Sample and Hold

Sample and hold for big-graph analytics bears some resemblance to the classic Sample and Hold (SH) approach [148], versions of which also appeared as counting samples by [149], and were used for attack detection by [150]. In SH, packets carry a key that identifies the flow to which they belong. A router maintains a cache of information concerning the flows of packets that traverse it. If the key of an arriving packet matches a key on which information is currently maintained in the router, the information for that key (such as packet and byte counts and timing information) is updated accordingly. Otherwise the packet is sampled with some probability p . If selected, a new entry is instantiated in the cache for that key. SH is more likely to sample longer flows. Thus, SH provides an efficient way to store information concerning the disposition of packet across the small proportion of flows that carry a large proportion of all network packets.

gSH can be viewed as an analog of SH in which the equivalence relation of packets according to their keys is replaced by adjacency relation between links. But this generalization brings many differences as well. In particular, many graph properties involve transitive properties (e.g., triangles) that are relatively uninteresting in network measurements (and hence, under explored). For many of these properties, it is important to realize that the accuracy of the analytics depends on the ordering of edges to some extent, which was not the case for the vast majority of network measurement problems considered in the literature.

5.3 Framework for Graph Sampling

5.3.1 Graph Stream Model

Let $G = (V, K)$ be a graph. We call two edges $k, k' \in K$ adjacent, $k \sim k'$, if they join at some node. Specifically:

- *Directed adjacency:* $k = (k_1, k_2) \sim k' = (k'_1, k'_2)$ iff $k_2 = k'_1$ or $k_1 = k'_2$. Note that \sim is not symmetric in this case.
- *Undirected adjacency:* $k = (k_1, k_2) \sim k' = (k'_1, k'_2)$ iff $k \cap k' \neq \emptyset$. Note that \sim is symmetric in this case.

Without loss of generality we assume edges are unique; otherwise distinguishing labels that are ignored by \sim can be appended.

The edges in K are arriving in an order $k : [|K|] \rightarrow K$. For $k, k' \in K$, we write $k \prec k'$ if k appears earlier than k' in arrival order. For $i \leq |K|$, $K_i = \{k \in K : k \preceq k_i\}$ comprises the first i arrivals.

5.3.2 Edge Sampling Model

We describe the sampling of edges through a random process $\{H_i\} = \{H_i : i \in [|K|]\}$ where $H_i = 1$ if k_i is selected, and $H_i = 0$ otherwise. Let \mathcal{F}_i denote the set of possible outcomes $\{H_1, \dots, H_i\}$; We assume that an edge is selected according to a probability that is a function of the sampling outcomes of previous edges. For example, the selection probability of an edge can be a function of the (random) number of previously selected edges that are adjacent to it. Thus we write

$$\mathbb{P}[k_i \text{ is selected} \mid \{H_1, \dots, H_{i-1}\}] = \mathbb{E}[H_i \mid \mathcal{F}_{i-1}] = p_i \quad (5.1)$$

where $p_i \in (0, 1]$ is the *random* probability that is determined by the first $i - 1$ sampling outcomes¹.

5.3.3 Subgraph Estimation

In this chapter, we are principally concerned with estimating the frequency of occurrence of certain subsets of K within the sample. Our principal tool is the **selection estimator** $\widehat{S}_i = H_i/p_i$ of the link k_i , which indicates the presence of k_i in K . It is uniquely defined by the properties: (i) $\widehat{S}_i \geq 0$; (ii) $\widehat{S}_i > 0$ iff $H_i > 0$; and (iii) $\mathbb{E}[\widehat{S}_i | \mathcal{F}_{i-1}] = 1$, which we prove in Theorem 5.3.1 below. We recognize \widehat{S}_i as a Horvitz-Thompson estimator [45] of unity.

The idea generalizes to indicators of general subsets of edges with K . We call a subset $J \subset K$ an ordered subset when written in the increasing arrival order $J = (j_{i_1}, j_{i_2}, \dots, j_{i_m})$ with $i_1 < i_2 < \dots < i_m$. For an ordered subset J of K we write

$$H(J) = \prod_{j_i \in J} H_i \quad \text{and} \quad P(J) = \prod_{j_i \in J} p_i \quad (5.2)$$

with the convention that $H(\emptyset) = P(\emptyset) = 1$. We say that J is selected if $H(J) = 1$. The selection estimator for an ordered subset J of K is

$$\widehat{S}(J) = \prod_{j_i \in J} \widehat{S}_{j_i} = H(J)/P(J) \quad (5.3)$$

which is our main structural result concerning the properties of $\widehat{S}(J)$.

Theorem 5.3.1 (i) $\mathbb{E}[\widehat{S}_i | \mathcal{F}_{i-1}] = 1$ and hence $\mathbb{E}[\widehat{S}_i] = 1$.

(ii) For any ordered subset $J = (j_{i_1}, \dots, j_{i_m})$ of K ,

$$\mathbb{E}[\widehat{S}(j_{i_1}, \dots, j_{i_m}) | \mathcal{F}_{i_m-1}] = \widehat{S}(j_{i_1}, \dots, j_{i_m-1}) \quad (5.4)$$

¹Formally, $\{\mathcal{F}_i\}$ is the natural filtration associated with the process $\{H_i\}$, and $\{p_i\}$ is previsible w.r.t. $\{\mathcal{F}_i\}$; see [151].

and hence

$$\mathbb{E}[\widehat{S}(J)] = 1 \quad (5.5)$$

(iii) Let J, J' be two ordered subsets of K . If $J \cap J' = \emptyset$ then

$$\mathbb{E}[\widehat{S}(J)\widehat{S}(J')] = 1 \text{ and hence } \text{Cov}(\widehat{S}(J), \widehat{S}(J')) = 0 \quad (5.6)$$

(iv) Let J_1, \dots, J_ℓ be disjoint ordered subsets of K . Let q be a polynomial in ℓ variables that is linear in each of its arguments. Then $\mathbb{E}[q(\widehat{S}(J_1), \dots, \widehat{S}(J_\ell))] = q(1, \dots, 1)$.

(v) Let J, J' be two ordered subsets of K with $J \ominus J'$ be their symmetric difference. Then $\widehat{C}(J, J')$ defined below is non-negative and an unbiased estimator of $\text{Cov}(\widehat{S}(J), \widehat{S}(J'))$, which is hence non-negative. $\widehat{C}(J, J')$ is defined to be 0 when $J \cap J' = \emptyset$, and otherwise:

$$\begin{aligned} \widehat{C}(J, J') &= \widehat{S}(J \ominus J')\widehat{S}(J \cap J') \left(\widehat{S}(J \cap J') - 1 \right) \\ &= \widehat{S}(J \cup J') \left(\widehat{S}(J \cap J') - 1 \right) \end{aligned} \quad (5.7)$$

(vi) $\widehat{S}(J) \left(\widehat{S}(J) - 1 \right)$ is an unbiased estimator of $\text{Var}(\widehat{S}(J))$.

Proof (i) $\mathbb{E}[\widehat{S}_i | \mathcal{F}_{i-1}] = \mathbb{E}[H_i/p_i | \mathcal{F}_{i-1}] = 1$, since $p_i > 0$.

(ii) is a corollary of (i) since

$$\mathbb{E}[\widehat{S}(j_{i_1}, \dots, j_{i_m}) | \mathcal{F}_{i_{m-1}}] \quad (5.8)$$

$$\begin{aligned} &= \mathbb{E} \left[\mathbb{E}[\widehat{S}_{i_m} | \mathcal{F}_{i_{m-1}}] \widehat{S}(j_{i_1}, \dots, j_{i_{m-1}}) | \mathcal{F}_{i_{m-1}} \right] \\ &= \widehat{S}(j_{i_1}, \dots, j_{i_{m-1}}) \end{aligned} \quad (5.9)$$

(iii) When $J \cap J' = \emptyset$, then by (ii)

$$\mathbb{E}[\widehat{S}(J)\widehat{S}(J')] = \mathbb{E}[\widehat{S}(J \cap J')] = 1 \quad (5.10)$$

Since J and J' are independent, and due to our convention that $P(\emptyset) = 1$.

(iv) Is a direct corollary of (iii)

(v) We prove the unbiasedness and non-negativity properties of $C(J, J')$.

- Unbiasedness: if $J \cap J' \neq \emptyset$, then by taking the expectation of Eq. 5.7,

$$\begin{aligned} \mathbb{E}[\widehat{C}(J, J')] &= \mathbb{E}[\widehat{S}(J)\widehat{S}(J')] - \mathbb{E}[\widehat{S}(J \cup J')] \\ &= \mathbb{E}[\widehat{S}(J)\widehat{S}(J')] - 1 = \text{Cov}(\widehat{S}(J), \widehat{S}(J')) \end{aligned} \quad (5.11)$$

since $\mathbb{E}[\widehat{S}(J)] = \mathbb{E}[\widehat{S}(J')] = 1$.

Note that the case $J \cap J' = \emptyset$ follows directly from (iii).

- Non-negativity: $\widehat{C}(J, J')$ is a product of non-negative terms.

Specifically, $\widehat{C}(J, J') = [H(J \cup J')/P(J \cup J')][H(J \cap J')/P(J \cap J') - 1] = [H(J \cup J')/P(J \cup J')][1/P(J \cap J') - 1] \geq 0$, since $H(A)H(B) = H(A)$ when $B \subset A$, for any two sets A and B .

(vi) is a special case of (v) with $J = J'$. ■

5.4 Unbiased Estimation

We now describe in more detail the process of estimation, and computing variance estimates. The most general quantity that we wish to estimate is a weighted sum over collections of subgraphs; for brevity, we will refer to these as **subgraph sums**. This class includes quantities such as counts of total nodes or links in G , or counts of more complex objects such as connected paths of length two, or triangles that have been a focus of study in the recent literature. However, the class of more general quantities in which a selector is applied to all subgraphs of a given type (e.g. triangles) or only subgraphs fulfilling a selection criterion (e.g. based on labels on the nodes of the triangle) are to be included in future work.

5.4.1 General Estimation and Variance

To allow for the greatest possible generality, we let $\mathcal{K} = 2^K$ denote the set of subsets of K , and let f be a real function on \mathcal{K} . For any subset $Q \subset \mathcal{K}$, the subset sum of f over Q is

$$f(Q) = \sum_{J \in Q} f(J) \quad (5.12)$$

Here Q represents the set of subgraphs fulfilling a selection criterion as described above. Let \widehat{Q} denote the set of objects in Q that are sampled, i.e., therefore $J = (k_{i_1}, \dots, k_{i_m}) \in Q$ for which all links are selected. The following is an obvious consequence of the linearity of expectation and Theorem 5.3.1

Theorem 5.4.1 (i) *An unbiased estimator of $f(Q)$ is*

$$\widehat{f}(Q) = \sum_{J \in Q} f(J) \widehat{S}(J) = \sum_{J \in \widehat{Q}} f(J) / P(J) \quad (5.13)$$

(ii) *An unbiased estimator of $\text{Var}(\widehat{f}(Q))$ is*

$$\sum_{J, J' \in \widehat{Q}: J \cap J' \neq \emptyset} f(J) f(J') (1/P(J \cup J')) (1/P(J \cap J') - 1) \quad (5.14)$$

Proof (i) In Theorem 5.3.1, we showed that $\mathbb{E}[\widehat{S}(J)] = 1$. Thus, as a direct consequence of Theorem 5.3.1 and the linearity of expectation, we show that

$$\mathbb{E}[\widehat{f}(Q)] = f(Q) \quad (5.15)$$

(ii) In Theorem 5.3.1, we showed that $\widehat{C}(J, J')$ is non-negative and an unbiased estimator of $\text{Cov}(\widehat{S}(J), \widehat{S}(J'))$. Thus, (ii) is a direct consequence of Theorem 5.3.1, and the variance properties for the sum of correlated variables. ■

Note that the sum in (5.14) can formally be left unrestricted since terms with non-intersecting J, J' are zero due to our convention that $P(\emptyset) = 1$.

5.4.2 Edges

As before K denotes the edges in G ; let \widehat{K} denote the set of sampled edges. Then

$$\widehat{N}_K = \sum_{k_i \in \widehat{K}} \frac{1}{p_i} \quad (5.16)$$

is an unbiased estimate of the unique edge count $N_K = |K|$. An unbiased estimate of the variance of \widehat{N}_K is

$$\text{Var}(\widehat{N}_K) = \sum_{k_i \in \widehat{K}} \frac{1}{p_i} \left(\frac{1}{p_i} - 1 \right) \quad (5.17)$$

5.4.3 Triangles

Let T denote the set of triangles $\tau = (k_1, k_2, k_3)$ in G , and \widehat{T} the set of sampled triangles. Then

$$\widehat{N}_T = \sum_{\tau \in \widehat{T}} 1/P(\tau) \quad (5.18)$$

is an unbiased estimate of $N_T = |T|$, the number of triangles in G . Since two intersecting triangles have either one link in common or are identical, an unbiased estimate of $\text{Var}(\widehat{N}_T)$ is

$$\text{Var}(\widehat{N}_T) = \sum_{\tau \in \widehat{T}} \frac{1}{P(\tau)} \left(\frac{1}{P(\tau)} - 1 \right) + \sum_{\tau \neq \tau' \in \widehat{T}} \frac{1}{P(\tau \cup \tau')} \left(\frac{1}{P(e(\tau, \tau'))} - 1 \right)$$

where $e(\tau, \tau')$ is the common edge between τ and τ'

5.4.4 Connected Paths of Length 2

Let Λ denote the set of connected paths of length two $L = (k_1, k_2)$ in G , and $\hat{\Lambda}$ the subset of these that are sampled. Then

$$\hat{N}_\Lambda = \sum_{L \in \hat{\Lambda}} 1/P(L) \quad (5.19)$$

is an unbiased estimate of $N_\Lambda = |\Lambda|$, the number of such paths in G . Since two non-identical members of Λ may have one edge in common, an unbiased estimate of $\text{Var}(\hat{N}_\Lambda)$ is

$$\text{Var}(\hat{N}_\Lambda) = \sum_{L \in \hat{\Lambda}} \frac{1}{P(L)} \left(\frac{1}{P(L)} - 1 \right) + \sum_{L \neq L' \in \hat{\Lambda}} \frac{1}{P(L \cup L')} \left(\frac{1}{P(e(L, L'))} - 1 \right)$$

where $e(L, L') = L \cap L'$ is the common edge between L and L' .

5.4.5 Clustering Coefficient

The global clustering coefficient of a graph is defined as $\alpha = 3N_T/N_\Lambda$. While we use $\hat{\alpha} = 3\hat{N}_T/\hat{N}_\Lambda$ as an estimator of α , it is not unbiased. However, the well known delta-method [152] suggests using a formal Taylor expansion. But we note that a rigorous application of this method depends on establishing asymptotic properties of \hat{N}_T and \hat{N}_Λ for large graphs, the study of which we defer to future work. With this caveat we proceed as follows. For a random vector $X = (X_1, \dots, X_n)$ a second order Taylor expansion results in the approximation

$$\text{Var}(f(X_1, \dots, X_n)) \approx v \cdot Mv \quad (5.20)$$

where $v = (\nabla f)(\mathbb{E}[X])$ and M is the covariance matrix of the X_i . Considering $f(\widehat{N}_T, \widehat{N}_\Lambda) = \widehat{N}_T/\widehat{N}_\Lambda$ we obtain the following approximation. For computation we replace all quantities by their corresponding unbiased estimators derived previously:

$$\text{Var}(\widehat{N}_T/\widehat{N}_\Lambda) \approx \frac{\text{Var}(\widehat{N}_T)}{\widehat{N}_\Lambda^2} + \frac{\widehat{N}_T^2 \text{Var}(\widehat{N}_\Lambda)}{\widehat{N}_\Lambda^4} - 2 \frac{\widehat{N}_T \text{Cov}(\widehat{N}_T, \widehat{N}_\Lambda)}{\widehat{N}_\Lambda^3} \quad (5.21)$$

Following Theorem 5.3.1, the covariance term is estimated as

$$\sum_{\substack{\tau \in \widehat{T}, L \in \widehat{\Lambda} \\ \tau \cap L \neq \emptyset}} \frac{1}{P(\tau \cup L)} \left(\frac{1}{P(\tau \cap L)} - 1 \right) \quad (5.22)$$

5.4.6 Nodes

Node selection is not directly expressed as a subgraph sum, but rather through a polynomial of the type treated in Theorem 5.3.1(iv). Let $K(x)$ denote the edges containing the node $x \in V$. Now observe x remains unsampled if and only if no edge in $K(x)$ is sampled. This motivates the following estimator of node selection:

$$\widehat{n}_x = 1 - \prod_{k_i \in K(x)} (1 - \widehat{S}_i) \quad (5.23)$$

The following is a direct consequence of Theorem 5.3.1(iv)

Lemma 5.4.1 $\widehat{n}_x = 0$ if and only if no edge from $K(x)$ is sampled, and $\mathbb{E}[n_x] = 1$.

5.5 Graph Sample and Hold

5.5.1 Algorithms

We now turn to specific sampling algorithms that conform to the edge sampling model of Section 5.3.2. **Graph Sample and Hold** $\text{gSH}(p, q)$ is a single pass al-

gorithm over a stream of edges. The edge k is somewhat analogous to the key of (standard) sample and hold. A matching edge is sampled with probability q . If there is not a match, the edge is stored with some probability p . An edge not sampled is discarded permanently. For estimation purposes we also need to keep track of the probability with which a selected edge is sampled. We formally specify $\text{gSH}(p, q)$ as Algorithm 5.1.

Algorithm 5.1: Graph Sample and Hold: $\text{gSH}(p, q)$

```

1  $\widehat{K} \leftarrow \emptyset;$ 
2 while new edge  $k$  do
3   if  $k \sim k'$  for some  $(k', p') \in \widehat{K}$  then
4      $r = q$ 
5   else
6      $r = p$ 
7   Append  $(k, r)$  to  $\widehat{K}$  with probability  $r$ 

```

In some sense, gSH samples connected components in the same way the standard sample and hold samples flows, although there are some differences. The main difference is a single connected component in the original graph may be sampled as multiple components. This can happen, for example, if omission of an edge from the sample can disconnect a component. Clearly the order in which nodes are streamed determines whether or not such sampling disconnection can occur.

Algorithm 5.2: Graph Sample and Hold for Triangles: $\text{gSH}_T(p, q)$

```

1  $\widehat{K} \leftarrow \emptyset;$ 
2 while new edge  $k$  do
3   if  $k$  would complete a triangle in  $\widehat{K}$  then
4      $r = 1$ 
5   else
6     if  $k \sim k'$  for some  $(k', p') \in \widehat{K}$  then
7        $r = q$ 
8     else
9        $r = p$ 
10  Append  $(k, r)$  to  $\widehat{K}$  with probability  $r$ 

```

Clearly, gSH would admit generalizations that allow a more complex dependence of the sampling probability for a new edge on the current sampled edge set. This can

Table 5.1.: Estimation on a path of length 3 using $\text{gSH}(p, 1)$

Order			Selection			Probability	Weights			Est. Node Degree				
(a,b)	(b,c)	(c,d)	(a,b)	(b,c)	(c,d)		(a,b)	(b,c)	(c,d)	a	b	c	d	
1	2	3	✓	✓	✓	p	$1/p$	1	1	$1/p$	$1/p+1$	2	1	
			·	✓	✓	$(1-p)p$	0	$1/p$	1	0	$1/p$	$1/p+1$	1	1
			·	·	✓	$(1-p)^2p$	0	0	$1/p$	0	0	0	$1/p$	$1/p$
			·	·	·	$(1-p)^3$	0	0	0	0	0	0	0	0
2	1	3	✓	✓	✓	p	1	$1/p$	1	1	$1/p+1$	$1/p+1$	1	
			✓	·	✓	$(1-p)p^2$	$1/p$	0	$1/p$	$1/p$	$1/p$	$1/p$	$1/p$	$1/p$
			·	·	✓	$(1-p)^2p$	0	0	$1/p$	0	0	$1/p$	$1/p$	$1/p$
			✓	·	·	$(1-p)^2p$	$1/p$	0	0	$1/p$	$1/p$	0	0	0
·	·	·	$(1-p)^3$	0	0	0	0	0	0	0	0	0		
1	3	2	✓	✓	✓	p^2	$1/p$	1	$1/p$	$1/p$	$1/p+1$	$1/p+1$	$1/p$	
			✓	✓	·	$p(1-p)$	$1/p$	1	0	$1/p$	$1/p+1$	1	0	
			·	✓	✓	$(1-p)p$	0	1	$1/p$	0	1	$1/p+1$	1	
			·	✓	·	$(1-p)^2p$	0	$1/p$	0	0	$1/p$	$1/p$	0	
·	·	·	$(1-p)^3$	0	0	0	0	0	0	0	0			

be achieved by adapting the flexible holding function. Consequently, the details of the sampling scheme (holding function) should allow certain subgraphs to be favored for selection. In this chapter, we do not delve into this matter in great detail, rather we look at a simple illustrative modification of gSH that favor the selection of triangles—called gSH_T . gSH_T is identical to gSH except that any arriving edge that would complete a triangle is selected with probability 1; see Algorithm 5.2. Obviously $\text{gSH}(p, 1)$ and $\text{gSH}_T(p, 1)$ are identical.

5.5.2 Illustration with $\text{gSH}(p,1)$

We use a simple example of a path of length 3 to illustrate that in Graph Sample and Hold $\text{gSH}(p, 1)$, the distribution of the random graph sample depends on the order in which the edges are presented. The graph $G = (V, K)$ comprises 4 nodes $V = a, b, c, d$ connected by 3 undirected edges $K = \{(a, b), (b, c), (c, d)\}$ which are the keys for our setting.

There are 6 possible arrival orders for the keys, of which we need only analyze 3, since the other orders can be obtained by time reversal. These are displayed in the “Order” columns in Table 5.1. For each order, the possible selection outcomes for the

Table 5.2.: Statistics of datasets. n is the number of nodes, N_K is the number of edges, N_T is the number of triangles, N_Λ is the number of connected paths of length 2, α is the global clustering coefficient, and D is the density.

graph	n	N_K	N_T	N_Λ	α	D
socfb-CMU	7K	249.9K	2.3M	37.4M	0.18526	0.0114
socfb-UCLA	20K	747.6K	5.1M	107.1M	0.14314	0.0036
socfb-Wisconsin	24K	835.9K	4.8M	121.4M	0.12013	0.0029
web-Stanford	282K	1.9M	11.3M	3.9T	0.00862	5.01×10^{-5}
web-Google	876K	4.3M	13.3M	727.4M	0.05523	1.15×10^{-5}
web-BerkStan	685K	6.6M	64.6M	27.9T	0.00694	2.83×10^{-5}

three edges by the check marks \checkmark , followed by the probability of each selection. The estimated weights for each outcome is displayed in “Weights” followed by corresponding estimate of the node degree, i.e., the sum of estimated weights of edges incident at each node. One can check by inspection that the probability-weighted sums of the weight estimators are 1, while the corresponding sums of the degree estimators yield the the true node degree.

5.6 Experiments

We test the performance of our proposed framework (gSH $_T$) as described in Algorithm 5.2 (with $r = 1$ for edges that are closing triangles), on various social and information networks with 250K–7M edges. For all network datasets, we consider an undirected graph, discard edge weights, self-loops, and we generate the stream by randomly permuting the edges. Table 5.2 summarizes the main characteristics of these graphs, such that n is the number of nodes, N_K is the number of edges, N_T is the number of triangles, N_Λ is the number of connected paths of length two, α is the global clustering coefficient, and D is the graph density.

1. **Social Facebook Graphs.** Here, the nodes are people and edges represent friendships among Facebook users in three different US schools (CMU, UCLA, and Wisconsin, see [153] for data analysis and downloads).

2. **Web Graphs**². Here, the nodes are web-pages and edges are hyperlinks among these pages in different domains.

We conduct the experiments on MacbookPro 2.66GHZ 6-Core Intel processor, with 48GB memory. In order to test the effect of parameter settings (i.e., p and q), we perform 100 independent experiments and we consider all possible combinations of p and q in the range $p, q = \{0.005, 0.008, 0.01, 0.03, 0.05, 0.08, 0.1\}$. Our experimental procedure is done independently for each $p = p_i, q = q_i$ as follows:

1. Given one parameter setting $p = p_i, q = q_i$, we obtain a sample of edges $\hat{K} \subset K$ using $\text{gSH}_T(p_i, q_i)$ —as described in Algorithm 5.2.
2. Using \hat{K} , compute the unbiased estimates of the following statistics: Edge counts \hat{N}_K ; Triangle counts \hat{N}_T ; Connected paths of length two \hat{N}_Λ ; Global Clustering Coefficient $\hat{\alpha}$.
3. Compute the unbiased estimates of the variance of the quantities mentioned above.

Note that the estimation of the count of unique edges \hat{N}_K is necessary when the graph stream is not simple (i.e., edges may occur more than once).

5.6.1 Performance Analysis

We proceed by first demonstrating the accuracy of the proposed estimators for the different graph statistics we discuss in this chapter across various social and web networks. Given a sample $\hat{K} \subset K$ (collected by gSH_T Algorithm 5.2), we consider the absolute relative error (i.e., $\frac{|E[\text{est}] - \text{Actual}|}{\text{Actual}}$) as a measure of how far is the estimated statistic from the actual graph statistic of interest, where $E[\text{est}]$ is the mean estimated value across 100 independent runs. Table 5.3 provides the estimated values in comparison to the actual statistics when the sample size is $\leq 40\text{K}$ with $p, q = 0.005$ for web-BerkStan and $p = 0.005, q = 0.008$ otherwise.

²Stanford Network Project, <http://snap.stanford.edu/>

We summarize below our main findings from Table 5.3:

- For edge count (N_K) estimates, we observe that the relative error is in the range of 0.03% – 0.5% across all graphs.
- For triangle count (N_T) estimates, we observe that the relative error is in the range of 0.03% – 0.95% across all graphs.
- For the number of connected paths of length two (N_Λ), we observe that the relative error of the estimates is in the range of 0.02% – 0.6% across all graphs.
- For clustering coefficient (α) estimates, we observe that the relative error is in the range of 0.02% – 0.76% across all graphs.
- Finally, we observe that the highest error is in the triangle count estimates and yet it is still $\leq 1\%$.

5.6.2 Confidence Bounds

Having selected a sample that can be used to estimate the actual statistic, it is also desirable to construct a confidence interval within which we are *sufficiently* sure that the actual graph statistic of interest lies. We construct a 95% confidence interval for the estimates of edge (N_K), triangle (N_T), connected paths of length two (N_Λ) counts, and clustering coefficient (α) as follows,

$$\text{est} \pm 1.96\sqrt{\text{Var}(\text{est})} \quad (5.24)$$

where the estimates ‘est’ and ‘Var(est)’ are computed using the equations of the unbiased estimators of counts and their variance as discussed in Section 5.4. For example, the 95% confidence interval for the edge count is,

$$\widehat{N}_K \pm 1.96\sqrt{\text{Var}(\widehat{N}_K)} \quad (5.25)$$

Table 5.3.: Estimated Properties using Graph Sample & Hold. Estimates of expected value and relative error, when sample size $\leq 40K$ edges, with sampling probability $p, q = 0.005$ for web-BerkStan, and $p = 0.005, q = 0.008$ otherwise. First column shows the statistics of the full graph, SSize is the number of sampled edges, and LB/UB are the 95% lower and upper bounds respectively.

Edges N_K						
	N_K	\hat{N}_K	$\frac{ \hat{N}_K - N_K }{N_K}$	SSize	LB	UB
socfb-CMU	249.9K	249.6K	0.0013	1.7K	236.8K	262.4K
socfb-UCLA	747.6K	751.3K	0.0050	5K	729.3K	773.34K
socfb-Wisconsin	835.9K	835.7K	0.0003	5.5K	812.2K	859.1K
web-Stanford	1.9M	1.9M	0.0004	14.8K	1.9M	2M
web-Google	4.3M	4.3M	0.0007	25.2K	4.2M	4.3M
web-BerkStan	6.6M	6.6M	0.0006	39.8K	6.5M	6.7M
Triangles N_T						
	N_T	\hat{N}_T	$\frac{ \hat{N}_T - N_T }{N_T}$	SSize	LB	UB
socfb-CMU	2.3M	2.3M	0.0003	1.7K	1.6M	2.9M
socfb-UCLA	5.1M	5.1M	0.0095	5K	4.2M	6.03M
socfb-Wisconsin	4.8M	4.8M	0.0058	5.5K	4M	5.7M
web-Stanford	11.3M	11.3M	0.0023	14.8K	3.7M	18.8M
web-Google	13.3M	13.4M	0.0029	25.2K	11.7M	15M
web-BerkStan	64.6M	65M	0.0063	39.8K	45.5M	84.6M
Path. Length two N_Λ						
	N_Λ	\hat{N}_Λ	$\frac{ \hat{N}_\Lambda - N_\Lambda }{N_\Lambda}$	SSize	LB	UB
socfb-CMU	37.4M	37.3M	0.0018	1.7K	32.6M	42M
socfb-UCLA	107.1M	107.8M	0.0060	5K	100.1M	115.42M
socfb-Wisconsin	121.4M	121.2M	0.0018	5.5K	108.9M	133.4M
web-Stanford	3.9T	3.9T	0.0004	14.8K	3.6T	4.2T
web-Google	727.4M	724.3M	0.0042	25.2K	677.1M	771.5M
web-BerkStan	27.9T	27.9T	0.0002	39.8K	26.5T	29.3T
Global Clustering α						
	α	$\hat{\alpha}$	$\frac{ \hat{\alpha} - \alpha }{\alpha}$	SSize	LB	UB
socfb-CMU	0.18526	0.18574	0.00260	1.7K	0.14576	0.22572
socfb-UCLA	0.14314	0.14363	0.00340	5K	0.12239	0.16487
socfb-Wisconsin	0.12013	0.12101	0.00730	5.5K	0.10125	0.14077
web-Stanford	0.00862	0.00862	0.00020	14.8K	0.00257	0.01467
web-Google	0.05523	0.05565	0.00760	25.2K	0.04825	0.06305
web-BerkStan	0.00694	0.00698	0.00680	39.8K	0.00496	0.00900

where $UB = \hat{N}_K + 1.96\sqrt{\text{Var}(\hat{N}_K)}$, $LB = \hat{N}_K - 1.96\sqrt{\text{Var}(\hat{N}_K)}$ are the upper and lower bounds for the edge count respectively.

Table 5.3 provides the 95% upper and lower bounds (i.e., UB, LB) for the sample when the sample size is $\leq 40K$ edges. We observe that the actual statistics across all different graphs lie in between the bounds of the confidence interval (i.e., $LB \leq \text{Actual} \leq UB$). Note that the sample is collected using gSH_T Algorithm 5.2.

Additionally, we study the properties of the sampling distribution of our proposed framework (gSH) as we change the sample size. Figure 5.1 shows the sampling distribution as we increase the sample size (for all possible settings of p, q in the range 0.005–0.1 as described previously). More specifically, we plot the fraction $\frac{E[\text{est}]}{\text{Actual}}$ (represented by blue diamond symbols in the figure), where $E[\text{est}]$ is the mean estimated value across 100 independent runs. Further, we plot the fractions $\frac{\text{UB}}{\text{Actual}}$, and $\frac{\text{LB}}{\text{Actual}}$ (represented by green circle symbols in the figure). These plots show the sampling distribution of all statistics for socfb-UCLA, and socfb-Wisconsin graphs.

We now summarize the findings that we observe from Figure 5.1:

- The sampling distribution is centered and balanced over the red line ($y_{\text{axis}} = 1$) which represents the actual value of the graph statistic. This shows the unbiased properties of the estimators for the four graph quantities of interest that we discussed in Section 5.3.
- The upper and lower bounds contain the actual value (represented by the red line) for different combinations of p, q .
- As we increase the sample size, the bounds *converge* to be more concentrated over the actual value of the graph statistic (i.e, the estimated variance is decreasing as we increase the sample size).
- The confidence intervals for edge counts are small in the range of 0.98–1.02.
- The confidence intervals for triangle counts and clustering coefficient are larger compared to other graph statistics (in the range of 0.87–1.12).
- Samples with size = 40K edges (dashed vertical line) provide a reasonable trade-off between sample size and unbiased estimates with low variance.
- Thus, we conclude that the sampling distribution of the proposed framework has many desirable properties of unbiasedness and low variance as we increase the sample size.

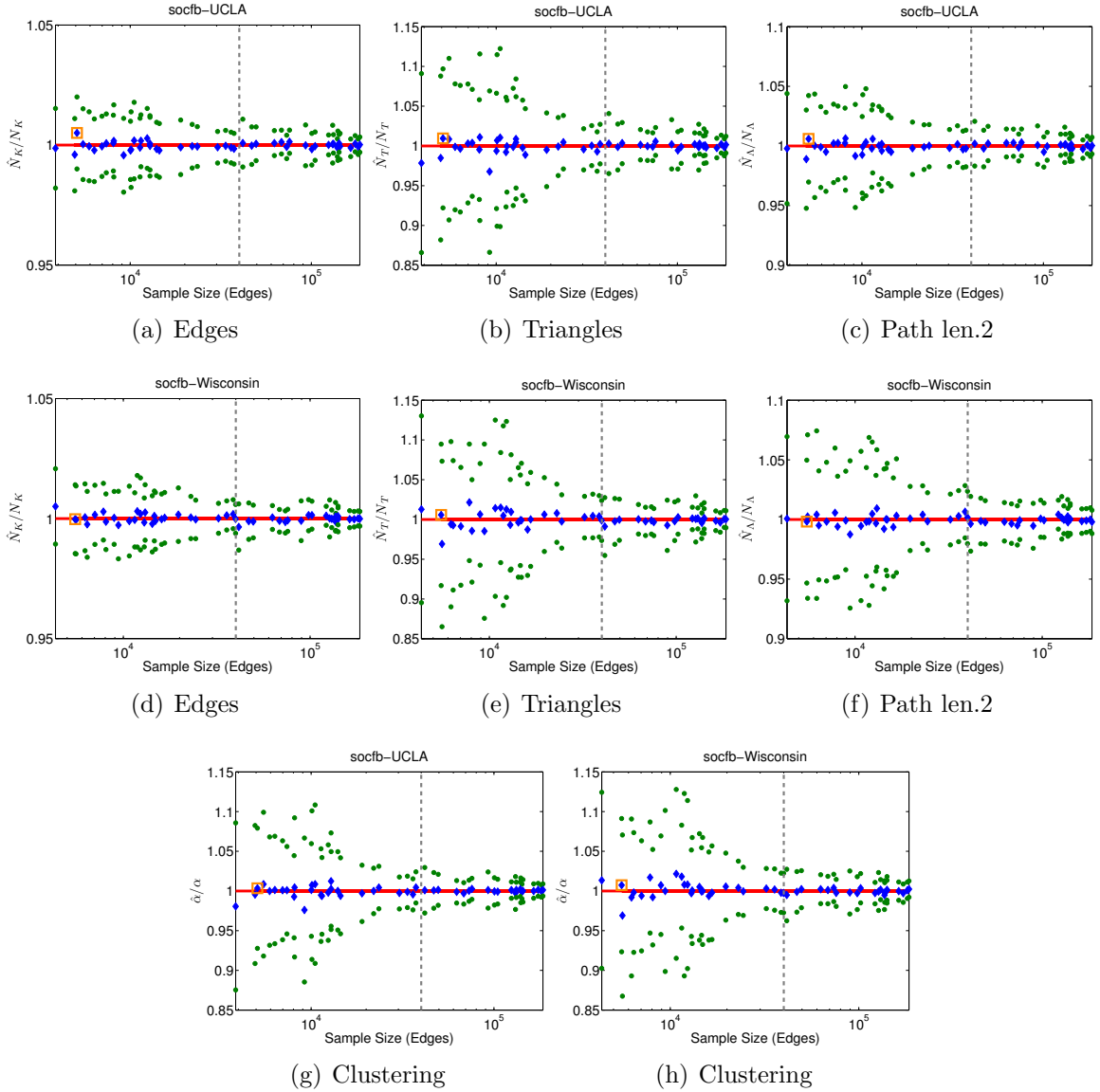


Fig. 5.1.: Convergence Analysis of Graph Sample & Hold. Convergence of the estimates (N_K , N_T , N_Λ , α , upper and lower bounds) for socfb-UCLA and socfb-Wisconsin graphs, for all possible samples with p, q in the range 0.005–0.1. Diamonds (Blue): $\frac{E[\text{est}]}{\text{Actual}}$. Circles (Green): $\frac{\text{UB}}{\text{Actual}}, \frac{\text{LB}}{\text{Actual}}$. Square (Orange): refers to the sample in Table 5.3. Dashed vertical line (Grey): refers to the sample at 40K edges

Table 5.4.: Coverage probability γ for 95% conf. interval

graph	γ_{N_K}	γ_{N_T}	γ_{N_Δ}	γ_α
socfb-CMU	0.94	0.95	0.96	0.92
socfb-UCLA	0.96	0.95	0.95	0.92
socfb-Wisconsin	0.95	0.95	0.96	0.95
web-Stanford	0.97	0.92	0.95	0.92
web-Google	0.95	0.93	0.95	0.95
web-BerkStan	0.96	0.94	0.93	0.93

Note that in Figure 5.1, we use a square (with orange color) to refer to the sample reported in Table 5.3. We also found similar observations for the rest of the graphs.

In addition to the analysis above, we compute the *exact coverage* probability γ of the 95% confidence as follows,

$$\gamma = \mathbb{P}(\text{LB} \leq \text{Actual} \leq \text{UB}) \quad (5.26)$$

For each $p = p_i, q = q_i$, we compute the proportion of samples in which the actual statistic lies in the confidence interval across 100 independent sampling experiments $\text{gSH}_T(p_i, q_i)$. We vary p, q in the range of 0.005–0.01, and for each possible combination of p, q (e.g., $p = 0.005, q = 0.008$), we compute the exact coverage probability γ . Table 5.4 provides the mean coverage probability with $p, q = \{0.005, 0.008, 0.01\}$ for all different graphs. Note $\gamma_{N_K}, \gamma_{N_T}, \gamma_{N_\Delta}$, and γ_α indicate the exact coverage probability of edge, triangle, paths of length two counts, and clustering coefficient respectively. We observe that the nominal 95% confidence interval holds to a good approximation, as $\gamma \approx 95\%$ across all graphs.

5.6.3 Comparison to Previous Work

We compare to the most recent research done on triangle counting by Jha *et al.* [19]. Jha *et al.* proposed a Streaming-Triangles algorithm to estimate the triangle counts. Their algorithm maintains two data structures. The first data structure is the edge reservoir and used to maintain a uniform random sample of edges as they

Table 5.5.: Comparison to Streaming Triangles. Relative error and sample size of Jha *et al.* [19] in comparison to our framework for triangle count estimation

graph	STREAMINGTRIANGLES		gSH _T	
	$\frac{ \hat{N}_T - N_T }{N_T}$	SSize	$\frac{ \hat{N}_T - N_T }{N_T}$	SSize
web-Stanford	≈ 0.07	40K	0.0023	14.8K
web-Google	≈ 0.04	40K	0.0029	25.2K
web-BerkStan	≈ 0.12	40K	0.0063	39.8K

streamed in. The second data structure is the wedge (path length two) reservoir and used to select a uniform sample of wedges created by the edge reservoir. The algorithm proceeds in a reservoir sampling fashion as a new edge e_t is streaming in. Then, edge e_t gets the chance to be sampled and replace a previously sampled edge with probability $1/t$. Similarly, a randomly selected new wedge (formed by e_t) replaces a previously sampled wedge from the wedge reservoir. Table 5.5 provides a comparison between our proposed framework (gSH) and the Streaming-Triangles algorithm proposed by Jha *et al.* [19]. Note that we compare with the results reported in their paper.

From Table 5.5, we observe that across the three web graphs, our proposed framework produces a relative error that is *orders of magnitude* smaller than the error produced by the Streaming-Triangles algorithm proposed in [19], and also uses a small(er) overhead storage (in most of the graphs). We note that Jha *et al.* [19] compares to other state of the art algorithms and shows that they are not practical and produce a very large error; see Section 5.7 for more details.

We have also compared to the work of Pavan *et al.* [42] and found their algorithm needs to store estimators, each of which stores *at least* one edge (≈ 36 bytes per estimator). Their algorithm also needs at least 128 estimators to obtain good results. On the other hand, gSH_T used orders of magnitude less storage to achieve even a better performance (results were omitted due to space constraints).

5.6.4 Effect of p, q on Sampling Rate

While Figure 5.1 shows that the sampling distribution of the proposed framework is unbiased regardless the choice of p, q , the question as to what effect the choice of p, q has on the sample size still needs to be explored. In this section, we study the effect of the choice of parameter settings on the fraction of edges sampled from the graph.

Figure 5.2 shows the fraction of sampled edges using gSH_T Algorithm 5.2, as we vary p, q in the range of 0.005–0.1 for two web graphs and two social Facebook graphs. Note that the graphs are ordered by their density (see Table 5.2) going from the most sparse to the most dense graph. We observe that when $q \leq 0.01$, regardless the choice of p , the fraction of sampled edges is in the range of 0.5% – 2.5% of the total number of edges in the graph. We also observe that as q goes from 0.01 to 0.03, the fraction of sampled edges would be in the range of 2.75% – 5%. These observations hold for all the graphs we studied.

On the other hand, as q goes from 0.03 to 0.1, the fraction of sampled edges depends on whether the graph is dense or sparse. For example, for the web-Google graph, as q goes from 0.03 to 0.1, the fraction of sampled edges goes from 5% to 15%. Also, for the web-Stanford graph, as q goes from 0.03 to 0.1, the fraction of sampled edges goes from 5% to 25%. However, for the most dense graph we have in this chapter (socfb-CMU), the fraction of sampled edges goes from 5% to 31%. Note that when we tried $q = 1$, regardless the choice of p , more than 80% of the edges were sampled.

Since p is the probability of sampling a fresh edge (not adjacent to a previously sampled edge), one could think of p as the probability of random jumps (similar to random walk methods) to explore unsampled regions in the graph. On the other hand, q is the probability of sampling an edge adjacent to previous edges. Therefore, one could think of q as the probability of exploring the neighborhood of previously sampled edges (similar to the forward probability in Forest Fire sampling).

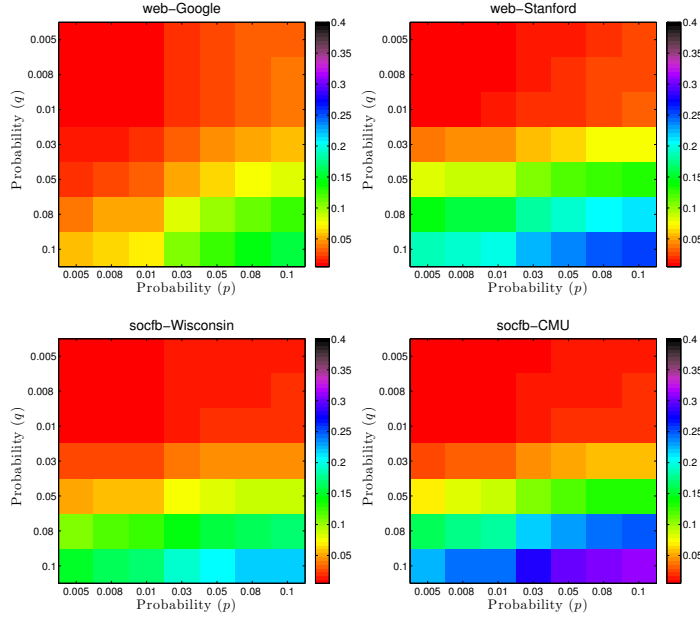


Fig. 5.2.: Analysis of sample and hold probabilities. Sampling Fraction ($\frac{SSize}{NK}$) as p, q changes in the range ‘0.005–0.1’ for web and social graphs (ordered from sparse \rightarrow dense).

From all the discussion above, we conclude that using a small p, q settings (i.e., ≤ 0.008) is better to control the fraction of sampled edges, and also recommended since the sampling distribution of the proposed framework is unbiased regardless the choice of p, q as we show in Figure 5.1 (also see Section 5.3). However, if a tight confidence interval is needed, then increasing p, q helps to reduce variance.

5.6.5 Implementation Issues

In practice, statistical variance estimators are costly to compute. In this chapter, we provide an efficient parallel procedure to compute the variance estimate. As an example, we illustrate this for the task of computing the variance of the triangle estimate ($Var(\widehat{N}_T)$ from Section 5.4.3). Consider any pair of triangles τ and τ' . Assuming τ and τ' are not identical, the covariance of τ and τ' is greater than zero (i.e., $Cov(\tau, \tau') > 0$), if and only if the two triangles are intersecting in one edge

Table 5.6.: Runtime for sampling and estimation using gSH_T

graph	Full Graph		Sampled Graph	
	Time	Graph size	Time	SSize
web-Stanford	19.68	1.9M	0.13	14.8K
web-Google	5.05	4.3M	0.55	25.2K
web-BerkStan	113.9	6.6M	1.05	39.8K

$e(\tau, \tau')$. Since two intersecting triangles have either one edge in common or are identical, we can find intersecting triangles by finding all triangles *incident* to a particular edge e . In this case, the intersection probability of the two triangles is $P(\tau \cap \tau') = P(e(\tau, \tau'))$. Note that if τ and τ' are identical, then the computation is straightforward. The procedure is very simple as follows,

- Given a sample set of edges $\widehat{K} \subset K$, for each edge $e \in \widehat{K}$
 - find the set of all triangles ‘ T_e ’ incident to e
 - for each pair of triangles (τ, τ') , where $\tau, \tau' \in T_e$, and $\tau \neq \tau'$, compute the $Cov(\tau, \tau')$ such that $P(\tau \cap \tau') = P(e(\tau, \tau'))$

Since, the computation of each edge is independent of other edges, we parallelize the computation of the variance estimators. Moreover, since the computation of triangle counts and paths of length two can themselves be parallelized, we compare the total elapsed time in seconds used to compute these counts on both the full graph and a sampled graph of size $\leq 40K$ edges. Table 5.6 provide the results of this comparison for the three web graphs. Note that in the case of the sampled graph, we report the sum of the total computations of both the variance estimators and expected values of the triangle and paths of length two count statistics. Also, note that we use the sample reported in Table 5.3 for these computations. The results show a significant reduction in the time needed to compute triangles and paths of length two counts. For example, consider the web-BerkStan graph, where the total time is reduced from 113 seconds to 1.05 seconds. Note that all the computations of Table 5.6 are performed on a MacPro laptop 2.9GHZ Intel Core i7 with 8GB memory.

5.7 Related Work

In this section, we discuss the related work on the problem of large-scale graph analytics and their applications. Generally speaking, there are two bodies of work related to this chapter: (i) graph analytics in the graph stream setting, and (ii) graph analytics in the non-streaming setting (e.g. using MAPREDUCE). In this chapter, we propose a generic stream sampling framework for big-graph analytics, called Graph Sample and Hold (gSH), that works in a *single pass* over the stream. Therefore, we focus on the related work for graph analytics in the graph stream setting.

Graph Analysis Using Streaming Algorithms. Before exploring the literature of graph stream analytics, we briefly review the literature in data stream analysis and mining that may not contain graph data. For example, for sequence sampling (e.g., reservoir sampling) [38, 109], for computing frequency counts [111], and for mining concept drifting data streams [117]. Additionally, the idea of *sample and hold* (SH) was introduced in [148] for unbiased sampling of network data with integral weights. Subsequently, other work explored adaptive SH, and SH with signed updates [154, 155]. Nevertheless, none of this work has considered the framework of *sample and hold* (SH) for social and information networks. In this chapter, however, we propose the framework of *graph sample and hold* (gSH) for big-graph analytics.

There has been an increasing interest in mining, analysis, and querying of massive graph streams as a result of the proliferation of graph data (e.g., social networks, emails, IP traffic, Twitter hashtags). For example, to count triangles [8, 19, 42–44, 126], finding common neighborhoods [127], estimating pagerank values [128], and characterizing degree sequences in multi-graph streams [129]. In the data mining field, there is the work done on clustering and outlier detection in graph streams [40, 130].

Much of this work has used various sampling schemes to sample from the stream of graph edges [11]. Surprisingly, the majority of this work has focused primarily on sampling schemes that can be used to estimate certain graph properties (e.g. triangle counts), while much less is known for the case when we need a generic approach

to estimate various graph properties with the same sampling scheme with minimum assumptions.

For example, the work done in [43] proposed an algorithm with space bound guarantees for triangle counting and clustering estimation in the *incidence stream model* where all edges incident to a node are arriving in order together. However, in the incidence stream model, counting triangles is a relatively easy problem, and counting the number of paths of length two is simply straightforward. On the other hand, it has been shown that these bounds and accurate estimates will no longer hold in the case of *adjacency stream model*, where the edges arrive arbitrarily with no particular order [19, 42].

Another example, the work done Jha *et al.* in [19] proposed a practical, single pass, $O(\sqrt{n})$ -space streaming algorithm specifically for triangle counting and clustering estimation with additive error guarantee (as opposed to other algorithms with relative error guarantee). Although, the algorithm is practical and approximates the triangle counts accurately at a sample size of $40K$ edges, their method is specifically designed for triangle counting. Nevertheless, we compare to the results of triangle counts reported in [19], and we show that our framework is not only generic but also produces errors with orders of magnitude less than the algorithm in [19], and with a small(er) storage overhead in many times.

More recently, Pavan *et al.* proposed a space-efficient streaming algorithm for counting and sampling triangles in [42]. This algorithm works in a single pass streaming fashion with order $O(N_K \Delta / N_T)$ -space, where Δ is the maximum degree of the graph. However, this algorithm needs to store estimators (i.e., wedges that may form potential triangles), and each of these estimators stores *at least* one edge. In their paper, they show that they need at least 128 estimators (i.e., more than $128K$ edges), to obtain accurate results (i.e., large storage overhead compared to those in this chapter).

Other semi-streaming algorithms were proposed for triangle counting, such as the work in [8], however, they are not practical and produce large error as discussed and analyzed by the work in [42].

Horvitz-Thompson estimation was proposed for social networks by Frank [156], including applications to subgraph sampling, but limited to a model of simple random sampling of vertices without replacement; see also Kolaczyk [157].

Graph Analysis Using Static and Parallel Algorithms. We briefly review other research for graph analysis in non-streaming setting (i.e., static). For example, exact counting of triangles with runtime $O(N_K^{3/2})$ [18], or approximately by sampling edges as in [16]. Although not working in a streaming fashion, the algorithm in [16] uses unbiased estimators of triangle counts similar to our work. Moreover, other algorithms were proposed based on wedge sampling and proved to be accurate in practice, such as the work in [158]. More recently, the work done in [159] proposed a parallel framework for finding the maximum clique.

Finally, there has been an increasing interest in the general problem of network sampling. For example, to obtain a representative subgraph [9, 11], and to preserve the community structure [28], and other sampling goals [101, 102].

5.8 Summary

In this chapter, we presented a generic framework for big-graph analytics called graph sample and hold (gSH). The gSH framework samples from massive graphs *sequentially in a single pass*, one edge at a time, while maintaining a small state typically less than 1% of the total number of edges in the graph. Our contributions can be summarized in the following points:

- We show how to produce unbiased estimators and their variance for four specific graph quantities of interest to estimate within the framework. Further, we show how to obtain confidence bounds using the variance unbiased estimators.

- We conduct several experiments on real world graphs, such as social Facebook graphs, and web graphs. The results show that the relative error ranging from 0.02% to 0.95% for a sample with $\leq 40\text{K}$ edges. Moreover, the results show that the sampling distribution is centered and balanced over the actual values of the four graph quantities of interest, with tight error bounds as the sample size increases.
- We compare to the state of the art and our proposed framework has a relative error *orders of magnitude* less than the Streaming-Triangles algorithm proposed in [19], as well as with a small(er) overhead storage (in most of the graphs).
- We show how to parallelize and efficiently compute the unbiased variance estimators, and we discuss the significant reductions in computation time that can be achieved by gSH framework.

6. FAST PARALLEL MOTIF COUNTING FOR LARGE GRAPHS

From social science to biology, numerous applications often rely on motifs for intuitive and meaningful characterization of networks at both the global macro-level as well as the local micro-level. While motifs have witnessed a tremendous success and impact in a variety of domains, there has yet to be a fast and efficient approach for computing the frequencies of these subgraph patterns. However, existing methods are not scalable to large networks with millions of nodes and edges, which impedes the application of motifs to new problems that require large-scale network analysis. To address these problems, we propose a fast, efficient, and parallel algorithm for counting motifs of size $k = \{3, 4\}$ -nodes that take only a fraction of the time to compute when compared with the current methods used. The proposed motif counting algorithms leverages a number of proven combinatorial arguments for different motifs. For each edge, we count a few motifs, and with these counts along with the combinatorial arguments, we obtain the exact counts of others in constant time. On a large collection of 300+ networks from a variety of domains, our motif counting strategies are on average 460x faster than current methods. This brings new opportunities to investigate the use of motifs on much larger networks and newer applications as we show in our experiments. To the best of our knowledge, this dissertation provides the largest motif computations to date as well as the largest systematic investigation on over 300+ networks from a variety of domains.

6.1 Motivation

Recursive decomposition of networks is a widely used approach in network analysis to factorize the complex structure of real-world networks into small subgraph

patterns of size k nodes, these patterns are called *motifs* [160]. Motifs (also known as graphlets [161]) are defined as subgraph patterns recurring in real-world networks at frequencies that are statistically significant from those in random networks. Given a network, we can count up the number of embeddings of each motif pattern in the network, creating a profile of sufficient statistics that characterizes the network structure [162]. While knowing the motif frequencies does not uniquely define the network structure, it has been shown that motif frequencies often carry significant information about the local network structure in a variety of domains [163–165]. This is in contrast to global topological properties (e. g., diameter, degree distribution), where networks with similar/exact global topological properties can exhibit significantly different local structure.

6.2 Motifs, Scalability, Applications

From social science to biology, motifs have found numerous applications and were used as the building blocks of network analysis [160]. In social science, motif analysis (typically known as k -subgraph census) is widely adopted in sociometric studies [163, 165]. Much of the work in this vein focused on analyzing triadic tendencies as important structural features of social networks (e. g., transitivity or triadic closure [166]) as well as analyzing triadic configurations as the basis for various social network theories (e. g., social balance, strength of weak ties, stability of ties, or trust [167]). In biology [5, 161], motifs were widely used for protein function prediction [162], network alignment [168], and phylogeny [169] to name a few. More recently, there has been an increased interest in exploring the role of motif analysis in computer networking [8, 170, 171] (e. g., for web spam detection, analysis of peer-to-peer protocols and Internet AS graphs), chemoinformatics [172, 173], image segmentation [174], among others [175].

While motif counting and discovery have witnessed a tremendous success and impact in a variety of domains from social science to biology, there has yet to be a fast

and efficient approach for computing the frequencies of these patterns. For instance, Shervashidze *et al.* [162] takes hours to count motifs on relatively small biological networks (i.e., few hundreds/thousands of nodes/edges) and uses such counts as features for graph classification [176]. Previous work showed that motif counting is computationally intensive since the number of possible k -subgraphs in a graph G increases exponentially with k in $\mathcal{O}(|V|^k)$ and can be computed in $\mathcal{O}(|V|\Delta^{k-1})$ for any bounded degree graph, where Δ is the maximum degree of the graph [162].

To address these problems, we propose a fast, efficient, and parallel algorithm for counting motifs of size $k = \{3, 4\}$ -nodes that take only a fraction of the time to compute when compared with the current methods used. The proposed motif counting algorithm leverages a number of proven combinatorial arguments for different motifs. For each edge, we count a few motifs, and with these counts along with the combinatorial arguments, we obtain the exact counts of others in constant time. On a large collection of 300+ networks from a variety of domains, our motif counting strategies are on average 460x faster than current methods. This brings new opportunities to investigate the use of motifs on much larger networks and newer applications as we show in our experiments. To the best of our knowledge, this chapter provides the largest motif computations to date as well as the largest systematic investigation on over 300+ networks from a variety of domains.

Furthermore, a number of important machine learning tasks are likely to benefit from such an approach, including graph anomaly detection [7, 46], entity resolution [47], as well as features for improving community detection [48], role discovery [49], and relational classification [50].

Recently, there is an increased interest in sampling and other heuristic approaches for obtaining approximate counts of various motif patterns [23, 103, 177]. However, our work focuses on exact motif counting and thus approximation/sampling methods are outside the scope of our work. Nevertheless, our fast and efficient motif algorithms may be used to speedup sampling research, as it can be used for counting the various

motif patterns once a subgraph is sampled, and thus our work is independent of the chosen sampling method [11].

We test the scalability of our proposed approach experimentally on 300+ networks from biological, social, and technological domains [178]. We compare our approach to the state of the art exact counting methods such as RAGE [179], FANMOD [180], and Orca [181]. We found that RAGE [179] took 2400 seconds to count the motifs on a small 26k node graph, whereas our proposed method is 460x faster, taking only 0.01 seconds. We also note that FANMOD [180], another recent approach, takes 172800 seconds, and Orca [181] takes 2.5 seconds for the same small graph. Our exact motif analysis is well-suited for shared-memory multi-core architectures (CPU and GPU), distributed architectures (MPI), and hybrid implementations that leverage the advantages of both.

6.3 Background

Motifs are subgraph patterns recurring in real-world networks at frequencies that are significantly higher than those in random networks [160, 161]. Previous work showed that motifs can be used to define universal classes of networks [160]. Moreover, motifs are at the heart and foundation of many network analysis tasks (e. g., network classification, network alignment, etc.) [5, 161, 182]. In this chapter, we introduce an efficient algorithm to compute the number of embeddings of each motif pattern of size $k = \{3, 4\}$ nodes in the network, creating a profile of sufficient statistics that characterizes the network structure.

6.3.1 Notations and Definitions

Given an undirected simplified input graph $G = (V, E)$ with no self edges, a motif of size k nodes is defined as any subgraph $G_k \subset G$ which consists of a subset of k nodes of the graph G . We distinguish between induced and partially-induced motifs. An *induced* motif is an induced subgraph that consist of *all* edges between its nodes that

are present in the input graph (as described in Definition 1), while a *partially-induced* motif is a relaxed notion of a motif, where the subgraph pattern may contain only *some* of these edges. In this chapter, we mainly focus on computing the frequencies of induced motifs. In Table 6.1, we provide a summary of the notation and properties of all possible induced motifs of size $k = \{3, 4\}$.

Definition 1 *Induced Motif:* an induced motif $G_k = (V_k, E_k)$ is a subgraph that consists of a subset of k vertices of the graph $G = (V, E)$ (i.e., $V_k \subset V$) together with all the edges whose endpoints are both in this subset (i.e., $E_k = \{\forall e \in E \mid e = (u, v) \wedge u, v \in V_k\}$).



































In addition, we distinguish between *connected* and *disconnected* induced motifs (see Table 6.1). A motif is connected if there is a path from any node to any other node in the motif. An induced motif that is not connected is said to be disconnected (as in Definition 2).

Definition 2 *Connected Motif:* a motif $G_k = (V_k, E_k)$ is connected when there is a path from any node to any other node in the motif (i.e., $\forall u, v \in V_k, \exists P_{u-v} : u, \dots, w, \dots, v$, such that $d(u, v) \geq 0 \wedge d(u, v) \neq \infty$). By definition, there exist one and only one connected component in a motif G_k (i.e., $|C| = 1$) if and only if G_k is connected.

Problem Definition. Given a family of motifs of size k nodes $\mathcal{G}_k = \{g_{k_1}, g_{k_2}, \dots, g_{k_m}\}$, our goal is to count the number of embeddings (appearances) of each motif $g_{k_i} \in \mathcal{G}_k$ in the input graph G . In other words, we need to count the number of induced motifs G_k in G that are isomorphic to each motif pattern $g_{k_i} \in \mathcal{G}_k$ in the family, such a number is denoted by $\binom{G}{g_{k_i}}$ [183]. A motif $g_{k_i} \in \mathcal{G}_k$ is embedded in the graph G , if and only if there an injective mapping $\sigma : V_{g_{k_i}} \rightarrow V$, with $e = (u, v) \in E_{g_{k_i}}$ and $e' = (\sigma(u), \sigma(v)) \in E$. Table 6.1 shows that $|\mathcal{G}_k| = \{2, 4, 11\}$ when $k = \{2, 3, 4\}$ respectively.

Table 6.1.: Summary of motif notation and properties

Summary of the notation and properties for the motifs of size $k = \{2, 3, 4\}$. Note that ρ denotes density, Δ and \bar{d} denote the max and mean degree, whereas assortativity is denoted by r . Also, $|T|$ denotes the total number of triangles, K is the max k-core number, χ denotes the Chromatic number, whereas D denotes the diameter, B denotes the max betweenness, and $|C|$ denotes the number of components. Note that if $|C| > 1$, then r , D, and B are from the largest component.

Motif	Description	Complement	ρ	Δ	\bar{d}	r	$ T $	K	χ	D	B	$ C $	
(k = 4)–MOTIFS													
CONNECTED		g_{4_1} 4-clique		1.00	3	3.0	1.00	4	3	4	1	0	1
		g_{4_2} 4-chordal cycle		0.83	3	2.5	-0.66	2	2	3	2	1	1
		g_{4_3} 4-tailed triangle		0.67	3	2.0	-0.71	1	2	3	2	2	1
		g_{4_4} 4-cycle		0.67	2	2.0	1.00	0	2	2	2	1	1
		g_{4_5} 3-star		0.50	3	1.5	-1.00	0	1	2	2	3	1
		g_{4_6} 4-path		0.50	2	1.5	-0.50	0	1	2	3	2	1
DISCONNECTED		g_{4_7} 4-node-1-triangle		0.50	2	1.5	1.00	1	2	3	1	0	2
		g_{4_8} 4-node-2-star		0.33	2	1.0	-1.00	0	1	2	2	1	2
		g_{4_9} 4-node-2-edge		0.33	1	1.0	1.00	0	1	2	1	0	2
		$g_{4_{10}}$ 4-node-1-edge		0.17	1	0.5	1.00	0	1	2	1	0	3
		$g_{4_{11}}$ 4-node-independent		0.00	0	0.0	0.00	0	0	1	∞	0	4
(k = 3)–MOTIFS													
	g_{3_1} triangle		1.00	2	2.0	1.00	1	2	3	1	0	1	
	g_{3_2} 2-star		0.67	2	1.33	-1.00	0	1	2	2	1	1	
	g_{3_3} 3-node-1-edge		0.33	1	0.67	1.00	0	1	2	1	0	2	
	g_{3_4} 3-node-independent		0.00	0	0.00	0.00	0	0	1	∞	0	3	
(k = 2)–MOTIFS													
	g_{2_1} edge		1.00	1	1.0	1.00	0	1	2	1	0	1	
	g_{2_2} 2-node-independent		0.00	0	0.0	0.00	0	0	1	∞	0	2	

Further, given a family $\mathcal{G}_k = \{g_{k_1}, g_{k_2}, \dots, g_{k_m}\}$ of motifs of size k nodes, we define $f(g_{k_i}, G)$ as the relative frequency of occurrence of any motif $g_{k_i} \in \mathcal{G}_k$ in the input graph G .

6.3.2 Relation to Graph Complement

The complement of a graph G , denoted by \bar{G} , is the graph defined on the same vertices as G such that two vertices are connected in \bar{G} if and only if they are not connected in G . Therefore, the graph sum $G + \bar{G}$ gives the complete graph on the set of vertices of G . There are direct relationships between the frequencies of motifs and the frequencies of their complement. For each motif g_{k_i} , there exists a non-isomorphic complementary motif pattern \bar{g}_{k_i} , such that two vertices are connected in \bar{g}_{k_i} if and only if they are not connected in g_{k_i} [183]. For example, cliques and independent sets of any size nodes are pairs of complementary motifs. Similarly, chordal cycles of size 4 nodes are complementary to the motif pattern 4-node-1edge (see Table 6.1). It is also worth to note that the 4-path motif pattern is a self-complementary pattern, which means the 4-path is isomorphic to its complement.

From this discussion, it is clear that the number of embeddings of each motif $g_{k_i} \in \mathcal{G}_k$ in the input graph G is equivalent to the number of embeddings of its complementary motif \bar{g}_{k_i} in the complement graph \bar{G} . In other words, $f(g_{k_i}, G) = f(\bar{g}_{k_i}, \bar{G})$ [183, 184].

6.3.3 Relation to Graph & Matrix Reconstruction Theorems

The graph reconstruction conjecture [183], states that an undirected graph G can be uniquely determined up to an isomorphism, from the set of all possible vertex-deleted subgraphs of G (i. e., $\{G_v\}_{v \in V}$) [185]. Verification of this conjecture for all possible graphs up to 6 vertices was carried by Kelly [186], and later was extended to up to 11 vertices by McKay [185]. Clearly, if two graphs are isomorphic (i. e., $G \cong G'$), then their motif frequencies would be the same (i. e., $f_k(G) = f_k(G')$), but the reverse remains a conjecture for the general case of graphs. In contrast, the matrix reconstruction theorem has been resolved [187], which states that any $N \times N$ matrix can be reconstructed from its list of all possible principal minors obtained by

the deletion of the k -th row and the k -th column [187], which is the foundation of graphlet kernel [162].

The aim and scope of this chapter is different from the problem of graph reconstruction. While graph reconstruction tries to test for the notion of isomorphism and structure equivalence between graphs, our goal is to relax the notion of equivalence to some form of *structural similarity* between graphs, such that the graph similarity is measured on the feature representation space of motifs.

6.4 Framework

In this section, we describe our proposed fast and efficient algorithm for motif counting that takes only a fraction of the time to compute when compared with the current methods used. We introduce a number of combinatorial arguments that we show for different motif patterns. The proposed motif counting algorithm leverages these combinatorial arguments to obtain significant improvement on the scalability of motif counting. For each edge, we count only a few of motifs, and with these counts along with the combinatorial arguments, we derive the exact counts of the others in constant time.

6.4.1 Searching Edge Neighborhoods

Our proposed algorithm iterates over all the edges of the input graph $G = (V, E)$. For each edge $e = (u, v) \in E$, we define the *neighborhood* of an edge e , denoted by $\mathcal{N}(e)$, as the set of all nodes that are connected to the endpoints of e — i. e., $\mathcal{N}(e) = \{\mathcal{N}(u) \setminus \{v\}\} \cup \{\mathcal{N}(v) \setminus \{u\}\}$, where $\mathcal{N}(u)$ and $\mathcal{N}(v)$ are the set of neighbors of u and v respectively. Given a single edge $e = (u, v) \in E$, we explore the subgraph surrounding this edge — i. e., the subgraph induced by both its endpoints and the nodes in its neighborhood. We call this subgraph the *egonet* of the edge e , where e is the center (ego) of the subgraph.

We search for possible motif patterns of size $k = \{3, 4\}$ in the egonets of all edges in the graph. By searching egonets of edges, we first map the problem to the local (lower-dimensional) space induced by the neighborhood of each edge, and then merge the search results for all edges. Searching over a local low-dimensional space of edge neighborhoods is clearly more efficient than searching over the global high-dimensional space of the whole graph. Moreover, searching over a local low-dimensional space of edge neighborhoods is amenable to parallel implementation, which offers additional speedup over iterative methods. Note that exhaustive search of the egonet of any edge $e \in E$ yields at least $\mathcal{O}(\Delta^{k-1})$ asymptotically, where Δ is the maximum degree in G . Clearly, exhaustive search is computationally intensive for large graphs, and our approach is more efficient as we will show next.

6.4.2 Counting Motifs of Size ($k = 3$) Nodes

Algorithm 6.1: TRIADCENSUS($G = (V, E)$) our exact triad census algorithm for counting all 3-node motifs. This Algorithm takes an undirected graph and returns the frequencies of all 3-node motifs $f(\mathcal{G}_3, G)$.

Input : Graph $G = (V, E)$
Output: Motif Counts of size 3 nodes $f(\mathcal{G}_3, G)$

```

1 Initialize Array  $X$ 
2 for each edge  $e = (u, v) \in E$  in parallel do
3    $\text{Star}_u = \emptyset, \text{Star}_v = \emptyset, \text{Tri}_e = \emptyset$ 
4   for each  $w \in \mathcal{N}(u)$  do
5     if  $w = v$  then continue
6     Add  $w$  to  $\text{Star}_u$  and Set  $X(w) = 1$ 
7   for each  $w \in \mathcal{N}(v)$  do
8     if  $w = u$  then continue
9     if  $X(w) = 1$  then  $\rightarrow$  found triangle
10    Add  $w$  to  $\text{Tri}_e$ 
11    Remove  $w$  from  $\text{Star}_u$ 
12   else Add  $w$  to  $\text{Star}_v$ 
13    $f(g_{31}, G) += |\text{Tri}_e|$ 
14    $f(g_{32}, G) += |\text{Star}_u| + |\text{Star}_v|$ 
15    $f(g_{33}, G) += |V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|$ 
16   for each  $w \in \mathcal{N}(u)$  do  $X(w) = 0$ 
17  $f(g_{31}, G) = 1/3 \cdot f(g_{31}, G)$ 
18  $f(g_{32}, G) = 1/2 \cdot f(g_{32}, G)$ 
19  $f(g_{34}, G) = \binom{|V|}{3} - f(g_{31}, G) - f(g_{32}, G) - f(g_{33}, G)$ 
20 return  $f(\mathcal{G}_3, G)$ 

```

Alg. 6.1 (TRIADCENSUS) shows how to count motifs of size $k = 3$ for each edge. There are four possible motifs of size $k = 3$ nodes, where only g_{3_1} (i. e., triangle patterns) and g_{3_2} (i. e., 2-star patterns) are connected motifs (see Table 6.1).

Connected motifs of size $k = 3$. Lines 4–12 of Algorithm 6.1 show how to find and count triangles incident to an edge. For any edge $e = (u, v)$, a triangle (u, v, w) exists, if and only if w is connected to *both* u and v . Let Tri_e be the set of all nodes that form a triangle with $e = (u, v)$, and $|\text{Tri}_e|$ be the number of such triangles. Then, Tri_e is the set of overlapping nodes in the neighborhoods of u and v — $\text{Tri}_e = \mathcal{N}(u) \cap \mathcal{N}(v)$.

Note that Algorithm 6.1 counts each triangle three times (one time for each edge in the triangle), and therefore we divide the total count by 3 as in Eq. (6.1),

$$f(g_{3_1}, G) = \frac{1}{3} \cdot \sum_{e=(u,v) \in E} |\text{Tri}_e| \quad (6.1)$$

Now we need to count 2-star patterns (i. e., g_{3_2}). For any edge $e = (u, v)$, let Star_e be the set of all nodes that form a 2-star with e , and $|\text{Star}_e|$ be the number of such star patterns. A 2-star pattern (u, v, w) exists, if and only if w is connected to *either* u or v but not both. Accordingly, $\text{Star}_e = \text{Star}_u \cup \text{Star}_v$, where Star_u and Star_v are the set of nodes that form a 2-star with e centered at u and v respectively. More formally, Star_u can be defined as $\text{Star}_u = \{w \in \mathcal{N}(u) \setminus \{v\} | w \notin \mathcal{N}(v)\}$, and Star_v can be defined as $\text{Star}_v = \{w \in \mathcal{N}(v) \setminus \{u\} | w \notin \mathcal{N}(u)\}$.

Similar to counting triangles, Algorithm 6.1 counts each 2-star pattern two times (one time for each edge in the 2-star). Thus, we divide the sum for all edges by 2 as stated in Eq. (6.2),

$$f(g_{3_2}, G) = \frac{1}{2} \cdot \sum_{e=(u,v) \in E} |\text{Star}_u| + |\text{Star}_v| \quad (6.2)$$

Disconnected motifs of size $k = 3$. There are two disconnected motifs of size $k = 3$ nodes, g_{3_3} (i. e., the 3-node-1-edge pattern) and g_{3_4} (i. e., the independent set defined on 3 nodes) (see Table 6.1). Lines 15 and 19 show how to count these patterns.

Eq. 6.3 shows that the number of 3-node-1-edge motifs per edge e is equivalent to the number of all nodes that are not in the neighborhood subgraph (egonet) of edge e (i. e., $V \setminus \{\mathcal{N}(u) \cup \mathcal{N}(v)\}$),

$$f(g_{3_3}, G) = \sum_{e=(u,v) \in E} |V| - |\mathcal{N}(u) \cup \mathcal{N}(v)| \quad (6.3)$$

where $|\mathcal{N}(u) \cup \mathcal{N}(v)| = |\text{Tri}_e| + |\text{Star}_e| + |\{u, v\}|$. Note that the number of 3-node-1-edge motifs can be computed in $o(1)$ for each edge.

Given that the total number of motifs of size 3 nodes is $\binom{N}{3}$, Eq. 6.4 shows how to compute the frequency of g_{3_4} , which clearly can be done in $o(1)$,

$$f(g_{3_4}, G) = \binom{|V|}{3} - (f(g_{3_1}, G) + f(g_{3_2}, G) + f(g_{3_3}, G)) \quad (6.4)$$

The complexity of counting all motifs of size $k = 3$ is $\mathcal{O}(|E| \cdot \Delta)$ asymptotically as we show next in Lemma 6.4.1.

Lemma 6.4.1 *Alg. 6.1 counts all motifs of size $k = 3$ -nodes in $\mathcal{O}(|E| \cdot \Delta)$.*

Proof For each edge $e = (u, v)$ such that $e \in E$, the runtime complexity of counting all triangle and 2-star patterns incident to e (i. e., $\text{Tri}_e, \text{Star}_e$ respectively as shown in Lines 4–12) is $O(|\mathcal{N}(u)| + |\mathcal{N}(v)|)$, and is asymptotically $O(\Delta)$ where Δ is the maximum degree in the graph. Further, the runtime complexity of counting all 3-node-1-edge patterns of size $k = 3$ incident to e can be counted in constant time $o(1)$ (Lines 15 and 19). Therefore, the total runtime complexity for counting all motifs of size $k = 3$ in the graph is $\mathcal{O}\left(\sum_{e \in E} (\Delta + o(1))\right) = \mathcal{O}(|E| \cdot \Delta)$. ■

6.5 Counting Motifs of Size ($k = 4$) Nodes

An exhaustive search of the egonet of any edge to count all 4-node motifs independently yields $\mathcal{O}(\Delta^3)$ asymptotically, where Δ is the maximum degree in G . Clearly, exhaustive search is computationally intensive for large graphs. On the other hand, our approach is hierarchical and more efficient as we show next.

For each edge $e = (u, v)$, we start by finding triangles and 2-star patterns. Our central principle is that any 4-node motif g_{4_i} can be decomposed into four 3-node motifs [183], obtained by deleting one node from g_{4_i} each time. Thus, we jointly count all possible 4-node motifs by leveraging the knowledge obtained from finding 3-node motifs and combinatorial arguments that describe relationships between pairs of motif patterns. We summarize this procedure in the following steps:

- *Step 1*: For each edge $e = (u, v)$, find all neighborhood nodes forming triangle and 2-star patterns with e
- *Step 2*: For each edge $e = (u, v)$, use the knowledge from Step 1 to count only 4-cliques and 4-cycles
- *Step 3*: For each edge $e = (u, v)$, use the knowledge from Step 1 and combinatorial arguments to compute unrestricted counts for all 4-node motifs (i. e., counts that can be computed in constant time)
- *Step 4*: Merge the counts from all edges in the graph, and use combinatorial arguments involving unrestricted counts (computed in Step 3) to obtain the exact frequencies of all 4-node motifs

Note that we refer to unrestricted counts as the counts that can be computed in constant time and using previous knowledge from Step 1. We discuss the details later in Section 6.5.2.

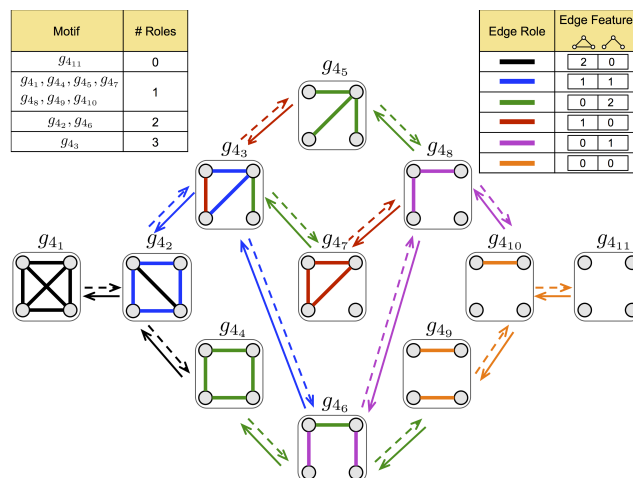


Fig. 6.1.: 4-node Motif Transition Diagram: Figure shows all possible ± 1 *edge* transitions between the set of all 4-node motifs. Dashed right arrows denote the deletion of one edge to transition from one motif to another. Solid left arrows denote the addition of one edge to transition from one motif to another. Edges are colored by their feature-based roles, where the set of feature are defined by the number of triangles and 2-stars incident to an edge (see Table in the top-right corner). We define six different classes of edge roles colored from black to orange (see Table in the top-right corner). Dashed/solid arrows are colored similar to the edge roles to denote which edge would be deleted/added to transition from one motif to another. The table in the top-left corner shows the number of edge roles per each motif.

6.5.1 Motif State Transition Diagram

Assume that each motif pattern is a state, Fig. 6.1 shows all possible ± 1 *edge* transitions between the states of all 4-node motifs. We can transition from one motif to another by the deletion (denoted by dashed right arrows) or addition (denoted by solid left arrows) of a single edge. We define six different classes of possible edge roles denoted by the colors from black to orange (see Table in the top-right corner in Fig. 6.1). An *edge role* is an edge-level connectivity pattern (e. g., a chord edge), where two edges belong to the same role (i. e., class) if they are similar in their topological features. For each edge, we define a topological feature vector that consists of the number of triangles and 2-stars incident to this edge. Then, we classify edges to one of the six roles based on their feature vectors. All edges that appear in 4-node motifs are colored by their roles. In addition, the transition arrows are colored similar to the

edge roles to denote which edge type should be deleted/added to transition from one motif to another. Note that a single edge deletion/addition changes the role (class) of other edges in a motif. The table in the top-left corner of Fig. 6.1 shows the number of edge roles per each motif.

For example, consider the 4-clique motif (g_{4_1}), where each edge participates exactly in two triangles. Therefore, all the edges in a 4-clique motif (g_{4_1}) belong to the first role (denoted by the black color). Similarly, consider the 4-chordalcycle motif (g_{4_2}), where each edge (except the chord edge) participates exactly in one triangle and one 2-star. Therefore, all edges in a 4-chordalcycle motif “ g_{4_2} ” belong to the second role (denoted by the blue color) except for the chord edge which belongs to the first role (denoted by the black color). Fig. 6.1 shows how to transition from the 4-clique motif to the 4-chordalcycle motif “ g_{4_2} ” by deleting one (any) edge from the 4-clique motif.

6.5.2 General Principle for Counting Motifs of size $k = 4$

Generally speaking, suppose we have $N^{(e)}$ distinct 4-node subgraphs that contains an edge $e = (u, v)$,

$$N^{(e)} = \left| \left\{ \{u, v, w, r\} \mid w, r \in V \setminus \{u, v\} \wedge w \neq r \right\} \right| \quad (6.5)$$

Now, each subgraph $\{u, v, w, r\}$ in this collection may satisfy one or two properties $a_i, a_j \in A = \{T, S_u, S_v, I\}$. These properties describe the topological properties of nodes w and r with respect to edge e , such that $A_w = a_i$ if $\{u, v, w\}$ forms subgraph pattern a_i , and $A_r = a_j$ if $\{u, v, r\}$ forms subgraph pattern a_j . For example, $A_w = T$ if w forms a triangle with e , and $A_w = S_u$ or S_v if w forms a 2-star with e centered around u or v respectively. Also, $A_w = I$ if w is independent (disconnected) from e . We clarify these properties by example in Fig. 6.2.

Accordingly, let $N_{a_i}^{(e)}$ denote the number of 4-node motifs $\{u, v, w, r\}$ having property $a_i \in A$, and let $N_{a_i, a_j}^{(e)}$ denote the number having properties $a_i, a_j \in A$,

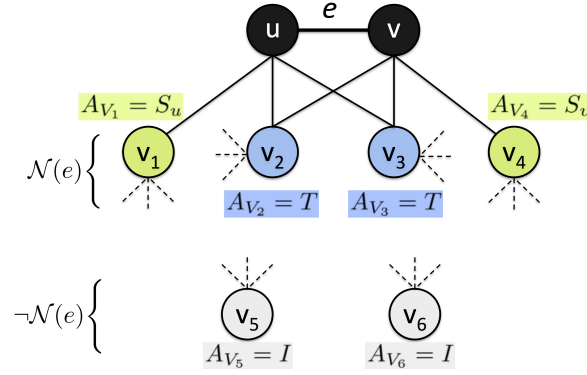


Fig. 6.2.: Illustration of edge neighborhood: Let T denotes the nodes forming triangles with edge (u, v) (i. e., V_2, V_3), whereas S_u and S_v denotes the nodes forming 2-stars centered at u and v respectively (i. e., V_1, V_4), and let I denote the nodes that are not connected to edge e (i. e., V_5, V_6). Further, the dotted lines represent edges incident to these nodes.

$$N_{a_i, a_j}^{(e)} = \left| \left\{ \{u, v, w, r\} \begin{array}{l} |w, r \in V \setminus \{u, v\} \\ \wedge w \neq r \\ \wedge A_w = a_i, A_r = a_j \end{array} \right\} \right| \quad (6.6)$$

Now that we defined the topological properties of nodes w and r relative to edge e , we need to define whether nodes w and r are connected themselves. Let e'_{wr} represent whether w and r are connected or not, such that $e'_{wr} = 1$ if $(w, r) \in E$ and $e'_{wr} = 0$ otherwise. Accordingly, let $N_{a_i, a_j, e'_{wr}}^{(e)}$ denotes the number of 4-node motifs $\{u, v, w, r\}$, where w, r satisfy property $a_i, a_j \in A$ respectively,

$$N_{a_i, a_j, e'_{wr}}^{(e)} = \left| \left\{ \{u, v, w, r\} \begin{array}{l} |w, r \in V \setminus \{u, v\} \\ \wedge w \neq r \\ \wedge A_w = a_i, A_r = a_j \\ \wedge e'_{wr} \in \{0, 1\} \end{array} \right\} \right| \quad (6.7)$$

For example, $N_{T, T, 1}^{(e)}$ is the number of all motifs $\{u, v, w, r\}$ containing edge e , where both w and r are forming triangles with e and there exist an edge between w and r . Using Equations 6.6 and 6.7, we provide a general principle for motif counting in the following theorem.

Theorem 6.5.1 *General Principle for Motif Counting:* Given a graph G , for any edge $e = (u, v)$ in G , and for any properties $a_i, a_j \in A$, the number of 4-node motifs $\{u, v, w, r\}$ satisfies the following rule,

$$N_{a_i, a_j, 0}^{(e)} = N_{a_i, a_j}^{(e)} - N_{a_i, a_j, 1}^{(e)} \quad (6.8)$$

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing edge e , where nodes w and r satisfy a_i, a_j properties respectively, and $(w, r) \in E$. Then the expression on the right side counts this subgraph once in the $N_{a_i, a_j}^{(e)}$ term, and once in the $N_{a_i, a_j, 1}^{(e)}$. By the principle of inclusion-exclusion [188], the total contribution of the subgraph $\{u, v, w, r\}$ in $N_{a_i, a_j, 0}^{(e)}$ is zero. Thus, $N_{a_i, a_j, 0}^{(e)}$ is the number of motifs having properties a_i, a_j , but $(w, r) \notin E$. ■

Clearly, it is sufficient to compute $N_{a_i, a_j}^{(e)}$ and $N_{a_i, a_j, 1}^{(e)}$ only, and use Theorem 6.5.1 to compute $N_{a_i, a_j, 0}^{(e)}$ in constant time. Note that $N_{a_i, a_j}^{(e)}$ is an unrestricted count and can be computed in constant time using the knowledge we have from finding 3-node motifs.

Now, to simplify the discussion in the following sections, we precisely show how to compute $N_{a_i, a_j}^{(e)}$, the number of 4-node motifs $\{u, v, w, r\}$ such that w, r satisfy property $a_i, a_j \in A$ respectively. Let \mathcal{W}_{a_i} be the set of nodes with property $a_i \in A$ (i. e., $\mathcal{W}_{a_i} = \{w \in V \setminus \{u, v\} \mid A_w = a_i, \forall a_i \in A\}$), and similarly \mathcal{R}_{a_j} be the set of nodes with property $a_j \in A$ (i. e., $\mathcal{R}_{a_j} = \{r \in V \setminus \{u, v\} \mid A_r = a_j, \forall a_j \in A\}$). If $a_i = a_j$, then $\mathcal{W}_{a_i} = \mathcal{R}_{a_j}$. Thus,

$$N_{a_i, a_i}^{(e)} = \binom{|\mathcal{W}_{a_i}|}{2} = \frac{1}{2} \cdot (|\mathcal{W}_{a_i}| - 1) \cdot |\mathcal{W}_{a_i}| \quad (6.9)$$

However, if $a_i \neq a_j$, then \mathcal{W}_{a_i} and \mathcal{R}_{a_j} are mutually exclusive (i. e., $\mathcal{W}_{a_i} \cap \mathcal{R}_{a_j} = \emptyset$). Thus,

$$N_{a_i, a_j}^{(e)} = |\mathcal{W}_{a_i}| \cdot |\mathcal{R}_{a_j}| \quad (6.10)$$

6.5.3 Analysis & Combinatorial Arguments

In this section, we discuss combinatorial arguments involving unrestricted counts that can be computed directly from our knowledge of 3-node motifs. These combinatorial arguments capture the relationships between the counts of pairs of 4-node motifs. The proofs of these relationships are based on Theorem 6.5.1 and the transition diagram in Fig. 6.1. For each pair of motifs g_{4_i} and g_{4_j} , we show the relationship for each edge in the graph (in Corollary 1–14), then we show a generalization for the whole graph (in Lemma 6.5.1–6.5.7).

Relationship between 4-Cliques & 4-ChordalCycles. Here, our goal is to show the relationship between the total number of 4-clique motifs (g_{4_1}) and the total number of 4-chordalcycle motifs (g_{4_2}). We start by showing the relationship for each edge in the graph (in Corollary 1 and 2), then we show a generalization for the whole graph (in Lemma 6.5.1).

Corollary 1 *For any edge $e = (u, v)$ in the graph, the number of 4-cliques containing e is $N_{T,T,1}^{(e)}$.*

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e . From graph theory, we know that any clique of size k contains k distinct cliques of size $k - 1$ [183]. Accordingly, $\{u, v, w, r\}$ is a 4-clique if and only if it contains all triangles in the set $\{(u, v, w), (u, v, r), (u, w, r), (v, w, r)\}$, which means $A_w = T$, $A_r = T$, and $e'_{wr} = 1$, as there is an edge between w and r . More generally, any subgraph $\{u, v, w, r\}$ containing e contributes once in the count $N_{T,T,1}^{(e)}$ if and only if it is a 4-clique. In Theorem 6.5.1, we showed that $N_{T,T,1}^{(e)} \leq N_{T,T}^{(e)}$. ■

Corollary 2 *For any edge $e = (u, v)$ in the graph, the number of 4-chordalcycles, where e is the chord edge of the cycle (denoted by the black color in Fig. 6.1), is $N_{T,T,0}^{(e)}$.*

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e . We say that $\{u, v, w, r\}$ is a 4-chordal-cycle with chord e if and only if there exist two nodes $w, r \in \text{Tri}_e$, and $(w, r) \notin E$. Clearly, if $\{u, v, w, r\}$ is a 4-chordal-cycle with chord e , then it contains two triangles (u, v, w) and (u, v, r) overlapping in e . This means $A_w = T$ and $A_r = T$ and $e'_{wr} = 0$, as there is no edge between w and r . More generally, any subgraph $\{u, v, w, r\}$ contributes once in the count $N_{T,T,0}^{(e)}$ if and only if it is a 4-chordal-cycle with e as the chord. In Theorem 6.5.1, we showed that $N_{T,T,0}^{(e)} \leq N_{T,T}^{(e)}$. ■

Lemma 6.5.1 *For any graph G , the relationship between the counts of 4-cliques (i. e., $f(g_{4_1}, G)$) and 4-chordal-cycles (i. e., $f(g_{4_2}, G)$) is,*

$$f(g_{4_2}, G) = \sum_{e \in E} \binom{|\text{Tri}_e|}{2} - 6 \cdot f(g_{4_1}, G)$$

Proof From Theorem 6.5.1 and the addition principle [188], the total count for all edges in G is,

$$\sum_{e \in E} N_{T,T,0}^{(e)} = \sum_{e \in E} N_{T,T}^{(e)} - \sum_{e \in E} N_{T,T,1}^{(e)} \quad (6.11)$$

Given that $N_{T,T}^{(e)}$ is the number of 4-node subgraphs $\{u, v, w, r\}$ containing e , such that $A_w = T, A_r = T$. Thus, from Eq.6.9, $N_{T,T}^{(e)} = \binom{|\text{Tri}_e|}{2}$. Now, from Corollary 1, each 4-clique will be counted 6 times (once for each edge in the clique). Thus, the total count of 4-cliques in G is $f(g_{4_1}, G) = \frac{1}{6} \cdot \sum_{e \in E} N_{T,T,1}^{(e)}$. Similarly, from Corollary 2, each 4-chordal-cycle is counted only once for each chord edge. Thus, the total count of 4-chordal-cycles in G is $f(g_{4_2}, G) = \sum_{e \in E} N_{T,T,0}^{(e)}$. By direct substitution in Eq.6.11, this lemma is true. ■

Relationship between 4-Cycles & 4-Paths. Our goal is to show the relationship between the total number of 4-cycle motifs (g_{4_4}) and the total number of 4-path motifs (g_{4_6}). We start by showing the relationship for each edge in the graph (in Corollary 3 and 4), then we show a generalization for the whole graph (in Lemma 6.5.2).

Corollary 3 For any edge $e = (u, v)$ in the graph, the number of 4-cycles containing e is $N_{S_u, S_v, 1}^{(e)}$.

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e , with some nodes $w, r \in \mathcal{N}(e)$. From graph theory, we know that any cycle of size k nodes has exactly k edges, and every node has exactly degree 2 [183] (i. e., each node is a center of 2-star). Accordingly, $\{u, v, w, r\}$ is a 4-cycle if and only if it contains all the 2-star subgraphs in the set $\{(v, u, w), (u, v, r), (v, r, w), (u, w, r)\}$, such that each 2-star is centered around one of the four nodes. This means $A_w = S_u$ and $A_r = S_v$ and $e'_{wr} = 1$, as there is an edge between w and r . More generally, any subgraph $\{u, v, w, r\}$ containing e contributes once in the count $N_{S_u, S_v, 1}^{(e)}$ if and only if it is a cycle of 4 nodes. In Theorem 6.5.1, we showed that $N_{S_u, S_v, 1}^{(e)} \leq N_{S_u, S_v}^{(e)}$. ■

Corollary 4 For any edge $e = (u, v)$ in the graph, the number of 4-paths containing e , where e is the middle edge in the path (denoted by the green color in Fig. 6.1), is $N_{S_u, S_v, 0}^{(e)}$.

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e , with some nodes $w, r \in \mathcal{N}(e)$. From graph theory, we know that a 4-path is a connected chain of 3 edges, with every node has degree at least 1 and at most 2. We say that $\{u, v, w, r\}$ is a 4-path containing e as a middle edge if and only if it contains the edges in the set $\{(w, u), (u, v), (v, r)\}$, where (w, u) is the start of the path, (v, r) is the end of the path, and $e = (u, v)$ is the middle edge. This means $A_w = S_u$ and $A_r = S_v$ and $e'_{wr} = 0$, as there is no edge between w and r . More generally, any subgraph $\{u, v, w, r\}$ containing e contributes once in the count $N_{S_u, S_v, 0}^{(e)}$ if and only if it is a path of 4 nodes with e is the middle edge. In Theorem 6.5.1, we showed that $N_{S_u, S_v, 0}^{(e)} \leq N_{S_u, S_v}^{(e)}$. ■

Lemma 6.5.2 For any graph G , the relationship between the counts of 4-cycles (i. e., $f(g_{4_4}, G)$) and 4-paths (i. e., $f(g_{4_6}, G)$) is,

$$f(g_{4_6}, G) = \sum_{e \in E} |\text{Star}_u| \cdot |\text{Star}_v| - 4 \cdot f(g_{4_4}, G)$$

Proof From Theorem 6.5.1 and the addition principle [188], the total count for all edges in G is,

$$\sum_{e \in E} N_{S_u, S_v, 0}^{(e)} = \sum_{e \in E} N_{S_u, S_v}^{(e)} - \sum_{e \in E} N_{S_u, S_v, 1}^{(e)} \quad (6.12)$$

Given that $N_{S_u, S_v}^{(e)}$ is the number of 4-node subgraphs $\{u, v, w, r\}$ containing e , such that $w, r \in A_w = S_u, A_r = S_v$. Thus, from Eq.6.10, $N_{S_u, S_v}^{(e)} = |\text{Star}_u| \cdot |\text{Star}_v|$. Now, from Corollary 3, each 4-cycle will be counted 4 times (once for each edge in the cycle). Thus, the total count of 4-cycles in G is $f(g_{4_4}, G) = \frac{1}{4} \cdot \sum_{e \in E} N_{S_u, S_v, 1}^{(e)}$. Similarly, from Corollary 4, each 4-path is counted only once for each middle edge in the path. Thus, the total count of 4-paths in G is $f(g_{4_6}, G) = \sum_{e \in E} N_{S_u, S_v, 0}^{(e)}$. By direct substitution in Eq.6.12, this lemma is true. \blacksquare

Relationship between 4-TailedTriangles & 4-ChordalCycles. Our goal is to show the relationship between the total number of 4-tailedtriangle motifs (g_{4_3}) and the total number of 4-chordalcycle motifs (g_{4_2}). We start by showing the relationship for each edge in the graph (in Corollary 5 and 6), then we show a generalization for the whole graph (in Lemma 6.5.3).

Corollary 5 *For any edge $e = (u, v)$ in the graph, the number of 4-tailedtriangles where e is part of both the triangle and 2-star patterns (denoted by the blue color in Fig. 6.1), is $N_{T, S_u \vee S_v, 0}^{(e)}$.*

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e , with some nodes $w, r \in \mathcal{N}(e)$. From graph theory, we know that any tailed triangle of size 4 nodes contains a triangle, with one of the nodes in the triangle connected to the tail edge and forming the center of a 2-star [183]. Accordingly, $\{u, v, w, r\}$ is a tailed-triangle where e is part of both the triangle and 2-star patterns, if and only if there is a node $w \in \text{Tri}_e$, and another node $r \in \text{Star}_u$ or Star_v , such that w, r are not connected by an edge. This means $A_w = T$ and $A_r = S_u \vee S_v$ and $e'_{wr} = 0$, as there is no edge between w and r . More generally, any subgraph $\{u, v, w, r\}$ containing e contributes once in the count

$N_{T,S_u \vee S_v,0}^{(e)}$ if and only if it is a 4-tailedtriangle where e is part of both the triangle and 2-star patterns. In Theorem 6.5.1, we showed that $N_{T,S_u \vee S_v,0}^{(e)} \leq N_{T,S_u \vee S_v}^{(e)}$. ■

Corollary 6 For any edge $e = (u, v)$ in the graph, the number of 4-chordalcycles where e is a cycle edge (denoted by the blue color in Fig. 6.1), is $N_{T,S_u \vee S_v,1}^{(e)}$.

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e . We say that $\{u, v, w, r\}$ is a 4-chordalcycle with e is a cycle edge if and only if there exist two nodes w, r such that $w \in \text{Tri}_e$ and $r \in \text{Star}_u \vee \text{Star}_v$, and $(w, r) \in E$. This means $A_w = T$ and $A_r = S_u \vee S_v$ and $e'_{wr} = 1$. More generally, any subgraph $\{u, v, w, r\}$ containing e contributes once in the count $N_{T,S_u \vee S_v,1}^{(e)}$ if and only if it is a 4-chordalcycle where e is a cycle edge. From Theorem 6.5.1, we showed that $N_{T,S_u \vee S_v,1}^{(e)} \leq N_{T,S_u \vee S_v}^{(e)}$. ■

Lemma 6.5.3 For any graph G , the relationship between the counts of 4-chordalcycles (i. e., $f(g_{4_2}, G)$) and 4-tailedtriangles (i. e., $f(g_{4_3}, G)$) is,

$$2 \cdot f(g_{4_3}, G) = \sum_{e \in E} |\text{Tri}_e| \cdot (|\text{Star}_u| + |\text{Star}_v|) - 4 \cdot f(g_{4_2}, G)$$

Proof From Theorem 6.5.1 and the addition principle [188], the total count for all edges in G is,

$$\sum_{e \in E} N_{T,S_u \vee S_v,0}^{(e)} = \sum_{e \in E} N_{T,S_u \vee S_v}^{(e)} - \sum_{e \in E} N_{T,S_u \vee S_v,1}^{(e)} \quad (6.13)$$

Given that $N_{T,S_u \vee S_v}^{(e)} = N_{T,S_u}^{(e)} + N_{T,S_v}^{(e)}$ is the number of 4-node subgraphs $\{u, v, w, r\}$ containing e , such that $A_w = T, A_r = S_u \vee S_v$. Thus, from Eq.6.10, $N_{T,S_u \vee S_v}^{(e)} = |\text{Tri}_e| \cdot (|\text{Star}_u| + |\text{Star}_v|)$. Now, from Corollary 6, each 4-chordalcycle is counted 4 times (once for each edge in the cycle). Thus, the total count of 4-chordalcycle in G is $f(g_{4_2}, G) = \frac{1}{4} \cdot \sum_{e \in E} N_{T,S_u \vee S_v,1}^{(e)}$. Similarly, from Corollary 5, each 4-tailedtriangle will be counted 2 times (once for each blue edge as in Fig. 6.1). Thus, the total count of 4-tailedtriangle in G is $f(g_{4_3}, G) = \frac{1}{2} \cdot \sum_{e \in E} N_{T,S_u \vee S_v,0}^{(e)}$. By direct substitution in Eq.6.13, this lemma is true. ■

Relationship between 4-TailedTriangles & 3-Stars. Our goal is to show the relationship between the total number of 4-tailedtriangle motifs (g_{4_3}) and the total number of 3-stars (g_{4_2}). We start by showing the relationship for each edge in the graph (in Corollary 7 and 8), then we show a generalization for the whole graph (in Lemma 6.5.4).

Corollary 7 *For any edge $e = (u, v)$ in the graph, the number of 4-tailedtriangles with e as the tail edge (denoted by the green color in Fig. 6.1) and u is part of the triangle, is $N_{S_u, S_u, 1}^{(e)}$.*

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e . $\{u, v, w, r\}$ is a 4-tailedtriangle with e as the tail edge and u is part of the triangle, if and only if $w, r \in S_u$ and w, r are connected by an edge. This means $A_w = S_u$ and $A_r = S_u$ and $e'_{wr} = 1$. More generally, any subgraph $\{u, v, w, r\}$ containing e contributes once in the count $N_{S_u, S_u, 1}^{(e)}$ if and only if it is a 4-tailedtriangle with e as the tail edge and u is part of the triangle. In Theorem 6.5.1, we showed that $N_{S_u, S_u, 1}^{(e)} \leq N_{S_u, S_u}^{(e)}$. ■

In a similar fashion, the number of 4-tailedtriangles with e as the tail edge and v is part of the triangle is $N_{S_v, S_v, 1}^{(e)}$. Thus, the total number of 4-tailedtriangles with e as the tail edge and $u \vee v$ is part of the triangle is $N_{S, S, 1}^{(e)} = N_{S_u, S_u, 1}^{(e)} + N_{S_v, S_v, 1}^{(e)}$.

Corollary 8 *For any edge $e = (u, v)$ in the graph, the number of 3-star centered around u is $N_{S_u, S_u, 0}^{(e)}$.*

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e . $\{u, v, w, r\}$ is a 3-star centered around u , if and only if $w, r \in S_u$ and w, r are not connected by an edge. This means $A_w = S_u$ and $A_r = S_u$ and $e'_{wr} = 0$. More generally, any subgraph $\{u, v, w, r\}$ containing e contributes once in the count $N_{S_u, S_u, 0}^{(e)}$ if and only if it is a 3-star centered around u . In Theorem 6.5.1, we showed that $N_{S_u, S_u, 0}^{(e)} \leq N_{S_u, S_u}^{(e)}$. ■

Again, the number of 3-stars centered around v is $N_{S_v, S_v, 0}^{(e)}$. Thus, the total number of 3-stars centered around u or v is $N_{S, S, 0}^{(e)} = N_{S_u, S_u, 0}^{(e)} + N_{S_v, S_v, 0}^{(e)}$.

Lemma 6.5.4 *For any graph G , the relationship between the counts of 3-stars (i. e., $f(g_{4_5}, G)$) and 4-tailedtriangles (i. e., $f(g_{4_3}, G)$) is,*

$$3 \cdot f(g_{4_5}, G) = \sum_{e \in E} \binom{|\text{Star}_u|}{2} + \binom{|\text{Star}_v|}{2} - f(g_{4_3}, G)$$

Proof From Theorem 6.5.1 and the addition principle [188], the total count for all edges in G is,

$$\sum_{e \in E} N_{S.,S.,0}^{(e)} = \sum_{e \in E} N_{S.,S.}^{(e)} - \sum_{e \in E} N_{S.,S.,1}^{(e)} \quad (6.14)$$

Given that $N_{S.,S.}^{(e)} = N_{S_u,S_u}^{(e)} + N_{S_v,S_v}^{(e)}$ is the number of 4-node subgraphs $\{u, v, w, r\}$ containing e , such that $A_w = S_u \wedge A_r = S_u$ or $A_w = S_v \wedge A_r = S_v$. Thus, from Eq.6.9, $N_{S.,S.}^{(e)} = \binom{|\text{Star}_u|}{2} + \binom{|\text{Star}_v|}{2}$. Now, from Corollary 8, each 3-star is counted 3 times (once for each edge in the star). Thus, the total count of 3-stars in G is $f(g_{4_5}, G) = \frac{1}{3} \cdot \sum_{e \in E} N_{S.,S.,0}^{(e)}$. Similarly, from Corollary 7, each 4-tailedtriangle will be counted once for each tail edge (denoted by the green color in Fig. 6.1). Thus, the total count of 4-tailedtriangle in G is $f(g_{4_3}, G) = \sum_{e \in E} N_{S.,S.,1}^{(e)}$. This holds whether the patterns are centered around u or v . By direct substitution in Eq.6.14, this lemma is true. ■

Relationship between 4-TailedTriangles & 4-Node-1-Triangles. Our goal is to show the relationship between the total number of 4-tailedtriangle motifs (g_{4_3}) and the total number of 4-Node-1-triangles (g_{4_7}). We start by showing the relationship for each edge in the graph (in Corollary 9 and 10), then we show a generalization for the whole graph (in Lemma 6.5.5).

Corollary 9 *For any edge $e = (u, v)$ in the graph, the number of 4-node-1-triangle is $N_{T,I,0}^{(e)}$.*

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e , for some nodes w, r . $\{u, v, w, r\}$ is a 4-node-1-triangle if and only if there are some nodes w, r such that

$w \in \text{Tri}_e$, $r \notin \mathcal{N}(e)$, and $(w, r) \notin E$. This means r is independent of e , and w forms a triangle with e . As such, $A_w = T$ and $A_r = I$ and $e'_{wr} = 0$. More generally, any subgraph $\{u, v, w, r\}$ containing e contributes once in the count $N_{T,I,0}^{(e)}$ if and only if it is a 4-node-1-triangle. In Theorem 6.5.1, we showed that $N_{T,I,0}^{(e)} \leq N_{T,I}^{(e)}$. ■

Corollary 10 *For any edge $e = (u, v)$ in the graph, the number of 4-tailedtriangles with e participating in the triangle but not connected to the tail edge (denoted by the red color in Fig. 6.1), is $N_{T,I,1}^{(e)}$.*

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e . $\{u, v, w, r\}$ is a 4-tailedtriangle with e participating in the triangle but not connected to the tail edge, if and only if there are some nodes w, r such that $w \in \text{Tri}_e$, $r \notin \mathcal{N}(e)$, and $(w, r) \in E$. This means r is independent of e , and w forms a triangle with e . As such, $A_w = T$ and $A_r = I$ and $e'_{wr} = 1$. More generally, any subgraph $\{u, v, w, r\}$ containing e contributes once in the count $N_{T,I,1}^{(e)}$ if and only if it is a 4-tailedtriangle with e participating in the triangle but not connected to the tail edge. In Theorem 6.5.1, we showed that $N_{T,I,1}^{(e)} \leq N_{T,I}^{(e)}$. ■

Lemma 6.5.5 *For any graph G , the relationship between the counts of 4-tailedtriangles (i. e., $f(g_{4_3}, G)$) and 4-node-1-triangles (i. e., $f(g_{4_7}, G)$) is,*

$$3.f(g_{4_7}, G) = \sum_{e \in E} \left(\text{Tri}_e \cdot (|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|) \right) - f(g_{4_3}, G)$$

Proof From Theorem 6.5.1 and the addition principle [188], the total count for all edges in G is,

$$\sum_{e \in E} N_{T,I,0}^{(e)} = \sum_{e \in E} N_{T,I}^{(e)} - \sum_{e \in E} N_{T,I,1}^{(e)} \quad (6.15)$$

Given that $N_{T,I}^{(e)}$ is the number of 4-node subgraphs $\{u, v, w, r\}$ containing e , such that $A_w = T$, $A_r = I$. And, the number of nodes independent of e is $|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|$. Thus, from Eq.6.10, $N_{T,I}^{(e)} = \text{Tri}_e \cdot (|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|)$. Now, from Corollary 10, each 4-tailedtriangle is counted one time (once for the red edge as in Fig. 6.1).

Thus, the total count of 4-tailedtriangles in G is $f(g_{43}, G) = \sum_{e \in E} N_{T,I,1}^{(e)}$. Similarly, from Corollary 9, each 4-node-1-triangle will be counted 3 times (once for each edge in the triangle). Thus, the total count of 4-node-1-triangles in G is $f(g_{47}, G) = \frac{1}{3} \cdot \sum_{e \in E} N_{T,I,0}^{(e)}$. By direct substitution in Eq.6.15, this lemma is true. ■

Relationship between 4-Paths & 4-node-2-Stars. Our goal is to show the relationship between the total number of 4-path motifs (g_{46}) and the total number of 4-node-2-star motifs (g_{48}). We start by showing the relationship for each edge in the graph (in Corollary 11 and 12), then we show a generalization for the whole graph (in Lemma 6.5.6).

Corollary 11 *For any edge $e = (u, v)$ in the graph, the number of 4-paths where e is the start or end of the path (denoted by the purple color in Fig. 6.1), is $N_{S_u \vee S_v, I, 1}^{(e)}$.*

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e . $\{u, v, w, r\}$ is a 4-path with e is the start or end of the path, if and only if there there are some nodes w, r where $w \in \text{Star}_u \vee \text{Star}_v$, $r \notin \mathcal{N}(e)$, and $(w, r) \in E$. This means r is independent of e , and w forms a 2-star with e . As such, $A_w = S_u \vee S_v$ and $A_r = I$ and $e'_{wr} = 1$. More generally, any subgraph $\{u, v, w, r\}$ containing e contributes once in the count $N_{S_u \vee S_v, I, 1}^{(e)}$ if and only if it is a 4-path. In Theorem 6.5.1, we showed that $N_{S_u \vee S_v, I, 1}^{(e)} \leq N_{S_u \vee S_v, I}^{(e)}$. ■

Corollary 12 *For any edge $e = (u, v)$ in the graph, the number of 4-node-2-stars where e is one of the star edges (denoted by the purple color in Fig. 6.1), is $N_{S_u \vee S_v, I, 0}^{(e)}$.*

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e . $\{u, v, w, r\}$ is a 4-node-2-star with e as one of the star edges, if and only if there there are some nodes w, r where $w \in \text{Star}_u \vee \text{Star}_v$, $r \notin \mathcal{N}(e)$, and $(w, r) \notin E$. This means r is independent of e , and w forms a 2-star with e . As such, $A_w = S_u \vee S_v$ and $A_r = I$ and $e'_{wr} = 0$. More generally, any subgraph $\{u, v, w, r\}$ containing e contributes once in the count $N_{S_u \vee S_v, I, 0}^{(e)}$ if and only if it is a 4-node-2-star. In Theorem 6.5.1, we showed that $N_{S_u \vee S_v, I, 0}^{(e)} \leq N_{S_u \vee S_v, I}^{(e)}$. ■

Lemma 6.5.6 *For any graph G , the relationship between the counts of 4-paths (i. e., $f(g_{4_6}, G)$) and 4-node-2-stars (i. e., $f(g_{4_8}, G)$) is,*

$$2.f(g_{4_8}, G) = \sum_{e \in E} |\text{Star}_e|. (|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|) - 2.f(g_{4_6}, G)$$

Proof From Theorem 6.5.1 and the addition principle [188], the total count for all edges in G is,

$$\sum_{e \in E} N_{S_u \vee S_v, I, 0}^{(e)} = \sum_{e \in E} N_{S_u \vee S_v, I}^{(e)} - \sum_{e \in E} N_{S_u \vee S_v, I, 1}^{(e)} \quad (6.16)$$

Given that $N_{S_u \vee S_v, I}^{(e)} = N_{S_u, I}^{(e)} + N_{S_v, I}^{(e)}$ is the number of 4-node subgraphs $\{u, v, w, r\}$ containing e , such that $A_w = S_u \vee S_v, A_r = I$. And, the number of nodes independent of e is $|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|$. Thus, $N_{S_u \vee S_v, I}^{(e)} = |\text{Star}_e|. (|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|)$, such that $|\text{Star}_e| = |\text{Star}_u| + |\text{Star}_v|$ (from Eq.6.10). Now, from Corollary 11, each 4-path is counted 2 times (for both the start and end edges in the path, denoted by the purple in Fig. 6.1). Thus, the total count of 4-paths in G is $f(g_{4_6}, G) = \frac{1}{2} \cdot \sum_{e \in E} N_{S_u \vee S_v, I, 1}^{(e)}$. Similarly, from Corollary 12, each 4-node-2-star will be counted 2 times (once for each edge in the star, denoted by the purple in Fig. 6.1). Thus, the total count of 4-node-2-star in G is $f(g_{4_8}, G) = \frac{1}{2} \cdot \sum_{e \in E} N_{S_u \vee S_v, I, 0}^{(e)}$. By direct substitution in Eq.6.16, this lemma is true. ■

Relationship between 4-node-2-edges & 4-node-1-edge. Our goal is to show the relationship between the total number of 4-node-2-edge motifs (g_{4_9}) and the total number of 4-node-1-edge motifs ($g_{4_{10}}$). We start by showing the relationship for each edge in the graph (in Corollary 13 and 14), then we show a generalization for the whole graph (in Lemma 6.5.6).

Corollary 13 *For any edge $e = (u, v)$ in the graph, the number of 4-node-2-edges where e is any of the two independent edges in the motif (denoted by the orange color in Fig. 6.1), is $N_{I, I, 1}^{(e)}$.*

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e . $\{u, v, w, r\}$ is a 4-node-2-edge with e is one of the two independent edges, if and only if there there are some nodes $w, r \notin \mathcal{N}(e)$, and $(w, r) \in E$. This means w and r are independent of e . As such, $A_w = I$ and $A_r = I$ and $e'_{wr} = 1$. More generally, any subgraph $\{u, v, w, r\}$ containing e contributes once in the count $N_{I,I,1}^{(e)}$ if and only if it is a 4-node-2-edge. In Theorem 6.5.1, we showed that $N_{I,I,1}^{(e)} \leq N_{I,I}^{(e)}$. ■

Corollary 14 *For any edge $e = (u, v)$ in the graph, the number of 4-node-1-edge where e is an isolated/single edge in the motif (denoted by the orange color in Fig. 6.1), is $N_{I,I,0}^{(e)}$.*

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e . $\{u, v, w, r\}$ is a 4-node-1-edge with e is an isolated/single edge, if and only if there there are some nodes $w, r \notin \mathcal{N}(e)$, and $(w, r) \notin E$. This means w and r are independent of e . As such, $A_w = I$ and $A_r = I$ and $e'_{wr} = 0$. More generally, any subgraph $\{u, v, w, r\}$ containing e contributes once in the count $N_{I,I,0}^{(e)}$ if and only if it is a 4-node-1-edge. In Theorem 6.5.1, we showed that $N_{I,I,0}^{(e)} \leq N_{I,I}^{(e)}$. ■

Lemma 6.5.7 *For any graph G , the relationship between the counts of 4-node-2-edge motifs (i. e., $f(g_{4_9}, G)$) and 4-node-1-edge motifs (i. e., $f(g_{4_{10}}, G)$) is,*

$$f(g_{4_{10}}, G) = \sum_{e \in E} \left(\binom{|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|}{2} \right) - 2 \cdot f(g_{4_9}, G)$$

Proof From Theorem 6.5.1 and the addition principle [188], the total count for all edges in G is,

$$\sum_{e \in E} N_{I,I,0}^{(e)} = \sum_{e \in E} N_{I,I}^{(e)} - \sum_{e \in E} N_{I,I,1}^{(e)} \quad (6.17)$$

Given that $N_{I,I}^{(e)}$ is the number of 4-node subgraphs $\{u, v, w, r\}$ containing e , such that $A_w = I, A_r = I$. And, the number of nodes independent of e is $|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|$. Thus, from Eq.6.9, $N_{I,I}^{(e)} = \binom{|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|}{2}$. Now, from Corollary 13, each 4-node-2-edge is counted 2 times (for the two edges in the motif, denoted by the

orange in Fig. 6.1). Thus, the total count of 4-node-2-edges in G is $f(g_{49}, G) = \frac{1}{2} \cdot \sum_{e \in E} N_{I,I,1}^{(e)}$. Similarly, from Corollary 14, each 4-node-1-edge will be counted once (for the isolated/single edge in the motif, denoted by the orange in Fig. 6.1). Thus, the total count of 4-node-1-edge in G is $f(g_{410}, G) = \sum_{e \in E} N_{I,I,0}^{(e)}$. By direct substitution in Eq.6.17, this lemma is true. \blacksquare

While it is straightforward to compute $N_{I,I}^{(e)}$ for each edge e , this is not the case for $N_{I,I,1}^{(e)}$ or $N_{I,I,0}^{(e)}$, as they require searching outside the local edge neighborhood. However, since $N_{I,I,1}^{(e)}$ is the number of edges outside the egonet of e , it can be computed as follows,

$$\begin{aligned} N_{I,I,1}^{(e)} &= |E| - |\mathcal{N}(u) \setminus \{v\}| - |\mathcal{N}(v) \setminus \{u\}| - |\{e\}| \\ &\quad - [N_{T,T,1}^{(e)} + N_{T,S_u \vee S_v,1}^{(e)} + N_{T,I,1}^{(e)}] \\ &\quad - [N_{S,S,1}^{(e)} + N_{S_u, S_v,1}^{(e)} + N_{S,I,1}^{(e)}] \end{aligned}$$

Thus, the total number of 4-node-2-edges is,

$$\begin{aligned} 2.f(g_{49}, G) &= \sum_{e \in E} N_{I,I,1}^{(e)} \tag{6.18} \\ &= \sum_{e \in E} |E| - |\mathcal{N}(u) \setminus \{v\}| - |\mathcal{N}(v) \setminus \{u\}| - |\{e\}| \\ &\quad - [6.f(g_{41}, G) + 4.f(g_{42}, G) + 2.f(g_{43}, G)] \\ &\quad - [4.f(g_{44}, G) + 2.f(g_{46}, G)] \end{aligned}$$

Finally, the number of 4-node-independent motifs (g_{411}) is,

$$f(g_{411}, G) = \binom{|V|}{4} - \sum_{i=1}^{10} f(g_{4i}, G) \tag{6.19}$$

Algorithm 6.2: MOTIFCENSUS($G = (V, E)$) our exact motif census algorithm for counting all 3, 4-node motifs. Alg. takes an undirected graph and returns the frequencies of all 3, 4-node motifs

Input : Graph $G = (V, E)$
Output: Motif Counts of size 3, 4 nodes $f(\mathcal{G}_3, G)$

- 1 Initialize Array X
- 2 $N_{T,T} = 0, N_{S_u, S_v} = 0, N_{T, S_u \vee S_v} = 0, N_{S_{\cdot}, S_{\cdot}} = 0$
- 3 $N_{T,I} = 0, N_{S_u \vee S_v, I} = 0, N_{I,I} = 0, N_{I,I,1} = 0$
- 4 **for each** edge $e = (u, v) \in E$ **in parallel do**
- 5 $\text{Star}_u = \emptyset, \text{Star}_v = \emptyset, \text{Tri}_e = \emptyset$
- 6 **for each** $w \in \mathcal{N}(u)$ **do**
- 7 **if** $w = v$ **then continue**
- 8 Add w to Star_u and Set $X(w) = 1$
- 9 **for each** $w \in \mathcal{N}(v)$ **do**
- 10 **if** $w = u$ **then continue**
- 11 **if** $X(w) = 1$ **then** \rightarrow found triangle
- 12 Add w to Tri_e and Set $X(w) = 2$
- 13 Remove w from Star_u
- 14 **else** Add w to Star_v and Set $X(w) = 3$
- 15 Compute $f(\mathcal{G}_3, G)$ as in Lines 13–15 of Alg. 6.1
- 16 // Get Counts of 4-Cliques & 4-Cycles
- 17 $f(g_{4_1}, G) += \text{CLIQUECOUNT}(X, \text{Tri}_e)$
- 18 $f(g_{4_4}, G) += \text{CYCLECOUNT}(X, \text{Star}_u)$
- 19 // Get Unrestricted Counts for 4-Node Connected Motifs
- 20 $N_{T,T} += \binom{|\text{Tri}_e|}{2}$
- 21 $N_{S_u, S_v} += |\text{Star}_u| \cdot |\text{Star}_v|$
- 22 $N_{T, S_u \vee S_v} += |\text{Tri}_e| \cdot (|\text{Star}_u| + |\text{Star}_v|)$
- 23 $N_{S_u, S_u} = \binom{|\text{Star}_u|}{2}$ and $N_{S_v, S_v} = \binom{|\text{Star}_v|}{2}$
- 24 $N_{S_{\cdot}, S_{\cdot}} += N_{S_u, S_u} + N_{S_v, S_v}$
- 25 // Get Unrestricted Counts for 4-Node Disconnected Motifs
- 26 $N_{T,I} += |\text{Tri}_e| \cdot (|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|)$
- 27 $N_{S_u, I} = |\text{Star}_u| \cdot (|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|)$
- 28 $N_{S_v, I} = |\text{Star}_v| \cdot (|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|)$
- 29 $N_{S_u \vee S_v, I} += N_{S_u, I} + N_{S_v, I}$
- 30 $N_{I,I} += \binom{|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|}{2}$
- 31 $N_{I,I,1} += |E| - |\mathcal{N}(u) \setminus \{v\}| - |\mathcal{N}(v) \setminus \{u\}| - 1$
- 32 **for each** $w \in \mathcal{N}(v)$ **do** $X(w) = 0$
- 33 Use Lem. 6.5.1–6.5.5 to compute $f(g_{4_i}, G)$ for $i = 1 : 8$
- 34 Use Eq.6.18 to compute $f(g_{4_9}, G)$ and Lem. 6.5.7 for $f(g_{4_{10}}, G)$
- 35 Use Eq. 6.19 to compute $f(g_{4_{11}}, G)$
- 36 **return** $f(\mathcal{G}_3, G), f(\mathcal{G}_4, G)$

37 **procedure** CLIQUECOUNT(X, Tri_e)

- 38 cliq _{e} = 0
- 39 **for each** node $w \in \text{Tri}_e$ **do**
- 40 **for each** $r \in \mathcal{N}(w)$ **do**
- 41 **if** $X(r) = 2$ **then** \rightarrow found 4-Clique
- 42 cliq _{e} += 1
- 43
- 44 $X(w) = 0$
- 45 **return** cliq _{e}

46 **procedure** CYCLECOUNT(X, Star_u)

- 47 cyc _{e} = 0
- 48 **for each** node $w \in \text{Star}_u$ **do**
- 49 **for each** $r \in \mathcal{N}(w)$ **do**
- 50 **if** $X(r) = 3$ **then** \rightarrow found 4-Cycle
- 51 cyc _{e} += 1
- 52
- 53 $X(w) = 0$
- 54 **return** cyc _{e}

6.5.4 Algorithm

Alg. 6.2 shows how to count all motifs of size $k = \{3, 4\}$ nodes efficiently (using Lem. 6.5.1—6.5.7). As discussed previously, we start by finding all triangle and 2-star patterns in Lines 6–14 (i. e., Step 1). Then, in Lines 17–18 we only count 4-cliques and 4-cycles (i. e., Step 2). Then, Lines 20–31 compute unrestricted counts for all 4-node motifs in constant time (using knowledge from steps 1 and 2, as discussed in Step3), and finally Lines 33–35 compute the final counts (using the lemma proved in Section 6.5.3) (i. e., Step4). Lemmas 6.5.8 and 6.5.9 show the complexity of Alg. 6.2. Alg. 6.2 counts all 4-cliques and 4-cycles in $\mathcal{O}(m \cdot \Delta \cdot T_{max})$ and $\mathcal{O}(m \cdot \Delta \cdot S_{max})$ respectively, where T_{max} is the maximum number of triangles incident to an edge and $T_{max} \ll \Delta$ for sparse graphs, and S_{max} is the maximum number of stars incident to an edge and $S_{max} \leq \Delta$. This is more efficient than $\mathcal{O}(|V| \cdot \Delta^3)$ given by [162], and $\mathcal{O}(\Delta \cdot |E| + |E|^2)$ given by [179]. In Section 6.6, we empirically compare to these methods on a variety of data sets and we show the efficiency of Alg. 6.2.

Lemma 6.5.8 *Alg. 6.2 counts all 4-cliques in $\mathcal{O}(|E| \cdot \Delta \cdot T_{max})$, where T_{max} is the maximum number of triangles incident to an edge.*

Proof For each edge $e = (u, v) \in E$, the runtime complexity of counting all 4-cliques incident to e is equivalent to finding the set of all edges $e' = (w, w')$ such that $\{e' = (w, w') \in E \mid w, w' \in \text{Tri}_e \wedge w \neq w'\}$, where Tri_e is the set of triangles incident to e . First, we show in Lem. 6.4.1 that the runtime complexity of finding all triangles incident to e is $\mathcal{O}(\Delta)$. Second, as described in Alg. 6.2 the runtime complexity of checking whether any two distinct nodes $w, w' \in \text{Tri}_e$ are connected by an edge $e' = (w, w')$ is $\mathcal{O}(\sum_{w \in \text{Tri}_e} \Delta) = \mathcal{O}(|\text{Tri}_e| \cdot \Delta)$, and can be computed asymptotically $\mathcal{O}(T_{max} \cdot \Delta)$, where T_{max} is the maximum triangle degree (i.e., the maximum number of triangles incident to an edge and $T_{max} \ll \Delta$). Therefore, the total runtime complexity is $\mathcal{O}\left(\sum_{e \in E} (\Delta + T_{max} \cdot \Delta)\right) = \mathcal{O}(|E| \cdot \Delta \cdot T_{max})$. ■

Lemma 6.5.9 *Alg. 6.2 counts all 4-cycles of size $k = 4$ in $\mathcal{O}(|E|\cdot\Delta\cdot S_{max})$, where S_{max} is the maximum number of 2-stars incident to an edge (proof is similar to Lem. 6.5.8).*

Proof For each edge $e = (u, v) \in E$, the runtime complexity of counting all 4-cycles incident to e is equivalent to finding the set of all edges $e' = (w, w')$ such that $\{e' = (w, w') \in E \mid w \in \text{Star}_u \wedge w' \in \text{Star}_v, w \neq w'\}$. First, we show in Lem. 6.4.1 that the runtime complexity of finding all 2-star patterns incident to e is $\mathcal{O}(\Delta)$. Second, Alg. 6.2 shows the runtime complexity of checking whether any two distinct nodes $w \in \text{Star}_u$, and $w' \in \text{Star}_v$ are connected by an edge $e' = (w, w')$ is $\mathcal{O}\left(\sum_{w \in \text{Star}_u} \Delta\right) = \mathcal{O}(|\text{Star}_u| \cdot \Delta)$, and is asymptotically $\mathcal{O}(S_{max} \cdot \Delta)$ (where S_{max} is the maximum number of 2-stars incident to an edge, and $S_{max} \leq \Delta$). Therefore, the total runtime complexity is $\mathcal{O}\left(\sum_{e \in E} (\Delta + S_{max} \cdot \Delta)\right) = \mathcal{O}(|E| \cdot \Delta \cdot S_{max})$. ■

6.6 Experiments & Applications

We systematically investigate the scalability of our proposed algorithm on a large collection of 1646 networks from a variety of domains with millions of nodes and edges (see [178] for data download). Tables 6.6–6.8 show the statistics of 200 networks from a variety of categories, including social, biological and other domains. In addition, we test our algorithms with large collections of networks. Specifically, Tables 6.4 and 6.5 show the statistics of 100 Facebook college networks [153]. In addition, Tables 6.11 and 6.12 show the statistics of a subset of 50 protein graphs (from the D&D collection of 1178 protein graphs) and 50 chemical compound graphs (from the MUTAG collection of 188 chemical compound graphs) [176]. Moreover, Tables 6.9 and 6.10 show the statistics of 80 dense graphs from the DIMACS challenge ¹.

We proceed by first demonstrating how fast the parallel motif census algorithm (Alg.6.2) counts all motifs of size $k = \{3, 4\}$ (both connected and disconnected motifs) on various social and information networks. We systematically investigate the

¹<http://dimacs.rutgers.edu/Challenges/>

scalability of our algorithm on all the above networks and we show detailed results for a representative subset of 55 networks categorized in 8 broad classes from social, Facebook, biological, web, technological, co-authorship, infrastructure, among other domains. Note that for all of the networks, we discard edge weights, self-loops, and edge direction. To the best of our knowledge, this is the largest study for motif counting, and these are the largest motif computations published to date. Our own implementation of Alg. 6.2 uses shared memory, but the algorithm is well-suited for shared-memory multi-core architectures (CPU and GPU) and distributed architectures (MPI). We used a two processor, Intel Xeon system with 6 cores and 48GB of memory.

We also discuss a number of applications that could benefit from our fast motif counting algorithm (Alg. 6.2), which facilitates exploring and understanding networks and their structure. Motifs could provide an intuitive and meaningful characterization of a network at the global macro-level as well as the local micro-level, thus, they are useful for numerous applications. At the macro-level, motifs are useful for finding similar networks (similarity queries), or finding networks that disagree most with that set (graph anomalies), or exploring a time-series of networks, among numerous other possibilities. Alternatively, motifs are also extremely useful for characterizing networks and their behavior at the local node/edge-level as known as the micro-level. For instance, given an edge $(u, v) \in E$, find the top-k most similar edges (with applications in role discovery, entity-resolution, link prediction, and other related matching/similarity applications). Also, motifs could be used for ranking nodes/edges to find unique patterns and anomalies such as large stars, cliques, etc.

6.6.1 Scalability & Runtime

Table 6.2 shows the runtime and statistics of a representative subset of 55 networks. For each network, we provide the counts of connected motifs of size $k = \{3, 4\}$ and the time (seconds) taken to count all motifs. Notably, Alg. 6.2 takes only few

Table 6.2.: Runtime & Statistics for a Subset of 55 Networks

graph	V	E	g _{3₁}	g _{3₂}	g _{4₁}	g _{4₂}	g _{4₄}	g _{4₆}	g _{4₅}	g _{4₃}	Seconds	
											Alg.6.2	RAGE
soc-brightkite	57k	213k	494k	12M	2.9M	12M	2.7M	533M	1.3B	114M	0.2	273.03
socfb-Berkeley13	23k	852k	5.4M	125M	27M	153M	87M	17B	25B	2.7B	4.94	2514.59
socfb-Wisconsin87	24k	836k	4.9M	107M	23M	121M	59M	12B	21B	1.9B	3.93	1450.31
socfb-FSU53	28k	1.0M	7.9M	130M	63M	242M	95M	16B	10B	2.9B	5.55	2192.94
socfb-MSU24	32k	1.1M	6.5M	139M	33M	183M	106M	16B	32B	2.6B	5.67	1904.09
socfb-Texas80	32k	1.2M	9.6M	160M	68M	316M	122M	21B	11B	3.9B	7.53	2967.01
socfb-Michigan23	30k	1.2M	8.3M	162M	49M	277M	146M	23B	13B	3.5B	7.57	2995.83
socfb-Indiana69	30k	1.3M	9.4M	181M	60M	269M	141M	25B	13B	3.8B	8.44	3212.10
socfb-Ullinois20	31k	1.3M	9.4M	172M	64M	273M	130M	23B	27B	3.8B	7.88	3088.77
socfb-UF21	35k	1.5M	12M	266M	98M	433M	186M	40B	150B	7.2B	14.49	N/A
soc-flickr	514k	3.2M	59M	963M	1.7B	14B	6.7B	244B	326B	90B	182.57	N/A
soc-orkut	3.1M	117M	628M	44B	3.2B	48B	70B	19T	98T	1.5T	14448.6	N/A
bio-celegans	453	2.0k	3.3k	69k	3.0k	37k	4.5k	495k	2.9M	363k	<0.001	1.7
bio-diseasome	516	1.2k	1.4k	5.4k	1.4k	923	42	18k	27k	19k	<0.001	0.44
bio-dmela	7.4k	26k	2.9k	572k	393	13k	107k	11M	9.2M	312k	0.01	2.47
bio-yeast-protein-inter	1.8k	2.2k	222	11k	41	198	140	31k	72k	2.6k	<0.001	0.53
bio-yeast	1.5k	1.9k	206	11k	39	195	139	31k	72k	2.5k	<0.001	0.43
bio-human-gene2	14k	9.0M	4.9B	10B	2.3T	3.7T	90B	4.4T	5.3T	8.4T	8023.84	N/A
bio-mouse-gene	43k	14M	3.6B	15B	670B	2.1T	223B	9.0T	6.7T	7.7T	5515.6	N/A
ca-CSphd	1.9k	1.7k	8	6.6k	0	5	8	9.4k	32k	93	<0.001	1.25
ca-GrQc	4.2k	13k	48k	85k	329k	66k	1.1k	553k	406k	628k	<0.001	5.99
ca-dblp-2012	317k	1.0M	2.2M	15M	17M	4.8M	203k	252M	259M	97M	0.48	227.79
ca-cit-HepTh	23k	2.4M	191M	1.6B	13B	47B	7.3B	538B	976B	385B	132.66	N/A
ca-cit-HepPh	28k	3.1M	196M	1.5B	9.8B	34B	6.1B	536B	479B	276B	125.49	N/A
ca-coauthors-dblp	540k	15M	444M	698M	15B	3.4B	31M	42B	27B	67B	40.26	N/A
ca-hollywood-2009	1.1M	56M	4.9B	33B	1.4T	635B	168B	21T	17T	8.9T	13799.6	N/A
tech-as-caida2007	26k	53k	36k	15M	54k	1.7M	407k	285M	7.8B	47M	0.19	36.83
tech-p2p-gnutella	63k	148k	2.0k	1.6M	16	826	42k	15M	8.1M	71k	0.02	7.44
tech-RL-caida	191k	608k	455k	21M	423k	7.4M	40M	583M	1.7B	77M	0.39	71.74
tech-WHOIS	7.5k	57k	782k	5.3M	12M	31M	2.9M	229M	566M	194M	0.14	44.52
tech-as-skitter	1.7M	11M	29M	16B	149M	20B	43B	819B	96T	162B	476.06	N/A
web-BerkStan-dir	685k	6.6M	65M	28B	1.1B	99B	25B	49B	382T	476B	149.17	N/A
web-edu	3.0k	6.5k	10k	81k	40k	4.6k	18	435k	1.3M	186k	<0.001	0.52
web-google-dir	876k	4.3M	13M	687M	40M	382M	38M	4.1B	650B	6.7B	4.45	N/A
web-indochina-2004	11k	48k	210k	481k	1.2M	88k	9.2k	5.5M	12M	4.9M	0.01	24.36
web-it-2004	509k	7.2M	339M	56M	29B	815M	175M	1.1B	1.4B	527M	25.26	N/A
web-baidu-baike	2.1M	17M	25M	31B	28M	4.5B	9.2B	3.3T	571T	327B	3975.81	N/A
web-wikipedia-growth	1.9M	37M	127M	123B	288M	38B	68B	29T	3.1P	3.2T	22389.2	N/A
web-ClueWeb09-50m	148M	447M	1.2B	494B	5.6B	243B	774B	34T	24P	3.4T	91697.4	N/A
inf-italy-osm	6.7M	7.0M	7.4k	8.2M	0	244	47k	9.9M	992k	27k	0.85	N/A
inf-openflights	2.9k	16k	73k	639k	286k	1.5M	319k	17M	17M	9.0M	0.01	2.46
inf-power	4.9k	6.6k	651	17k	90	385	324	38k	20k	5.1k	<0.001	0.58
inf-roadNet-CA	2.0M	2.8M	120k	5.6M	40	13k	249k	11M	2.4M	521k	0.35	N/A
inf-roadNet-PA	1.1M	1.5M	67k	3.2M	16	5.7k	152k	6.2M	1.4M	295k	0.19	N/A
inf-road-usa	24M	29M	439k	50M	90	21k	1.6M	81M	18M	1.5M	4.05	N/A
ia-email-EU-dir	265k	364k	267k	194M	581k	10M	6.7M	4.4B	221B	341M	1.52	887.18
ia-enron-only	143	623	889	4.8k	779	2.7k	648	29k	17k	14k	<0.001	0.12
ia-reality	6.8k	7.7k	400	497k	63	1.7k	2.8k	1.6M	26M	93k	<0.001	1.39
ia-wiki-Talk-dir	2.4M	4.7M	9.2M	13B	65M	1.0B	924M	1.2T	192T	64B	281.33	N/A
ia-wikiquote-user-edits	93k	238k	279k	636M	411k	70M	44M	8.9B	2.4T	2.5B	2.41	691.28
ia-wiki-user-edits-page	2.1M	5.6M	6.7M	550B	10M	70B	44B	4.8T	88P	2.0T	5691.92	N/A
brock200-3	200	12k	291k	570k	3.2M	12M	4.1M	11M	3.5M	16M	0.02	22.96
brock200-4	200	13k	373k	584k	5.2M	16M	4.3M	8.9M	3.0M	17M	0.02	21.85
brock400-3	400	60k	4.4M	4.5M	184M	372M	63M	84M	28M	251M	0.4	997.15
brock400-4	400	60k	4.4M	4.5M	185M	373M	63M	84M	28M	250M	0.4	1010.26

N/A: Alg. was timed out after 30 hours of runtime

seconds to count all motifs for large social, web, and technological graphs (among others). For example, for a large road network (i. e., inf-road-usa) with 24M nodes and 29M edges, Alg. 6.2 takes only 4 seconds to count all motifs. Also as shown in Table 6.2, for large Facebook networks with nearly 2M edges, Alg. 6.2 takes only 15 seconds, and for large web graphs with nearly 8M edges, Alg. 6.2 takes only 25 seconds.

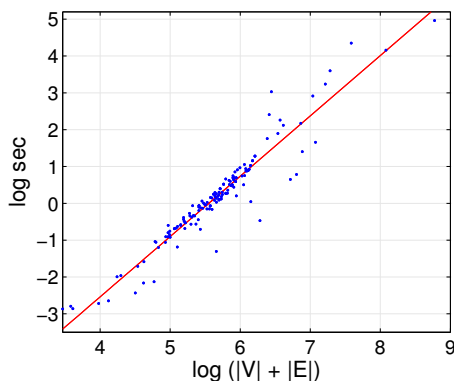


Fig. 6.3.: Runtime of exact motif counting (Alg.6.2) for social and information networks scales almost linearly with the network dimension.

We compare the empirical runtime of Alg. 6.2 to the state-of-the-art baseline method RAGE [179]. For social and Facebook networks, we observed that Alg. 6.2 is on average 460x faster than RAGE. For all other networks, we observed that Alg. 6.2 is on average 600x faster than RAGE. Notably, Alg. 6.2 takes only 7 seconds to count motifs of Facebook networks with 1.3M edges, while RAGE takes almost an hour for the same networks. For larger networks with millions of nodes/edges, RAGE was timed out (as it did not finish within 30 hours of runtime). Moreover, for dense graphs from the DIMACS challenge, RAGE takes almost 17 minutes, while Alg. 6.2 takes less than a second. We also compared to the baseline method FANMOD [180] and Orca [181], we found that for a Facebook network with 250k edges, FANMOD takes roughly 2.5 hours for counting all motifs, RAGE takes almost 7 minutes for the same network, and Orca takes almost 10 seconds, while Alg. 6.2 takes less than a second.

Note that both RAGE and Orca count only connected motifs, while our algorithm and FANMOD count both connected and disconnected motifs.

Finally, in Figure 6.3, we plot the runtime of Alg. 6.2 for a representative subset of 150 social and information networks. The figure shows that our algorithm exhibits nearly linear-time scaling over social and information networks ranging from 1000 nodes to 100 million nodes.

6.6.2 Large-Scale Graph Comparison & Classification

Motifs are also useful for large-scale comparison and classification of graphs. In this case, we relax the notion of equivalence and isomorphism to some form of *structural similarity* between graphs, such that the graph similarity is measured using feature-based motif counts. We study the full data set of Facebook100, which contains 100 Facebook college networks collected from a variety of US schools [153]. We plot the graphlet frequency distribution (GFD) score pictorially in Figure 6.4 for all California schools. The GFD score is simply the normalized frequencies of graphlets (motifs) of size k [161]. In our case, we use $k = 4$. The figure shows Caltech noticeably different than others, consistent with the discussion in [153] which shows how Caltech is well-known to be organized almost exclusively according to its undergraduate “Housing” residence system, in contrast to other schools that follow the “dormitory” residence system. The residence system seems to impact the organization of the community structures at Caltech.

Moreover, we use counts of motifs of size $k = \{2, 3, 4\}$ -nodes as features, from which we learn a model to predict the label of the unlabeled graphs (e. g., the function of proteins). We test our approach on protein graphs (D&D collection of 1178 protein graphs) and chemical compound graphs (MUTAG collection of 188 chemical compound graphs) [176]. We extract the motif features using our fast Alg. 6.2. Then, we learn a model using SVM (RBF kernel) to predict the labels of the unlabeled graphs, and we use 10-fold validation for evaluation. Table 6.3 shows the accuracy

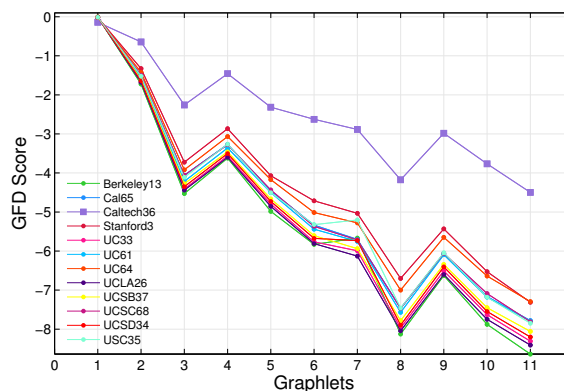


Fig. 6.4.: Anomaly detection in Facebook university networks. Using the space of motifs (graphlets) of size $k = 4$, Caltech is noticeably different than other California universities, which is consistent with the findings in [153].

Table 6.3.: Accuracy & Standard Error for Classification of Large Collection of Biological & Chemical Graphs

graph	Type	No. Graphs	Accuracy(%)	Alg. 6.2 (Time in Secs.)
D&D	Protein	1178	76.13 ± 0.03	1.05 secs
MUTAG	Chemicals	188	86.4 ± 0.21	0.14 secs

of this approach is 76% for protein function prediction, and 86% for mutagenic effect prediction. Note that by using all motif-based features up to size 4 nodes, we were able to obtain better accuracy than previous work (previous work achieved maximum 75% and 83% for D&D and MUTAG respectively [162]). More importantly, Alg. 6.2 extracts all the features (motif counts up to size 4 nodes) in almost one second. This yields a significant improvement over the motif extraction method in [162], which takes 2.45 hours to extract the same features from the D&D graph collection.

6.6.3 Finding Large Stars, Cliques, and Other Patterns

How can we quickly and efficiently find large cliques, stars, and other unique patterns? Further, how can we identify the top-k largest cliques, stars, etc? Note that many of these problems are NP-hard, e. g., finding the clique of maximum size

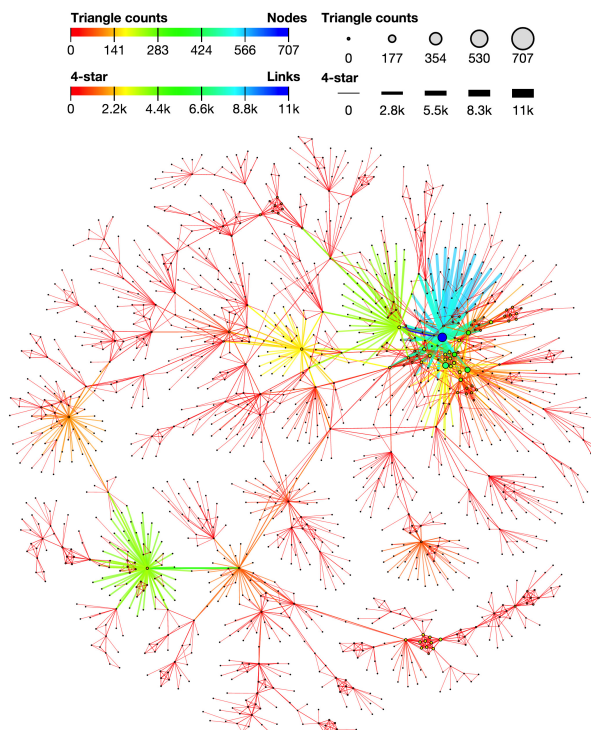


Fig. 6.5.: Visualization of the human disease network: A network of disorders and disease genes linked by known disorder-gene associations [189]. Edges are weighted/-colored by their number of incident star motifs of size 4 nodes, nodes are weighted/-colored by their triangle counts. The large star on the right denoted by light blue color corresponds to colon cancer; the large star on the lower left denoted by lime green color corresponds to deafness; and the large star on the right denoted by lime green color corresponds to leukemia. Notably this figure highlights the few phenotypes (such as colon cancer, leukemia, and deafness) correspond to hubs (large stars) that are connected to a large number of distinct disorders, which is consistent with the findings in [189].

is a well-known NP-hard problem [183]. To answer these and other related queries, we leverage the proposed parallel motif counting method in Alg. 6.2.

The idea is clearly shown in Figure. 6.5. Figure 6.5 provides a visualization of the human disease network [189], where we used Alg. 6.2 to rank (weight) all the edges in the network by the number of star patterns of size 4 nodes [57]. The intuition behind the method is that if an edge (or node) has a (relatively) large number of stars of 4 nodes (cliques, or another motif of interest), then it is also likely to be part of

a star of a large size. Recall that removing a node from a k -star or k -clique forms a star or clique of size $k - 1$ [183]. Accordingly, edges with large weights are likely to be members of large stars. Thus, as shown in Figure. 6.5, a visualization based on the parallel motif counting method can help to quickly highlight such large stars by using the counts (of stars of size 4 nodes) as edge weights or colors. Note that the same approach is also applicable for finding cliques and other interesting patterns, since edges with a high number of 4-cliques are likely to be members of the largest clique in the network.

6.7 Summary

In this chapter, we proposed a fast, efficient, and parallel algorithm for counting motifs of size $k = \{3, 4\}$ -nodes that take only a fraction of the time to compute when compared with the current methods used. The proposed motif counting algorithm leverages a number of proven combinatorial arguments for different motifs. For each edge, we count a few motifs, and with these counts along with the combinatorial arguments, we obtain the exact counts of others in constant time. We systematically investigate the scalability of our proposed algorithm on a large collection of 300+ networks from a variety of domains with millions of nodes and edges. The experiments show that our motif counting strategies are on average 460x faster than current methods. We also test and discuss new opportunities to investigate the use of motifs on much larger networks and newer applications than the state-of-the-art. For example, for finding large stars and cliques, as well as top-k queries. To the best of our knowledge, this chapter provides the largest motif computations to date. A summary of the main contributions of this chapter are:

- **Algorithms.** A fast, efficient, and parallel motif counting algorithm that leverages a number of combinatorial arguments that we show for different motifs. The combinatorial arguments we show in this chapter enable us to obtain significant improvement on the scalability of motif counting.

- **Scalability.** The proposed motif counting algorithm achieves on average 460 times faster runtime compared to the state-of-the-art methods. In addition, we analyze motif counts on graphs of sizes that are beyond the scope of the state-of-the-art (e. g., on graphs with roughly 150 million nodes and 0.5 billion edges).
- **Effectiveness.** Largest motif computations to date and largest systematic evaluation on over 300+ large-scale networks from a variety of domains, from biological networks to social and information.
- **Applications.** We show a variety of applications for motif counting, such as finding unique patterns in graphs, as well as graph similarity and classification.

Table 6.4.: Statistics of Facebook100 Networks

graph	$ V $	$ E $	$ g_{3_1} $	$ g_{3_2} $	$ g_{4_1} $	$ g_{4_2} $	$ g_{4_4} $	$ g_{4_6} $	$ g_{4_5} $	$ g_{4_3} $
socfb-American75	6.4k	218k	1.5M	23M	6.5M	36M	23M	2.2B	1.2B	439M
socfb-Amherst41	2.2k	91k	916k	9.0M	4.5M	33M	18M	754M	361M	242M
socfb-Auburn71	18k	974k	10M	192M	85M	438M	229M	33B	53B	6.8B
socfb-Baylor93	13k	680k	7.0M	114M	55M	303M	181M	18B	11B	3.5B
socfb-BC17	12k	487k	3.5M	63M	16M	117M	77M	7.5B	4.2B	1.4B
socfb-Berkeley13	23k	852k	5.4M	125M	27M	153M	87M	17B	25B	2.7B
socfb-Bingham82	10k	363k	2.4M	38M	11M	63M	34M	3.8B	1.7B	691M
socfb-Bowdoin47	2.3k	84k	710k	7.7M	2.8M	22M	13M	599M	301M	178M
socfb-Brandeis99	3.9k	138k	1.0M	16M	3.7M	33M	20M	1.3B	1.8B	397M
socfb-Brown11	8.6k	385k	3.0M	52M	12M	96M	72M	6.7B	3.5B	1.2B
socfb-BU10	20k	638k	3.1M	67M	12M	61M	38M	6.9B	4.6B	974M
socfb-Bucknell39	3.8k	159k	1.3M	16M	5.9M	38M	23M	1.4B	620M	352M
socfb-Cal65	11k	351k	2.1M	33M	11M	46M	23M	3.1B	1.3B	573M
socfb-Caltech36	769	17k	120k	873k	460k	2.5M	965k	34M	19M	15M
socfb-Carnegie49	6.6k	250k	2.3M	30M	14M	84M	46M	3.5B	1.9B	838M
socfb-Colgate88	3.5k	155k	1.4M	16M	6.7M	46M	27M	1.5B	715M	399M
socfb-Columbia2	12k	444k	3.3M	66M	14M	119M	76M	8.2B	12B	1.8B
socfb-Cornell5	19k	791k	6.1M	117M	36M	206M	112M	16B	18B	2.9B
socfb-Dartmouth6	7.7k	304k	2.3M	38M	8.8M	72M	53M	4.4B	2.3B	833M
socfb-Duke14	9.9k	506k	5.1M	78M	31M	206M	126M	11B	5.9B	2.2B
socfb-Emory27	7.5k	330k	3.2M	41M	19M	112M	61M	4.6B	2.2B	1.0B
socfb-FSU53	28k	1.0M	7.9M	130M	63M	242M	95M	16B	10B	2.9B
socfb-Georgetown15	9.4k	426k	3.4M	58M	14M	111M	79M	7.3B	3.9B	1.4B
socfb-GWU54	12k	470k	3.2M	60M	15M	90M	56M	7.1B	5.6B	1.3B
socfb-Hamilton46	2.3k	96k	913k	9.8M	4.1M	31M	18M	826M	421M	251M
socfb-Harvard1	15k	825k	8.3M	158M	41M	378M	319M	29B	13B	4.9B
socfb-Haverford76	1.4k	60k	628k	5.6M	3.1M	24M	13M	427M	195M	153M
socfb-Howard90	4.0k	205k	2.2M	31M	12M	107M	87M	4.0B	2.2B	1.0B
socfb-Indiana69	30k	1.3M	9.4M	181M	60M	269M	141M	25B	13B	3.8B
socfb-JMU79	14k	486k	2.7M	54M	13M	63M	36M	5.4B	8.8B	1.0B
socfb-JohnsHopkins55	5.2k	187k	1.6M	21M	9.1M	54M	30M	2.1B	1.1B	518M
socfb-Lehigh96	5.1k	198k	1.6M	21M	8.1M	48M	28M	1.9B	1.0B	469M
socfb-Maine59	9.1k	243k	1.1M	20M	3.5M	21M	13M	1.6B	920M	281M
socfb-Maryland58	21k	745k	4.7M	94M	25M	117M	64M	11B	16B	1.8B
socfb-Mich67	3.7k	82k	468k	5.8M	1.9M	11M	5.0M	383M	206M	100M
socfb-Michigan23	30k	1.2M	8.3M	162M	49M	277M	146M	23B	13B	3.5B
socfb-Middlebury45	3.1k	125k	1.1M	13M	5.1M	35M	20M	1.1B	504M	301M
socfb-Mississippi66	11k	611k	8.3M	111M	75M	461M	252M	19B	9.8B	4.5B
socfb-MIT8	6.4k	251k	2.4M	32M	14M	88M	51M	3.8B	1.9B	909M
socfb-MSU24	32k	1.1M	6.5M	139M	33M	183M	106M	16B	32B	2.6B
socfb-MU78	15k	649k	4.6M	78M	27M	116M	59M	9.3B	4.0B	1.6B
socfb-Northeastern19	14k	382k	1.7M	34M	5.4M	32M	19M	3.0B	1.7B	455M
socfb-Northwestern25	11k	488k	4.4M	69M	25M	146M	85M	9.1B	6.2B	1.8B
socfb-NotreDame57	12k	541k	3.5M	73M	12M	98M	75M	9.4B	5.5B	1.4B
socfb-NYU9	22k	716k	3.6M	90M	13M	90M	59M	11B	11B	1.5B
socfb-Oberlin44	2.9k	90k	556k	7.9M	1.6M	14M	9.7M	618M	325M	143M
socfb-Oklahoma97	17k	893k	10M	163M	97M	482M	247M	29B	20B	5.7B
socfb-Penn94	42k	1.4M	7.2M	199M	31M	188M	113M	27B	57B	3.4B
socfb-Pepperdine86	3.4k	152k	1.6M	18M	9.2M	62M	37M	1.9B	915M	523M
socfb-Princeton12	6.6k	293k	2.5M	39M	11M	88M	63M	4.7B	2.1B	952M

Table 6.5.: Statistics of Facebook100 Networks (Table 6.4 continued)

graph	$ V $	$ E $	$ g_{3_1} $	$ g_{3_2} $	$ g_{4_1} $	$ g_{4_2} $	$ g_{4_4} $	$ g_{4_6} $	$ g_{4_5} $	$ g_{4_3} $
socfb-Reed98	962	19k	97k	1.0M	233k	2.0M	1.2M	44M	25M	15M
socfb-Rice31	4.1k	185k	1.9M	22M	11M	69M	36M	2.4B	1.1B	633M
socfb-Rochester38	4.6k	161k	1.3M	16M	6.4M	36M	19M	1.3B	873M	346M
socfb-Rutgers89	25k	785k	4.5M	83M	20M	101M	46M	8.8B	5.3B	1.4B
socfb-Santa74	3.6k	152k	1.5M	17M	7.6M	53M	31M	1.7B	914M	463M
socfb-Simmons81	1.5k	33k	169k	1.9M	457k	3.4M	1.7M	82M	48M	26M
socfb-Smith60	3.0k	97k	635k	8.0M	2.4M	13M	7.5M	597M	266M	139M
socfb-Stanford3	12k	568k	5.8M	94M	37M	226M	151M	15B	7.0B	2.8B
socfb-Swarthmore42	1.7k	61k	553k	5.7M	2.3M	19M	11M	423M	224M	143M
socfb-Syracuse56	14k	544k	4.4M	64M	31M	121M	58M	7.2B	4.1B	1.4B
socfb-Temple83	14k	361k	1.9M	38M	6.6M	50M	37M	3.8B	3.2B	649M
socfb-Tennessee95	17k	771k	7.2M	134M	52M	286M	164M	20B	34B	4.2B
socfb-Texas80	32k	1.2M	9.6M	160M	68M	316M	122M	21B	11B	3.9B
socfb-Texas84	36k	1.6M	11M	302M	71M	377M	215M	51B	119B	7.6B
socfb-Trinity100	2.6k	112k	1.1M	11M	5.3M	36M	20M	960M	409M	277M
socfb-Tufts18	6.7k	250k	1.8M	28M	8.6M	55M	33M	3.0B	1.4B	595M
socfb-Tulane29	7.8k	284k	2.4M	31M	16M	76M	35M	3.2B	1.6B	741M
socfb-UC33	17k	522k	3.2M	55M	16M	78M	36M	5.7B	3.4B	1.0B
socfb-UC61	14k	442k	3.5M	49M	24M	98M	40M	5.4B	2.6B	1.2B
socfb-UC64	6.8k	155k	938k	12M	4.6M	21M	9.1M	879M	479M	204M
socfb-UCF52	15k	429k	3.4M	54M	29M	102M	38M	5.3B	20B	1.4B
socfb-UChicago30	6.6k	208k	1.4M	22M	5.3M	35M	22M	2.1B	1.8B	456M
socfb-UCLA26	20k	748k	5.1M	92M	29M	131M	66M	11B	5.4B	1.8B
socfb-UConn91	17k	605k	3.4M	67M	17M	87M	44M	7.4B	4.4B	1.1B
socfb-UCSB37	15k	482k	3.1M	50M	18M	74M	34M	5.1B	2.4B	942M
socfb-UCSC68	9.0k	225k	1.1M	16M	4.2M	22M	9.3M	1.2B	527M	237M
socfb-UCSD34	15k	443k	2.7M	45M	13M	59M	26M	4.4B	3.9B	820M
socfb-UF21	35k	1.5M	12M	266M	98M	433M	186M	40B	150B	7.2B
socfb-UGA50	24k	1.2M	10M	179M	73M	341M	159M	27B	17B	4.6B
socfb-Ullinois20	31k	1.3M	9.4M	172M	64M	273M	130M	23B	27B	3.8B
socfb-UMass92	17k	519k	2.6M	57M	10M	60M	36M	5.4B	11B	958M
socfb-UNC28	18k	767k	5.4M	125M	30M	198M	132M	18B	26B	3.0B
socfb-UPenn7	15k	687k	5.5M	98M	28M	171M	110M	14B	6.9B	2.3B
socfb-USC35	17k	802k	7.2M	128M	55M	256M	133M	18B	24B	3.5B
socfb-USF51	13k	321k	1.8M	31M	9.5M	48M	29M	2.9B	1.7B	549M
socfb-USFCA72	2.7k	65k	372k	4.7M	1.2M	8.4M	5.1M	305M	156M	78M
socfb-UVA16	17k	789k	6.2M	120M	36M	211M	123M	17B	16B	2.9B
socfb-Vanderbilt48	8.1k	428k	4.7M	64M	33M	186M	108M	8.7B	5.8B	2.0B
socfb-Vassar85	3.1k	119k	850k	12M	2.7M	24M	19M	1.1B	485M	244M
socfb-Vermont70	7.3k	191k	947k	16M	3.4M	20M	11M	1.2B	701M	240M
socfb-Villanova62	7.8k	315k	2.5M	37M	11M	73M	46M	4.0B	2.2B	843M
socfb-Virginia63	21k	698k	4.5M	115M	27M	138M	69M	13B	79B	2.9B
socfb-Wake73	5.4k	279k	3.3M	39M	25M	137M	70M	4.9B	2.7B	1.3B
socfb-WashU32	7.8k	368k	3.6M	51M	21M	124M	74M	6.5B	4.1B	1.5B
socfb-Wellesley22	3.0k	95k	607k	8.7M	1.8M	16M	12M	710M	381M	168M
socfb-Wesleyan43	3.6k	138k	1.1M	14M	4.6M	32M	20M	1.2B	532M	291M
socfb-William77	6.5k	266k	2.1M	32M	11M	63M	39M	3.6B	2.2B	746M
socfb-Williams40	2.8k	113k	1.0M	11M	4.3M	32M	19M	999M	499M	282M
socfb-Wisconsin87	24k	836k	4.9M	107M	23M	121M	59M	12B	21B	1.9B
socfb-Yale4	8.6k	405k	3.6M	60M	17M	127M	87M	8.0B	5.9B	1.7B

Table 6.6.: Statistics of Biological, Co-authorship & Interaction Networks

graph	$ V $	$ E $	$ g_{3_1} $	$ g_{3_2} $	$ g_{4_1} $	$ g_{4_2} $	$ g_{4_4} $	$ g_{4_6} $	$ g_{4_5} $	$ g_{4_3} $
bio-celegans-dir	453	2.0k	3.3k	69k	3.0k	37k	4.5k	495k	2.9M	363k
bio-celegans	453	2.0k	3.3k	69k	3.0k	37k	4.5k	495k	2.9M	363k
bio-diseasome	516	1.2k	1.4k	5.4k	1.4k	923	42	18k	27k	19k
bio-dmela	7.4k	26k	2.9k	572k	393	13k	107k	11M	9.2M	312k
bio-yeast-protein-inter	1.8k	2.2k	222	11k	41	198	140	31k	72k	2.6k
bio-yeast	1.5k	1.9k	206	11k	39	195	139	31k	72k	2.5k
bio-human-gene1	22k	12M	6.8B	15B	3.3T	6.1T	131B	5.7T	9.8T	14T
bio-human-gene2	14k	9.0M	4.9B	10B	2.3T	3.7T	90B	4.4T	5.3T	8.4T
bio-mouse-gene	43k	14M	3.6B	15B	670B	2.1T	223B	9.0T	6.7T	7.7T
ca-AstroPh	18k	197k	1.4M	8.7M	9.6M	15M	1.3M	420M	299M	178M
ca-cit-HepPh	28k	3.1M	196M	1.5B	9.8B	34B	6.1B	536B	479B	276B
ca-cit-HepTh	23k	2.4M	191M	1.6B	13B	47B	7.3B	538B	976B	385B
ca-citeeater	227k	814k	2.7M	9.7M	19M	7.7M	83k	91M	609M	79M
ca-CondMat	21k	91k	171k	1.4M	289k	585k	38k	26M	26M	8.9M
ca-CSphd	1.9k	1.7k	8	6.6k	0	5	8	9.4k	32k	93
ca-dblp-2010	226k	716k	1.6M	7.7M	8.4M	3.2M	96k	99M	97M	50M
ca-dblp-2012	317k	1.0M	2.2M	15M	17M	4.8M	203k	252M	259M	97M
ca-Erdos992	6.1k	7.5k	1.6k	110k	450	5.1k	1.9k	664k	1.1M	89k
ca-GrQc	4.2k	13k	48k	85k	329k	66k	1.1k	553k	406k	628k
ca-HepPh	11k	118k	3.4M	5.2M	150M	35M	821k	204M	143M	462M
ca-IMDB	896k	3.8M	4.4k	162M	0	5.1k	23M	5.7B	9.1B	1.4M
ca-MathSciNet	333k	821k	577k	11M	407k	1.3M	256k	167M	169M	28M
ca-netscience	379	913	920	3.6k	631	894	7	8.5k	13k	8.7k
ca-sandi-auths	86	123	41	336	7	10	1	578	553	262
ca-coauthors-dblp	540k	15M	444M	698M	15B	3.4B	31M	42B	27B	67B
ca-hollywood-2009	1.1M	56M	4.9B	33B	1.4T	635B	168B	21T	17T	8.9T
ia-dbpedia-team-bi	365k	780k	3.7k	102M	0	97k	9.1M	1.3B	26B	4.9M
ia-email-EU-dir	265k	364k	267k	194M	581k	10M	6.7M	4.4B	221B	341M
ia-email-EU	32k	54k	49k	5.3M	67k	786k	403k	162M	461M	20M
ia-email-univ	1.1k	5.5k	5.3k	80k	3.4k	21k	13k	1.1M	546k	217k
ia-enron-email-dynamic	87k	297k	1.2M	46M	5.5M	51M	25M	2.9B	9.0B	654M
ia-enron-large	34k	181k	725k	23M	2.3M	22M	6.8M	1.4B	4.5B	376M
ia-enron-only	143	623	889	4.8k	779	2.7k	648	29k	17k	14k
ia-escorts-dynamic	10k	39k	2.5k	1.2M	54	11k	239k	28M	43M	593k
ia-fb-messages	1.3k	6.5k	2.5k	163k	255	11k	54k	3.4M	2.4M	248k
ia-infect-dublin	410	2.8k	7.1k	28k	14k	31k	7.4k	254k	100k	128k
ia-infect-hyper	113	2.2k	17k	52k	70k	279k	69k	548k	338k	634k
ia-radoslaw-email	167	3.2k	37k	95k	263k	767k	68k	668k	1.3M	1.6M
ia-reality	6.8k	7.7k	400	497k	63	1.7k	2.8k	1.6M	26M	93k
ia-southernwomen	18	64	97	194	73	173	25	265	189	441
ia-wiki-Talk-dir	2.4M	4.7M	9.2M	13B	65M	1.0B	924M	1.2T	192T	64B
ia-wiki-Talk	92k	361k	836k	51M	2.2M	32M	34M	5.8B	6.5B	668M
ia-wiki-trust-dir	139k	716k	3.0M	227M	12M	165M	56M	23B	303B	5.2B
ia-wikiquote-user-edits	93k	238k	279k	636M	411k	70M	44M	8.9B	2.4T	2.5B
ia-wiki-user-edits-page	2.1M	5.6M	6.7M	550B	10M	70B	44B	4.8T	88P	2.0T

Table 6.7.: Statistics of Infrastructure, Strong Components & Social Networks

graph	$ V $	$ E $	$ g_{3_1} $	$ g_{3_2} $	$ g_{4_1} $	$ g_{4_2} $	$ g_{4_4} $	$ g_{4_6} $	$ g_{4_5} $	$ g_{4_3} $
inf-euroroad	1.2k	1.4k	32	2.7k	0	3	38	5.4k	1.8k	218
inf-italy-osm	6.7M	7.0M	7.4k	8.2M	0	244	47k	9.9M	992k	27k
inf-openflights	2.9k	16k	73k	639k	286k	1.5M	319k	17M	17M	9.0M
inf-power	4.9k	6.6k	651	17k	90	385	324	38k	20k	5.1k
inf-roadNet-CA	2.0M	2.8M	120k	5.6M	40	13k	249k	11M	2.4M	521k
inf-roadNet-PA	1.1M	1.5M	67k	3.2M	16	5.7k	152k	6.2M	1.4M	295k
inf-USAir97	332	2.1k	12k	56k	61k	152k	4.5k	413k	972k	667k
inf-road-usa	24M	29M	439k	50M	90	21k	1.6M	81M	18M	1.5M
scc-enron-only	151	9.8k	425k	58k	13M	2.7M	34	66k	68k	1.2M
scc-fb-forum	897	71k	7.3M	3.3M	544M	292M	251k	21M	64M	253M
scc-fb-messages	1.9k	532k	138M	91M	26B	20B	971k	2.3B	6.8B	20B
scc-infect-dublin	11k	176k	2.3M	2.4M	26M	23M	83k	29M	12M	64M
scc-infect-hyper	113	6.2k	224k	9.5k	5.9M	423k	0	2.0k	2.1k	94k
scc-reality	6.8k	4.7M	2.2B	7.0B	742B	2.5T	20M	471B	7.1T	3.3T
scc-retweet-crawl	1.1M	24k	24k	176k	41k	115k	14k	2.0M	2.8M	997k
scc-retweet	18k	66k	3.6M	6.5M	155M	226M	1.6M	234M	330M	489M
scc-rt-lolgop	9.7k	4.5k	58k	185k	563k	1.3M	329	918k	5.7M	3.6M
scc-twitter-copen	8.6k	474k	97M	125M	15B	18B	10M	3.3B	19B	27B
soc-brightkite	57k	213k	494k	12M	2.9M	12M	2.7M	533M	1.3B	114M
soc-epinions	27k	100k	160k	4.9M	305k	2.7M	2.0M	222M	240M	35M
soc-flickr	514k	3.2M	59M	963M	1.7B	14B	6.7B	244B	326B	90B
soc-gowalla	197k	950k	2.3M	284M	6.1M	86M	42M	15B	784B	3.1B
soc-slashdot	70k	359k	402k	45M	1.8M	13M	15M	3.4B	11B	255M
soc-twitter-follows	405k	713k	30k	148M	2.8k	865k	15M	2.9B	21B	20M
soc-wiki-Vote	889	2.9k	2.1k	44k	836	11k	6.9k	467k	514k	124k
soc-youtube-snap	1.1M	3.0M	3.1M	1.5B	5.0M	222M	232M	91B	5.7T	12B
soc-youtube	496k	1.9M	2.4M	825M	3.8M	155M	162M	50B	3.2T	7.6B
soc-orkut-dir	3.1M	117M	628M	44B	3.2B	48B	70B	19T	98T	1.5T

Table 6.8.: Statistics of Technological, Retweet, & Web Networks

graph	$ V $	$ E $	$ g_{3_1} $	$ g_{3_2} $	$ g_{4_1} $	$ g_{4_2} $	$ g_{4_4} $	$ g_{4_6} $	$ g_{4_5} $	$ g_{4_3} $
tech-as-caida2007	26k	53k	36k	15M	54k	1.7M	407k	285M	7.8B	47M
tech-as-skitter	1.7M	11M	29M	16B	149M	20B	43B	819B	96T	162B
tech-internet-as	40k	85k	63k	28M	85k	5.1M	1.5M	643M	19B	106M
tech-p2p-gnutella	63k	148k	2.0k	1.6M	16	826	42k	15M	8.1M	71k
tech-pgp	11k	24k	55k	270k	239k	274k	22k	2.7M	4.0M	2.0M
tech-RL-caida	191k	608k	455k	21M	423k	7.4M	40M	583M	1.7B	77M
tech-routers-rf	2.1k	6.6k	10k	106k	21k	78k	23k	1.1M	1.2M	474k
tech-WHOIS	7.5k	57k	782k	5.3M	12M	31M	2.9M	229M	566M	194M
rt-retweet-crawl	1.1M	2.3M	175k	156M	29k	965k	10M	7.3B	64B	55M
rt-retweet	96	117	12	449	1	5	14	1.1k	1.1k	181
rt-twitter-copen	761	1.0k	149	7.0k	11	179	183	35k	38k	4.3k
web-arabic-2005	164k	1.7M	22M	3.6M	232M	3.4M	79k	27M	490M	26M
web-baidu-baike-related	416k	2.4M	14M	65B	329M	21B	213B	8.4B	2.7P	2.8B
web-BerkStan-dir	685k	6.6M	65M	28B	1.1B	99B	25B	49B	382T	476B
web-BerkStan	12k	20k	10k	78k	26k	16k	553	189k	354k	57k
web-edu	3.0k	6.5k	10k	81k	40k	4.6k	18	435k	1.3M	186k
web-EPA	4.3k	8.9k	997	240k	47	2.1k	20k	1.7M	7.3M	68k
web-frwikinews-user-edits	25k	69k	22k	148M	3.5k	5.5M	38M	842M	385B	142M
web-google-dir	876k	4.3M	13M	687M	40M	382M	38M	4.1B	650B	6.7B
web-google	1.3k	2.8k	5.1k	13k	13k	2.5k	116	38k	100k	27k
web-hudong	2.0M	14M	22M	19B	702M	1.3B	3.6B	1.6T	118T	88B
web-indochina-2004	11k	48k	210k	481k	1.2M	88k	9.2k	5.5M	12M	4.9M
web-it-2004	509k	7.2M	339M	56M	29B	815M	175M	1.1B	1.4B	527M
web-italycnr-2000	326k	3.1M	26M	7.9B	173M	62B	38B	22B	42T	44B
web-NotreDame	326k	1.1M	8.9M	278M	232M	161M	28M	1.3B	447B	5.9B
web-polblogs	643	2.3k	3.0k	47k	3.4k	20k	8.5k	350k	1.0M	149k
web-sk-2005	121k	334k	993k	3.3M	8.9M	919k	418k	16M	141M	11M
web-spam	4.8k	37k	129k	2.3M	374k	3.1M	1.5M	99M	114M	31M
web-Stanford	282k	2.3M	14M	3.9B	84M	13B	4.5B	29B	25T	43B
web-uk-2005	130k	12M	838M	712k	52B	0	0	109M	122M	24M
web-webbase-2001	16k	26k	21k	2.5M	80k	235k	20k	4.0M	895M	2.5M
web-wiki-ch-internal	1.9M	9.0M	18M	7.2B	30M	2.1B	4.2B	1.3T	31T	131B
web-wikipedia2009	1.9M	4.5M	2.2M	131M	1.5M	30M	64M	4.3B	10B	400M
web-baidu-baike	2.1M	17M	25M	31B	28M	4.5B	9.2B	3.3T	571T	327B
web-wikipedia-growth	1.9M	37M	127M	123B	288M	38B	68B	29T	3.1P	3.2T
web-ClueWeb09-50m	148M	447M	1.2B	494B	5.6B	243B	774B	34T	24P	3.4T

Table 6.9.: Statistics of DIMACS Networks

graph	$ V $	$ E $	$ g_{3_1} $	$ g_{3_2} $	$ g_{4_1} $	$ g_{4_2} $	$ g_{4_4} $	$ g_{4_6} $	$ g_{4_5} $	$ g_{4_3} $
brock200-1	200	15k	544k	558k	11M	23M	3.9M	5.3M	1.8M	16M
brock200-2	200	9.9k	160k	490k	950k	5.9M	3.0M	12M	4.1M	12M
brock200-3	200	12k	291k	570k	3.2M	12M	4.1M	11M	3.5M	16M
brock200-4	200	13k	373k	584k	5.2M	16M	4.3M	8.9M	3.0M	17M
brock400-1	400	60k	4.4M	4.5M	184M	373M	63M	84M	28M	250M
brock400-2	400	60k	4.5M	4.5M	185M	374M	63M	84M	28M	250M
brock400-3	400	60k	4.4M	4.5M	184M	372M	63M	84M	28M	251M
brock400-4	400	60k	4.4M	4.5M	185M	373M	63M	84M	28M	250M
brock800-1	800	208k	23M	38M	1.3B	4.1B	1.1B	2.4B	801M	4.4B
brock800-2	800	208k	23M	38M	1.3B	4.2B	1.1B	2.4B	794M	4.4B
brock800-3	800	207k	23M	38M	1.3B	4.1B	1.1B	2.4B	802M	4.4B
brock800-4	800	208k	23M	38M	1.3B	4.1B	1.1B	2.4B	799M	4.4B
c-fat200-1	200	1.5k	5.4k	6.0k	12k	14k	0	33k	0	27k
c-fat200-2	200	3.2k	26k	25k	132k	125k	0	275k	0	250k
c-fat200-5	200	8.5k	182k	163k	2.6M	2.3M	0	4.7M	0	4.5M
c-fat500-1	500	4.5k	19k	20k	48k	53k	0	125k	0	106k
c-fat500-10	500	47k	2.2M	2.0M	73M	60M	0	122M	0	120M
c-fat500-2	500	9.1k	82k	78k	487k	453k	0	984k	0	906k
c-fat500-5	500	23k	547k	488k	8.7M	7.4M	0	15M	0	15M
C1000-9	1k	450k	122M	40M	22B	15B	802M	352M	118M	3.2B
C125-9	125	7.0k	231k	78k	5.1M	3.4M	191k	91k	30k	789k
C2000-5	2k	1.0M	167M	499M	10B	62B	31B	125B	42B	125B
C2000-9	2k	1.8M	971M	323M	354B	235B	13B	5.8B	1.9B	52B
C250-9	250	28k	1.9M	630k	84M	57M	3.2M	1.4M	483k	13M
C4000-5	4k	4.0M	1.3B	4.0B	167B	1.0T	500B	2.0T	666B	2.0T
C500-9	500	112k	15M	5.0M	1.4B	909M	50M	22M	7.3M	201M
DSJC1000-5	1k	250k	21M	62M	649M	3.9B	1.9B	7.8B	2.6B	7.8B
DSJC500-5	500	63k	2.6M	7.8M	41M	245M	122M	483M	161M	486M
gen200-p0-9-44	200	18k	958k	318k	34M	23M	1.3M	562k	239k	5.0M
gen200-p0-9-55	200	18k	958k	318k	34M	23M	1.3M	575k	203k	5.1M
gen400-p0-9-55	400	72k	7.7M	2.6M	560M	370M	20M	9.2M	3.8M	82M
gen400-p0-9-65	400	72k	7.7M	2.6M	560M	369M	20M	9.4M	3.3M	83M
gen400-p0-9-75	400	72k	7.7M	2.6M	561M	369M	20M	9.4M	3.3M	84M
hamming10-2	1.0k	519k	173M	5.1M	43B	2.6B	13M	369k	0	46M
hamming10-4	1.0k	434k	101M	66M	14B	20B	2.3B	1.0B	1.2B	6.4B
hamming6-2	64	1.8k	31k	10k	342k	228k	13k	3.8k	0	46k
hamming6-4	64	704	960	12k	240	5.8k	24k	110k	54k	32k
hamming8-2	256	32k	2.5M	246k	144M	28M	470k	43k	0	1.7M
hamming8-4	256	21k	672k	1.4M	9.1M	43M	18M	20M	16M	43M

Table 6.10.: Statistics of DIMACS Networks (Table 6.9 continued)

graph	$ V $	$ E $	$ g_{3_1} $	$ g_{3_2} $	$ g_{4_1} $	$ g_{4_2} $	$ g_{4_4} $	$ g_{4_6} $	$ g_{4_5} $	$ g_{4_3} $
johnson16-2-4	120	5.5k	120k	131k	1.4M	3.6M	721k	262k	524k	1.4M
johnson32-2-4	496	108k	14M	6.0M	1.1B	1.1B	82M	12M	56M	163M
johnson8-2-4	28	210	420	1.7k	105	2.5k	2.5k	3.4k	2.2k	5.0k
johnson8-4-4	70	1.9k	24k	25k	162k	383k	73k	40k	43k	181k
keller4	171	9.4k	217k	387k	2.2M	8.9M	2.9M	4.2M	2.5M	8.5M
keller5	776	226k	33M	34M	2.6B	5.5B	999M	1.0B	572M	3.3B
keller6	3.4k	4.6M	3.5B	2.3B	1.6T	2.2T	251B	184B	99B	903B
MANN-a27	378	71k	8.7M	258k	789M	47M	234k	8.8k	44k	1.0M
MANN-a45	1.0k	533k	182M	2.0M	46B	1.0B	1.9M	43k	340k	12M
MANN-a81	3.3k	5.5M	6.1B	21M	5.0T	35B	21M	256k	3.6M	228M
MANN-a9	45	918	11k	2.8k	92k	49k	2.1k	252	468	4.6k
p-hat1000-1	1k	122k	3.1M	23M	20M	265M	282M	3.0B	1.2B	1.3B
p-hat1000-2	1k	245k	25M	56M	1.3B	4.6B	1.2B	5.5B	2.7B	7.5B
p-hat1000-3	1k	372k	70M	67M	7.9B	14B	2.0B	3.3B	1.3B	9.6B
p-hat1500-1	1.5k	285k	11M	83M	124M	1.6B	1.6B	16B	6.3B	7.2B
p-hat1500-2	1.5k	569k	91M	194M	7.9B	26B	6.5B	27B	14B	40B
p-hat1500-3	1.5k	847k	247M	224M	43B	72B	10B	16B	6.0B	48B
p-hat300-3	300	33k	1.9M	1.8M	63M	112M	17M	26M	10M	77M
p-hat500-1	500	32k	419k	3.0M	1.5M	19M	19M	200M	77M	88M
p-hat500-2	500	63k	3.3M	7.2M	93M	313M	80M	337M	167M	485M
p-hat500-3	500	94k	9.1M	8.3M	519M	881M	124M	192M	74M	588M
p-hat700-1	700	61k	1.1M	8.2M	5.4M	69M	71M	749M	291M	327M
p-hat700-2	700	122k	8.9M	19M	347M	1.2B	295M	1.3B	644M	1.8B
p-hat700-3	700	183k	25M	23M	1.9B	3.4B	479M	762M	296M	2.3B
san1000	1k	251k	29M	39M	2.1B	3.4B	647M	5.5B	1.1B	4.8B
san200-0-7-1	200	14k	467k	528k	9.0M	17M	3.4M	8.1M	1.1M	18M
san200-0-7-2	200	14k	485k	506k	10M	16M	2.9M	7.3M	1.9M	16M
san200-0-9-1	200	18k	960k	313k	35M	22M	1.2M	625k	87k	5.6M
san200-0-9-2	200	18k	958k	318k	34M	23M	1.3M	590k	119k	5.3M
san200-0-9-3	200	18k	957k	320k	34M	23M	1.3M	558k	185k	5.1M
san400-0-5-1	400	40k	1.7M	2.7M	46M	92M	20M	150M	34M	127M
san400-0-7-1	400	56k	3.8M	4.2M	152M	260M	52M	132M	15M	297M
san400-0-7-2	400	56k	3.8M	4.3M	145M	274M	55M	130M	20M	290M
san400-0-7-3	400	56k	3.7M	4.4M	139M	290M	58M	127M	28M	280M
san400-0-9-1	400	72k	7.7M	2.6M	560M	368M	20M	9.8M	1.3M	87M
sanr200-0-7	200	14k	444k	580k	7.4M	19M	4.2M	7.3M	2.4M	17M
sanr200-0-9	200	18k	950k	325k	34M	23M	1.3M	604k	202k	5.3M
sanr400-0-5	400	40k	1.3M	4.0M	17M	99M	49M	197M	66M	198M
sanr400-0-7	400	56k	3.6M	4.7M	124M	318M	68M	117M	39M	272M

Table 6.11.: Statistics of Biological D& D Networks

graph	$ V $	$ E $	$ g_{3_1} $	$ g_{3_2} $	$ g_{4_1} $	$ g_{4_2} $	$ g_{4_4} $	$ g_{4_6} $	$ g_{4_5} $	$ g_{4_3} $
DD1	327	899	749	2.2k	244	896	50	5.7k	695	3.1k
DD10	146	328	209	671	44	196	23	1.4k	155	752
DD100	349	1.0k	848	2.7k	260	1.1k	87	7.8k	1.2k	4.0k
DD1000	183	408	238	900	53	215	41	2.0k	311	926
DD1001	88	203	139	401	36	131	18	814	101	432
DD1002	104	255	194	567	58	228	10	1.3k	155	738
DD1003	53	116	75	227	22	68	16	409	48	238
DD1004	94	230	153	535	28	173	14	1.2k	144	637
DD1005	370	903	606	2.1k	142	622	81	4.7k	686	2.5k
DD1006	246	568	339	1.3k	66	337	55	3.0k	493	1.4k
DD1007	309	732	472	1.6k	109	485	56	3.7k	536	1.8k
DD1008	109	304	225	846	37	294	31	2.3k	326	1.2k
DD1009	129	272	148	568	26	138	20	1.1k	147	577
DD101	306	728	480	1.6k	113	508	66	3.6k	529	1.8k
DD1010	157	363	211	834	37	204	23	1.9k	289	850
DD1011	47	136	132	313	48	178	11	697	82	482
DD1012	146	365	279	770	73	297	30	1.6k	210	988
DD1013	93	211	133	462	27	139	18	992	153	509
DD1014	119	273	184	545	44	186	18	1.1k	106	594
DD1015	102	244	160	605	26	188	18	1.4k	227	797
DD1016	113	291	199	728	32	229	10	1.8k	234	954
DD1017	162	376	263	711	62	254	23	1.4k	128	803
DD1018	296	680	413	1.5k	71	406	42	3.1k	425	1.6k
DD1019	131	353	305	732	93	359	17	1.6k	114	948
DD102	561	1.4k	1.0k	3.4k	261	1.1k	108	8.2k	1.0k	4.1k
DD1020	228	541	375	1.2k	94	390	34	2.4k	335	1.3k
DD1021	329	787	490	1.9k	97	491	70	4.8k	808	2.2k
DD1022	294	730	510	1.7k	133	554	52	4.0k	545	2.0k
DD1023	172	425	282	1.1k	58	301	28	2.7k	409	1.3k
DD1024	59	160	139	333	43	152	8	774	65	434
DD1025	88	205	119	507	14	139	24	1.2k	197	567
DD1026	247	578	380	1.2k	94	378	48	2.6k	383	1.3k
DD1027	108	223	114	468	15	95	8	979	142	479
DD1028	72	137	53	296	4	42	11	605	118	233
DD1029	99	215	125	455	26	129	13	936	143	442
DD103	265	647	410	1.6k	97	443	58	3.9k	631	1.8k
DD1030	136	351	283	724	82	324	24	1.6k	127	881
DD1031	64	149	103	314	24	112	11	628	81	388
DD1032	159	387	256	968	55	290	33	2.5k	424	1.2k
DD1033	573	1.3k	748	2.9k	134	760	90	6.5k	939	3.0k
DD1034	183	514	426	1.3k	126	517	34	3.5k	429	1.9k
DD1035	56	122	68	245	7	68	8	454	52	240
DD1036	174	510	476	1.3k	171	619	32	3.2k	402	1.8k
DD1037	65	171	127	419	29	157	10	1.1k	141	535
DD1038	286	663	408	1.5k	88	431	49	3.4k	537	1.6k
DD1039	64	143	81	335	16	88	13	780	132	331
DD104	372	999	775	2.5k	229	932	66	6.8k	971	3.4k
DD1040	104	229	137	492	22	139	13	1.0k	140	547
DD1041	70	195	140	596	20	195	21	1.7k	291	902
DD1042	201	466	302	1.0k	73	306	40	2.2k	329	1.1k

Table 6.12.: Statistics of Chemical MUTAG Networks

graph	$ V $	$ E $	$ g_{3_1} $	$ g_{3_2} $	$ g_{4_1} $	$ g_{4_2} $	$ g_{4_4} $	$ g_{4_6} $	$ g_{4_5} $	$ g_{4_3} $
MUTAG1	23	27	0	41	0	0	0	63	10	0
MUTAG10	17	19	0	28	0	0	0	38	7	0
MUTAG100	20	23	0	35	0	0	0	53	9	0
MUTAG101	23	27	0	41	0	0	0	62	10	0
MUTAG102	19	21	0	31	0	0	0	43	8	0
MUTAG103	28	31	0	48	0	0	0	70	14	0
MUTAG104	26	30	0	46	0	0	0	70	12	0
MUTAG105	16	18	0	26	0	0	0	36	6	0
MUTAG106	23	27	0	41	0	0	0	61	10	0
MUTAG107	18	19	0	26	0	0	0	32	6	0
MUTAG108	17	19	0	27	0	0	0	36	6	0
MUTAG109	19	22	0	33	0	0	0	50	8	0
MUTAG11	15	17	0	25	0	0	0	36	6	0
MUTAG110	12	12	0	16	0	0	0	18	4	0
MUTAG111	25	28	0	43	0	0	0	64	12	0
MUTAG112	16	18	0	26	0	0	0	36	6	0
MUTAG113	23	27	0	41	0	0	0	62	10	0
MUTAG114	23	27	0	41	0	0	0	61	10	0
MUTAG115	16	18	0	26	0	0	0	36	6	0
MUTAG116	20	22	0	32	0	0	0	42	8	0
MUTAG117	23	27	0	41	0	0	0	63	10	0
MUTAG118	25	29	0	45	0	0	0	68	12	0
MUTAG119	19	22	0	33	0	0	0	48	8	0
MUTAG12	12	13	0	18	0	0	0	23	4	0
MUTAG120	23	27	0	41	0	0	0	63	10	0
MUTAG121	19	22	0	33	0	0	0	49	8	0
MUTAG122	19	20	0	29	0	0	0	38	8	0
MUTAG123	25	28	0	43	0	0	0	63	12	0
MUTAG124	18	20	0	29	0	0	0	40	7	0
MUTAG125	15	17	0	25	0	0	0	36	6	0
MUTAG126	16	17	0	23	0	0	0	28	5	0
MUTAG127	24	25	0	36	0	0	0	47	10	0
MUTAG128	11	11	0	14	0	0	0	16	3	0
MUTAG129	13	13	0	18	0	0	0	21	5	0
MUTAG13	21	22	0	31	0	0	0	39	8	0
MUTAG130	16	17	0	23	0	0	0	29	5	0
MUTAG131	10	10	0	13	0	0	0	14	3	0
MUTAG132	21	22	0	31	0	0	0	40	8	0
MUTAG133	13	14	0	20	0	0	0	26	5	0
MUTAG134	17	18	0	25	0	0	0	32	6	0
MUTAG135	11	11	0	15	0	0	0	17	4	0
MUTAG136	14	14	0	19	0	0	0	23	5	0
MUTAG137	11	11	0	14	0	0	0	17	3	0
MUTAG138	19	20	0	26	0	0	0	31	5	0
MUTAG139	11	11	0	14	0	0	0	17	3	0
MUTAG14	23	27	0	41	0	0	0	62	10	0
MUTAG140	21	23	0	35	0	0	0	48	10	0
MUTAG141	11	11	0	15	0	0	0	17	4	0
MUTAG142	13	14	0	20	0	0	0	25	5	0
MUTAG143	11	11	0	15	0	0	0	17	4	0

7. SUMMARY AND CONCLUSION

In this dissertation, we propose sampling as well as fast, efficient, and scalable methods for network analysis and mining for both static and streaming graphs. This dissertation is positioned to extend the range, applicability, and performance gains of network sampling as a tool that is not only useful for web crawling and data collection, but also for the analysis and mining of massive disk-resident and streaming graphs efficiently. We show how network sampling can be used as a mediator of various problem-specific constraints, such as the characteristics of the data (e.g., heavy-tailed data distribution), the data access constraints (e.g., streaming vs. distributed data), the available resources (e.g., memory, bandwidth), and the accuracy needs of queries.

Moreover, we propose *graph sample and hold*, a flexible framework for sampling from large streaming graphs. Using *graph sample and hold*, we formulated network sampling as a principled approach with two main functions: (1) the sampling function, and (2) the holding function. We showed how such approach allows tuning the sampling and estimation of graph properties more efficiently and accurately than the state-of-the-art. We developed a suite of algorithms, based on *graph sample and hold*, to sample and estimate various graph properties, while processing the graph sequentially as a stream of edges.

The potential benefits of the proposed framework are two-fold. First, it will lead to more interpretable sampling designs, that efficiently capture the specific graph properties of interest. This should benefit big graph analytics and data mining applications in general, since interpretability is a quality that is often important for domain experts to design useful sampling methods. Second, it will lead to samples with better quality that efficiently mirror the properties of the population graph.

In addition, we propose a fast, parallel, and efficient algorithm for motif counting that is significantly faster than the current methods used, while also scaling to much larger networks with millions of nodes and edges. The proposed motif counting algorithm leverages a number of proven combinatorial arguments, which enable us to obtain significant improvement on the scalability of motif counting. Thus, this brings new opportunities to investigate the use of motifs on much larger networks and newer applications.

Furthermore, we discuss how a number of important machine learning applications are likely to benefit from such algorithm, including graph-based anomaly detection [7, 46], entity resolution [47], as well as features for improving community detection [48], role discovery [49], and relational classification [50, 51].

For all the proposed methods, we implemented prototypes showing their applicability. Moreover, we conducted experimental studies on real-world data from social, biological, technological, and communication domains. We validate the efficiency and effectiveness of our approaches by large-scale comparisons against baseline methods.

7.1 Contributions

The algorithmic, theoretical, and empirical contributions of this work can be summarized as follows.

- Framework
 - A framework outlining the general problem of network sampling, highlighting the different goals, population and units, evaluation and classes of network sampling methods
 - Extending the above framework to include a spectrum of computational models within which to design network sampling methods (i. e., going from static to streaming graphs)
 - A generic framework called *graph sample and hold* for the design of stream sampling algorithms

- Theoretical Contributions
 - Analysis and proofs of the sampling bias in the streaming computational model, and investigation of its effect on the sampled subgraph
 - Development and proofs of unbiased estimators, variance estimators, upper and lower bounds for counts of frequent subgraphs
 - Analysis and proofs of a number of combinatorial arguments involving various motif patterns
 - Development and proofs of the relationship between pairs of motif patterns based on combinatorial arguments
- Algorithmic Contributions
 - A generic 2-pass streaming algorithm for sampling from large static graphs with linear runtime $O(|E|)$
 - A generic single-pass streaming algorithm for sampling a subgraph from streaming graphs with linear runtime $O(|E|)$
 - Extension of existing sampling algorithms to the streaming computational model
 - A generic single-pass streaming algorithm for unbiased estimation of subgraph counts in large networks
 - Development of parallel methods for computing the unbiased estimators and variance estimators of counts of frequent subgraphs efficiently
 - A fast, parallel, and efficient algorithm for motif counting, leveraging a number of combinatorial arguments to obtain significant scalability over current method (460 times faster than the state of the art).
- Empirical Contributions
 - Empirical comparison of proposed algorithms to existing sampling methods across a range of data sets from social, communication, and web graphs

- Illustration of the effect of different network sampling methods on the performance of relational classification
- Illustration of the effects of graph characteristics (e. g., graph sparsity) on the performance of different sampling methods
- Large-scale investigation and comparison of motif counting on 300+ networks with millions of nodes and edges
- Illustration of various applications of motif counting in biology and social network analysis – such as for large-scale graph classification, prediction, and anomaly detection

7.2 Future Directions

Generally, in data streams, the recent history is typically the more frequently analyzed, and also the more relevant for several applications. As the graph stream evolves overtime, which means new edges are added, some of the sampled edges may become outdated and less relevant over time. For example, an email exchanged between A and B today is more certain, relevant, and indicative of an existing relationship between A and B, than a previous email exchanged months ago. Therefore, a natural next step from this dissertation would be to extend the methods in Chapter 4 to bias the sample to the recent time-horizon. Moreover, since the structure of streaming graphs is continuously changing overtime, these graphs can be viewed as a dynamical system. The normal operation of any dynamical system can be described by a process that transitions between different states over time [190]. Therefore, another next step would be to extend the applicability of the methods in Chapter 4 to modeling and analyzing the state-space and evolution of graph streams.

Second, the work in Chapter 3 investigated the relationship between sampling and relational classification. Our work shows that biases may result from the size of the sample, the applied sampling method, or both. Our proposed work showed to simultaneously satisfy the two different goals for relational learning applications

(i. e., parameter estimation and accuracy evaluation). As a broad extension of this work, the relationship between sampling and machine learning is generally unexplored. Thus, a natural extension would be to propose sampling methods that can particularly perform well for certain machine learning tasks. For example, custom sampling methods for clustering [49], classification [53], anomaly detection [7], and graph partitioning [191]. Moreover, this work naturally extends itself to the area of privacy preserving data mining, where sampling can be used to answer queries and perform simulations, while maintaining the privacy of the user data.

Third, the work in Chapter 5 can be extended to different variations of adaptive techniques based on statistical sub-sampling and reservoir methods to limit the memory budget of our proposed framework. This means as the stored state reaches its boundary, new sampled edges would replace older ones to maintain the budget size. The risk of applying these sub-sampling techniques is the expected reduction in the estimation accuracy. Therefore, the trade-off between the sample size and the estimation accuracy has to be considered. At the same time, the major goal is to ensure that we can obtain the maximum benefit of this bounded budget to estimate the graph properties of interest.

Fourth, the algorithms in Chapter 6 can potentially accelerate a much broader class of machine learning tasks beyond prediction and classification. For example, interactive visual analytics [57, 192], graph-based anomaly detection [7, 46], entity resolution [47], as well as features for improving community detection [48], role discovery [49], and relational classification [50].

REFERENCES

REFERENCES

- [1] J. Lafferty, “Facebook users worldwide: 3.2 billion likes and comments per day,” http://allfacebook.com/facebook-marketing-infographic-engagement_b98277, August 2012.
- [2] J. Leskovec, L. A. Adamic, and B. A. Huberman, “The dynamics of viral marketing,” *ACM Transactions on the Web*, vol. 1, no. 1, p. 5, 2007.
- [3] A. Anderson, D. Huttenlocher, J. Kleinberg, and J. Leskovec, “Steering user behavior with badges,” in *Proceedings of the 22nd International Conference on World Wide Web*, 2013, pp. 95–106.
- [4] R. Pastor-Satorras and A. Vespignani, “Immunization of complex networks,” *Physical Review E*, vol. 65, no. 3, p. 036104, 2002.
- [5] T. Milenkoviæ and N. Pržulj, “Uncovering biological network function via graphlet degree signatures,” *Cancer Informatics*, vol. 6, p. 257, 2008.
- [6] A. Lakhina, M. Crovella, and C. Diot, “Diagnosing network-wide traffic anomalies,” in *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, 2004, pp. 219–230.
- [7] C. C. Noble and D. J. Cook, “Graph-based anomaly detection,” in *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 631–636.
- [8] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis, “Efficient semi-streaming algorithms for local triangle counting in massive graphs,” in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.
- [9] J. Leskovec and C. Faloutsos, “Sampling from large graphs,” in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 631–636.
- [10] A. S. Maiya and T. Y. Berger-Wolf, “Benefits of bias: Towards better characterization of network sampling,” in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2011, pp. 105–113.
- [11] N. K. Ahmed, J. Neville, and R. Kompella, “Network sampling: From static to streaming graphs,” *ACM Transactions on Knowledge Discovery From Data*, vol. 8, no. 2, pp. 1–56, 2014.
- [12] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, “On the evolution of user interaction in facebook,” in *Proceedings of the 2nd ACM Workshop on Online Social Networks*, 2009, pp. 37–42.

- [13] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl, “On the streaming model augmented with a sorting primitive,” in *Proceedings of the IEEE Symposium on Foundations of Computer Science*, 2004, pp. 540–549.
- [14] M. S. Cline, M. Smoot, E. Cerami, A. Kuchinsky, N. Landys, C. Workman, R. Christmas, I. Avila-Campilo, M. Creech, B. Gross *et al.*, “Integration of biological networks and gene expression data using cytoscape,” *Nature protocols*, vol. 2, no. 10, pp. 2366–2382, 2007.
- [15] S. Suri and S. Vassilvitskii, “Counting triangles and the curse of the last reducer,” in *Proceedings of the 20th International Conference on World Wide Web*, 2011, pp. 607–614.
- [16] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, “Doulion: Counting triangles in massive graphs with a coin,” in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009, pp. 837–846.
- [17] P. Zhao, C. C. Aggarwal, and M. Wang, “gsketch: On query estimation in graph streams,” *Proceedings of the VLDB Endowment*, vol. 5, no. 3, pp. 193–204, 2011.
- [18] T. Schank, “Algorithmic aspects of triangle-based network analysis,” *PhD in CS, University Karlsruhe*, 2007.
- [19] M. Jha, C. Seshadhri, and A. Pinar, “A space efficient streaming algorithm for triangle counting using the birthday paradox,” in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013, pp. 589–597.
- [20] C. Seshadhri, A. Pinar, and T. G. Kolda, “Fast triangle counting through wedge sampling,” in *Proc. of SIAM*, 2013.
- [21] T. G. Kolda, A. Pinar, T. Plantenga, C. Seshadhri, and C. Task, “Counting triangles in massive graphs with mapreduce,” *SIAM Journal on Scientific Computing*, 2013.
- [22] N. Duffield, C. Lund, and M. Thorup, “Estimating flow distributions from sampled flow statistics,” *Transactions on Networking*, vol. 13, no. 5, pp. 933–946, 2005.
- [23] N. K. Ahmed, N. Duffield, J. Neville, and R. Kompella, “Graph sample and hold: A framework for big-graph analytics,” in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 1446–1455.
- [24] C. Clifton, “Protecting against data mining through samples,” in *Research Advances in Database and Information Systems Security*. Springer, 2000, pp. 193–207.
- [25] O. Frank, “Sampling and inference in a population graph,” *International Statistical Review/Revue Internationale de Statistique*, vol. 48, no. 1, pp. 33–41, 1980.
- [26] M. Gjoka, M. Kurant, C. Butts, and A. Markopoulou, “Walking in facebook: A case study of unbiased sampling of OSNs,” in *IEEE International Conference on Computer Communications*, 2010, pp. 1–9.

- [27] C. Gkantsidis, M. Mihail, and A. Saberi, “Random walks in peer-to-peer networks,” in *IEEE International Conference on Computer Communications*, 2004, pp. 1–12.
- [28] A. S. Maiya and T. Y. Berger-Wolf, “Sampling Community Structure,” in *Proceedings of the 19th International Conference on World Wide Web*, 2010, pp. 701–710.
- [29] L. Backstrom and J. Kleinberg, “Network bucket testing,” in *Proceedings of the 20th International Conference on World Wide Web*, 2011, pp. 615–624.
- [30] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, “Measurement and analysis of online social networks,” in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, 2007, pp. 29–42.
- [31] A. Mislove, H. Koppula, K. Gummadi, P. Druschel, and B. Bhattacharjee, “Growth of the flickr social network,” in *Proceedings of the 2nd ACM Workshop on Online Social Networks*, 2008, pp. 25–30.
- [32] B. Ribeiro and D. Towsley, “Estimating and sampling graphs with multidimensional random walks,” in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, 2010, pp. 390–403.
- [33] C. C. Aggarwal and H. Wang, *Managing and Mining Graph Data*. Springer, 2010, vol. 40.
- [34] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. Cambridge University Press, 2012.
- [35] S. Muthukrishnan, *Data streams: Algorithms and applications*. Now Publishers Inc., 2005.
- [36] C. Aggarwal, J. Han, J. Wang, and P. Yu, “A framework for clustering evolving data streams,” in *Proceedings of the 29th International Conference on Very Large Data Bases*, 2003, pp. 81–92.
- [37] G. Cormode and N. Duffield, “Sampling for big data: A tutorial,” in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.
- [38] J. S. Vitter, “Random sampling with a reservoir,” *ACM Transactions on Mathematical Software*, vol. 11, pp. 37–57, 1985.
- [39] D. R. Karger, “Random sampling in cut, flow, and network design problems,” in *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, 1994, pp. 648–657.
- [40] C. Aggarwal, Y. Zhao, and P. Yu, “Outlier detection in graph streams,” in *IEEE 27th International Conference on Data Engineering*, 2011, pp. 399–409.
- [41] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, “On graph problems in a semi-streaming model,” *Automata, Languages and Programming*, pp. 17–44, 2004.

- [42] A. Pavan, K. Tangwongsan, S. Tirthapura, and K.-L. Wu, “Counting and sampling triangles from a graph stream,” *Proceedings of the VLDB Endowment*, vol. 6, no. 14, pp. 1870–1881, 2013.
- [43] L. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler, “Counting triangles in data streams,” in *Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2006, pp. 253–262.
- [44] H. Jowhari and M. Ghodsi, “New streaming algorithms for counting triangles in graphs,” in *Computing and Combinatorics*, 2005, pp. 710–716.
- [45] D. G. Horvitz and D. J. Thompson, “A generalization of sampling without replacement from a finite universe,” *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 663–685, 1952.
- [46] L. Akoglu, H. Tong, and D. Koutra, “Graph based anomaly detection and description: A survey,” *Data Mining and Knowledge Discovery*, pp. 1–63, 2014.
- [47] I. Bhattacharya and L. Getoor, “Entity resolution in graphs,” *Mining graph data*, p. 311, 2006.
- [48] S. E. Schaeffer, “Graph clustering,” *Computer Science Review*, vol. 1, no. 1, 2007.
- [49] R. A. Rossi and N. K. Ahmed, “Role discovery in networks,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 4, pp. 1112–1131, 2015.
- [50] L. Getoor and B. Taskar, *Introduction to Statistical Relational Learning*. MIT press, 2007.
- [51] J. Neville, B. Gallagher, and T. Eliassi-Rad, “Evaluating statistical tests for within-network classifiers of relational data,” in *IEEE 9th International Conference on Data Mining*, 2009, pp. 397–406.
- [52] N. K. Ahmed, J. Neville, and R. Kompella, “Reconsidering the foundations of network sampling,” in *Proceedings of the 2nd Workshop on Information in Networks*, 2010.
- [53] ———, “Network sampling designs for relational classification,” in *Proceedings of the International AAAI Conference on Weblogs and Social Media*, 2012, pp. 1–4.
- [54] N. K. Ahmed, F. Berchmans, J. Neville, and R. Kompella, “Time-based sampling of social network activity graphs,” in *Proceedings of the 8th Workshop on Mining and Learning with Graphs*, 2010, pp. 1–9.
- [55] N. K. Ahmed, J. Neville, and R. Kompella, “Space-efficient sampling from social activity streams,” in *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, 2012, pp. 53–60.
- [56] N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield, “Fast parallel graphlet counting for large networks,” *arXiv:1506.04322*, 2015.

- [57] N. K. Ahmed and R. A. Rossi, “Interactive visual graph analytics on the web,” in *Proceedings of the Ninth International AAAI Conference on Web and Social Media*, 2015.
- [58] J. Watters and P. Biernacki, “Targeted sampling: Options for the study of hidden populations,” *Social Problems*, vol. 36, no. 4, pp. 416–430, 1989.
- [59] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer, “Learning probabilistic relational models,” in *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1999, pp. 1300–1309.
- [60] R. Rossi and J. Neville, “Time-evolving relational classification and ensemble methods,” in *Advances in Knowledge Discovery and Data Mining*. Springer, 2012, vol. 7301, pp. 1–13.
- [61] R. A. Rossi, L. K. McDowell, D. W. Aha, and J. Neville, “Transforming graph data for statistical relational learning,” *Journal of Artificial Intelligence Research*, vol. 45, no. 1, pp. 363–441, 2012.
- [62] E. Bakshy, I. Rosenn, C. Marlow, and L. Adamic, “The role of social networks in information diffusion,” in *Proceedings of the 21st International Conference on World Wide Web*, 2012, pp. 519–528.
- [63] M. Hansen and W. Hurwitz, “On the theory of sampling from finite populations,” *The Annals of Mathematical Statistics*, vol. 14, no. 4, pp. 333–362, 1943.
- [64] M. Stumpf, C. Wiuf, and R. May, “Subnets of scale-free networks are not scale-free: Sampling properties of networks,” *Proceedings of the National Academy of Sciences*, vol. 102, no. 12, pp. 4221–4224, 2005.
- [65] S. Lee, P. Kim, and H. Jeong, “Statistical properties of sampled networks,” *Physical Review E*, vol. 73, p. 016102, 2006.
- [66] D. Heckathorn, “Respondent-driven sampling: A new approach to the study of hidden populations,” *Social Problems*, no. 2, pp. 174–199, 1997.
- [67] J. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, “The web as a graph: Measurements, models, and methods,” in *Computing and Combinatorics*, ser. Lecture Notes in Computer Science. Springer, 1999, vol. 1627, pp. 1–17.
- [68] S. Redner, “How popular is your paper? an empirical study of the citation distribution,” *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 4, no. 2, pp. 131–134, 1998.
- [69] D. Chakrabarti, Y. Zhan, and C. Faloutsos, “R-mat: A recursive model for graph mining,” in *SIAM International Conference on Data Mining*, 2004, pp. 442–446.
- [70] D. J. Watts and S. H. Strogatz, “Collective dynamics of small-world networks,” *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.
- [71] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir, “A model of internet topology using k-shell decomposition,” *Proceedings of the National Academy of Sciences*, vol. 104, no. 27, pp. 11 150–11 154, 2007.

- [72] J. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani, "K-core decomposition of internet graphs: Hierarchies, self-similarity and measurement biases," *arXiv cs/0511007*, 2005.
- [73] R. Kumar, J. Novak, and A. Tomkins, "Structure and evolution of online social networks," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 337–357.
- [74] C. Seshadhri, A. Pinar, and T. G. Kolda, "An in-depth analysis of stochastic kronecker graphs," *Journal of the ACM*, vol. 60, no. 2, pp. 1–32, 2013.
- [75] M. Newman, *Networks: An Introduction*. Oxford University Press, 2010.
- [76] L. Lee, "On the effectiveness of the skew divergence for statistical language analysis," in *Artificial Intelligence and Statistics*, 2001, pp. 65–72.
- [77] J. Zhang, *Managing and Mining Graph Data*. Springer, 2010, chapter 13: A survey on streaming algorithms for massive graphs, pp. 393–420.
- [78] M. Gjoka, M. Kurant, C. Butts, and A. Markopoulou, "Practical recommendations on crawling online social networks," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1872–1892, 2011.
- [79] O. Frank, "Survey sampling in graphs," *Journal of Statistical Planning and Inference*, vol. 1, no. 3, pp. 235–264, 1977.
- [80] —, "A survey of statistical methods for graph analysis," *Sociological Methodology*, vol. 12, pp. 110–155, 1981.
- [81] L. Goodman, "Snowball sampling," *The Annals of Mathematical Statistics*, vol. 32, no. 1, pp. 148–170, 1961.
- [82] M. Granovetter, "Network sampling: Some first steps," *American Journal of Sociology*, vol. 81, no. 6, pp. 1287–1303, 1976.
- [83] K. Gile and M. Handcock, "Respondent-driven sampling: An assessment of current methodology," *Sociological Methodology*, vol. 40, no. 1, pp. 285–327, 2010.
- [84] E. Kolaczyk, *Statistical Analysis of Network Data*. Springer, 2009, chapter 5 Sampling and Estimation in Network Graphs, pp. 123–152.
- [85] S. Yoon, S. Lee, S.-H. Yook, and Y. Kim, "Statistical properties of sampled networks by random walks," *Physical Review E*, vol. 75, p. 046114, 2007.
- [86] A. Lakhina, J. Byers, M. Crovella, and P. Xie, "Sampling biases in ip topology measurements," in *IEEE International Conference on Computer Communications*, 2003, pp. 332–341.
- [87] Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong, "Analysis of topological characteristics of huge online social networking services," in *Proceedings of the 16th International Conference on World Wide Web*, 2007, pp. 835–844.
- [88] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao, "User interactions in social networks and their implications," in *Proceedings of the 4th ACM European Conference on Computer Systems*, 2009, pp. 205–218.

- [89] S. Ye, J. Lang, and F. Wu, "Crawling online social graphs," in *IEEE 12th International Asia-Pacific Web Conference*, 2010, pp. 236–242.
- [90] M. Kurant, A. Markopoulou, and P. Thiran, "Towards unbiased bfs sampling," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1799–1809, 2011.
- [91] E. Baykan, M. Henzinger, S. Keller, S. De Castelberg, and M. Kinzler, "A comparison of techniques for sampling web pages," *arXiv:0902.1604*, 2009.
- [92] M. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork, "On near-uniform url sampling," *Computer Networks*, vol. 33, no. 1, pp. 295–308, 2000.
- [93] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger, "On unbiased sampling for unstructured peer-to-peer networks," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, 2006, pp. 27–40.
- [94] A. Rasti, M. Torkjazi, R. Rejaie, N. Duffield, W. Willinger, and D. Stutzbach, "Respondent-driven sampling for characterizing unstructured overlays," in *IEEE International Conference on Computer Communications*, 2009, pp. 2701–2705.
- [95] K. Avrachenkov, B. Ribeiro, and D. Towsley, "Improving random walk estimation accuracy with uniform restarts," in *Algorithms and Models for the Web-Graph*, ser. Lecture Notes in Computer Science. Springer, 2010, vol. 6516, pp. 98–109.
- [96] C. Hubler, H.-P. Kriegel, K. M. Borgwardt, and Z. Ghahramani, "Metropolis algorithms for representative subgraph sampling," in *IEEE 8th International Conference on Data Mining*, 2008, pp. 283–292.
- [97] V. Krishnamurthy, M. Faloutsos, M. Chrobak, J. Cui, L. Lao, and A. Percus, "Sampling large Internet topologies for simulation purposes," *Computer Networks*, vol. 51, no. 15, pp. 4284–4302, 2007.
- [98] A. Vattani, D. Chakrabarti, and M. Gurevich, "Preserving personalized pagerank in subgraphs," in *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 793–800.
- [99] Y. Jia, J. Hoberock, M. Garland, and J. Hart, "On the visualization of social and other scale-free networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1285–1292, 2008.
- [100] X. Lu and S. Bressan, "Sampling connected induced subgraphs uniformly at random," in *Scientific and Statistical Database Management*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7338, pp. 195–212.
- [101] A. Dasgupta, R. Kumar, and D. Sivakumar, "Social sampling," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2012, pp. 235–243.
- [102] M. Al Hasan and M. Zaki, "Output space sampling for graph patterns," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 730–741, 2009.

- [103] M. A. Bhuiyan, M. Rahman, M. Rahman, and M. Al Hasan, “Guise: Uniform sampling of graphlets for large graph analysis,” in *IEEE 12th International Conference on Data Mining*, 2012, pp. 91–100.
- [104] M. Papagelis, G. Das, and N. Koudas, “Sampling online social networks,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 3, pp. 662–676, 2013.
- [105] M. De Choudhury, Y. Lin, H. Sundaram, K. Candan, L. Xie, and A. Kelliher, “How does the data sampling strategy impact the discovery of information diffusion in social media,” in *Proceedings of the International AAAI Conference on Weblogs and Social Media*, 2010, pp. 34–41.
- [106] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, “Models and issues in data stream systems,” in *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2002, pp. 1–16.
- [107] L. Golab and M. Özsu, “Issues in data stream management,” *ACM Sigmod Record*, vol. 32, no. 2, pp. 5–14, 2003.
- [108] C. C. Aggarwal, Ed., *Data Streams - Models and Algorithms*, ser. Advances in Database Systems. Springer, 2007, vol. 31.
- [109] B. Babcock, M. Datar, and R. Motwani, “Sampling from a moving window over streaming data,” in *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002, pp. 633–634.
- [110] C. C. Aggarwal, “On Biased Reservoir Sampling in the Presence of Stream Evolution.” in *Proceedings of the 32nd International Conference on Very Large Data Bases*, 2006, pp. 607–618.
- [111] G. S. Manku and R. Motwani, “Approximate Frequency Counts over Data Streams.” in *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002, pp. 346–357.
- [112] M. Charikar, K. Chen, and M. Farach-Colton, “Finding frequent items in data streams,” in *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, 2002, pp. 693–703.
- [113] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker, “Load shedding in a data stream manager,” in *Proceedings of the 29th International Conference on Very Large Data Bases*, 2003, pp. 309–320.
- [114] H. Wang, W. Fan, P. Yu, and J. Han, “Mining concept-drifting data streams using ensemble classifiers,” in *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 226–235.
- [115] J. Gao, W. Fan, J. Han, and P. Yu, “A general framework for mining concept-drifting data streams with skewed distributions,” in *SIAM International Conference on Data Mining*, 2007, pp. 3–14.
- [116] W. Fan, “Systematic data selection to mine concept-drifting data streams,” in *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp. 128–137.

- [117] —, “StreamMiner: A classifier ensemble-based engine to mine concept-drifting data streams,” in *Proceedings of the 30th International Conference on Very Large Data Bases*, 2004, pp. 1257–1260.
- [118] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O’Callaghan, “Clustering data streams: Theory and practice,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 515–528, 2003.
- [119] W. Fan, Y. Huang, H. Wang, and P. Yu, “Active mining of data streams,” in *SIAM International Conference on Data Mining*, 2004, pp. 457–461.
- [120] X. Li, P. Yu, B. Liu, and S. Ng, “Positive unlabeled learning for data stream classification,” in *SIAM International Conference on Data Mining*, 2009, pp. 259–270.
- [121] P. Domingos and G. Hulten, “Mining high-speed data streams,” in *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000, pp. 71–80.
- [122] G. Hulten, L. Spencer, and P. Domingos, “Mining time-changing data streams,” in *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001, pp. 97–106.
- [123] M. Gaber, A. Zaslavsky, and S. Krishnaswamy, “Mining data streams: A review,” *ACM Sigmod Record*, vol. 34, no. 2, pp. 18–26, 2005.
- [124] H. Wang, P. Yu, and J. Han, *Data Mining and Knowledge Discovery Handbook*. Springer, 2010, ch. Mining Concept-Drifting Data Streams, pp. 789–802.
- [125] M. Henzinger, P. Raghavan, and S. Rajagopalan, “Computing on data streams,” in *External Memory Algorithms: Dimacs Workshop External Memory and Visualization*, vol. 50, 1999, pp. 107–118.
- [126] Z. Bar-Yossef, R. Kumar, and D. Sivakumar, “Reductions in streaming algorithms with an application to counting triangles in graphs,” in *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002, pp. 623–632.
- [127] A. Buchsbaum, R. Giancarlo, and J. Westbrook, “On finding common neighborhoods in massive graphs,” *Theoretical Computer Science*, vol. 299, no. 1, pp. 707–718, 2003.
- [128] A. D. Sarma, S. Gollapudi, and R. Panigrahy, “Estimating pagerank on graph streams,” in *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2008, pp. 69–78.
- [129] G. Cormode and S. Muthukrishnan, “Space efficient mining of multigraph streams,” in *Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2005, pp. 271–282.
- [130] C. Aggarwal, Y. Zhao, and P. Yu, “On clustering graph streams,” in *SIAM International Conference on Data Mining*, 2010, pp. 478–489.
- [131] L. Chen and C. Wang, “Continuous subgraph pattern search over certain and uncertain graph streams,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 8, pp. 1093–1109, 2010.

- [132] C. Aggarwal, Y. Li, P. Yu, and R. Jin, "On dense pattern mining in graph streams," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 975–984, 2010.
- [133] A. McGregor, "Graph mining on streams," *Encyclopedia of Database Systems*, 2009.
- [134] M. Newman, "Assortative mixing in networks," *Physical Review Letters*, vol. 89, no. 20, p. 208701, 2002.
- [135] D. F. Gleich, "Graph of flickr photo-sharing social network crawled in may 2006," Feb 2012. [Online]. Available: <https://research.hub.purdue.edu/publications/1002>
- [136] J. Leskovec, "Stanford large network dataset collection," 2014. [Online]. Available: <http://snap.stanford.edu/data/index.html>
- [137] W. Haemers, "Interlacing eigenvalues and graphs," *Linear Algebra and its Applications*, vol. 226, pp. 593–616, 1995.
- [138] N. K. Ahmed, J. Neville, and R. R. Kompella, "Network sampling via edge-based node selection with graph induction," Purdue Digital Library, Tech. Rep. 11-016, 2011.
- [139] C. Körner and S. Wrobel, "Bias-free hypothesis evaluation in multirelational domains," in *Proceedings of the 4th International Workshop on Multi-relational Mining*, 2005, pp. 33–38.
- [140] S. Macskassy and F. Provost, "Classification in networked data: A toolkit and a univariate case study," *Journal of Machine Learning Research*, vol. 8, pp. 935–983, 2007.
- [141] B. Taskar, E. Segal, and D. Koller, "Probabilistic classification and clustering in relational data," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, 2001, pp. 870–876.
- [142] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.
- [143] R. Xiang, J. Neville, and M. Rogati, "Modeling relationship strength in online social networks," in *Proceedings of the 19th International Conference on World Wide Web*, 2010, pp. 981–990.
- [144] L. Adamic and N. Glance, "The political blogosphere and the 2004 us election: Divided they blog," in *Proceedings of the 3rd International Workshop on Link Discovery*, 2005, pp. 36–43.
- [145] A. Gautreau, A. Barrat, and M. Barthlemy, "Microdynamics in stationary complex networks," *Proceedings of the National Academy of Sciences*, vol. 106, no. 22, pp. 8847–8852, 2009.
- [146] M. Conover, J. Ratkiewicz, M. Francisco, B. Gonçalves, A. Flammini, and F. Menczer, "Political polarization on twitter," in *Proceedings of the International AAAI Conference on Weblogs and Social Media*, 2011, pp. 89–96.

- [147] L. Isella, J. Stehl, A. Barrat, C. Cattuto, J. Pinton, and W. Van den Broeck, “What’s in a crowd? analysis of face-to-face behavioral networks,” *Journal of Theoretical Biology*, vol. 271, no. 1, pp. 166–180, 2011.
- [148] C. Estan and G. Varghese, “New directions in traffic measurement and accounting,” in *Proceedings of the 2nd ACM SIGCOMM Conference on Internet Measurement*, 2002, pp. 323–336.
- [149] P. B. Gibbons and Y. Matias, “New sampling-based summary statistics for improving approximate query answers,” in *ACM SIGMOD Record*, vol. 27, no. 2, 1998, pp. 331–342.
- [150] Smitha, I. Kim, and A. Reddy, “Identifying long-term high-bandwidth flows at a router,” in *Proceedings of High Performance Computing*. Springer, 2001, pp. 361–371.
- [151] D. Williams, *Probability with Martingales*. Cambridge University Press, 1991.
- [152] M. J. Schervish, *Theory of Statistics*. Springer, 1995.
- [153] A. L. Traud, P. J. Mucha, and M. A. Porter, “Social structure of facebook networks,” *Physica A: Statistical Mechanics and its Applications*, 2012.
- [154] E. Cohen, G. Cormode, and N. Duffield, “Don’t let the negatives bring you down: Sampling from streams of signed updates,” *SIGMETRICS*, vol. 40, no. 1, pp. 343–354, 2012.
- [155] E. Cohen, N. Duffield, H. Kaplan, C. Lund, and M. Thorup, “Algorithms and estimators for accurate summarization of internet traffic,” in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, 2007, pp. 265–278.
- [156] O. Frank, “Sampling and estimation in large social networks,” *Social Networks*, vol. 1, no. 1, pp. 91–101, 1979.
- [157] E. D. Kolaczyk, “Sampling and estimation in network graphs,” in *Statistical Analysis of Network Data*. Springer, 2009, pp. 1–30.
- [158] C. Seshadhri, A. Pinar, and T. G. Kolda, “Wedge sampling for computing clustering coefficients and triangle counts on large graphs,” *arXiv:1309.3321*, 2013.
- [159] R. A. Rossi, D. F. Gleich, A. H. Gebremedhin, and M. A. Patwary, “Fast maximum clique algorithms for large graphs,” in *Proc. of WWW*, 2014.
- [160] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, “Network motifs: Simple building blocks of complex networks,” *Science*, vol. 298, no. 5594, pp. 824–827, 2002.
- [161] N. Pržulj, D. G. Corneil, and I. Jurisica, “Modeling interactome: Scale-free or geometric?” *Bioinformatics*, vol. 20, no. 18, pp. 3508–3515, 2004.
- [162] N. Shervashidze, T. Petri, K. Mehlhorn, K. M. Borgwardt, and S. Vishwanathan, “Efficient graphlet kernels for large graph comparison,” in *International Conference on Artificial Intelligence and Statistics*, 2009, pp. 488–495.

- [163] P. W. Holland and S. Leinhardt, “Local structure in social networks,” *Sociological Methodology*, vol. 7, pp. pp. 1–45, 1976.
- [164] K. Faust, “A puzzle concerning triads in social networks: Graph constraints and the triad census,” *Social Networks*, vol. 32, no. 3, pp. 221–233, 2010.
- [165] O. Frank, “Triad count statistics,” *Annals of Discrete Mathematics*, vol. 38, pp. 141–149, 1988.
- [166] G. Simmel and K. H. Wolff, *The Sociology of Georg Simmel*. Simon and Schuster, 1950.
- [167] M. Granovetter, “The strength of weak ties: A network theory revisited,” *Sociological Theory*, vol. 1, no. 1, pp. 201–233, 1983.
- [168] T. Milenković, W. L. Ng, W. Hayes, and N. Pržulj, “Optimal network alignment with graphlet degree vectors,” *Cancer Informatics*, vol. 9, p. 121, 2010.
- [169] O. Kuchaiev, T. Milenković, V. Memišević, W. Hayes, and N. Pržulj, “Topological network alignment uncovers biological function and phylogeny,” *Journal of the Royal Society Interface*, vol. 7, no. 50, pp. 1341–1354, 2010.
- [170] D. Feldman and Y. Shavitt, “Automatic large scale generation of internet pop level maps,” in *Proceedings of the IEEE Global Communications Conference*, 2008, pp. 1–6.
- [171] D. Hales and S. Arteconi, “Motifs in evolving cooperative networks look like protein structure networks,” *Journal of Networks and Heterogeneous Media*, vol. 3, no. 2, 2008.
- [172] L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi, “Graph kernels for chemical informatics,” *Neural Networks*, vol. 18, no. 8, pp. 1093–1110, 2005.
- [173] H. Kashima, H. Saigo, M. Hattori, and K. Tsuda, “Graph kernels for chemoinformatics,” *Chemoinformatics and Advanced Machine Learning Perspectives: Complex Computational Methods and Collaborative Techniques: Complex Computational Methods and Collaborative Techniques*, p. 1, 2010.
- [174] L. Zhang, M. Song, Z. Liu, X. Liu, J. Bu, and C. Chen, “Probabilistic graphlet cut: Exploiting spatial structure cue for weakly supervised image segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1908–1915.
- [175] L. Zhang, Y. Han, Y. Yang, M. Song, S. Yan, and Q. Tian, “Discovering discriminative graphlets for aerial image categories recognition,” *IEEE Transactions on Image Processing*, vol. 22, no. 12, pp. 5071–5084, 2013.
- [176] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, “Graph kernels,” *Journal of Machine Learning Research*, vol. 11, pp. 1201–1242, 2010.
- [177] M. Gonen and Y. Shavitt, “Approximating the number of network motifs,” *Internet Mathematics*, vol. 6, no. 3, pp. 349–372, 2009.

- [178] R. A. Rossi and N. K. Ahmed, “The network data repository with interactive graph analytics and visualization,” in *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015. [Online]. Available: <http://networkrepository.com>
- [179] D. Marcus and Y. Shavitt, “Rage—A rapid graphlet enumerator for large networks,” *Computer Networks*, vol. 56, no. 2, pp. 810–819, 2012.
- [180] S. Wernicke and F. Rasche, “Fanmod: A tool for fast network motif detection,” *Bioinformatics*, vol. 22, no. 9, pp. 1152–1153, 2006.
- [181] T. Hočevár and J. Demšar, “A combinatorial approach to graphlet counting,” *Bioinformatics*, vol. 30, no. 4, pp. 559–565, 2014.
- [182] W. Hayes, K. Sun, and N. Pržulj, “Graphlet-based measures are suitable for biological network comparison,” *Bioinformatics*, vol. 29, no. 4, pp. 483–491, 2013.
- [183] J. L. Gross, J. Yellen, and P. Zhang, *Handbook of Graph Theory*, 2nd ed. Chapman & Hall/CRC, 2013.
- [184] J. Ugander, L. Backstrom, and J. Kleinberg, “Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections,” in *Proceedings of the 22nd International Conference on World Wide Web*, 2013, pp. 1307–1318.
- [185] B. D. McKay, “Small graphs are reconstructible,” *Australasian Journal of Combinatorics*, vol. 15, pp. 123–126, 1997.
- [186] P. J. Kelly, “A congruence theorem for trees.” *Pacific Journal of Mathematics*, vol. 7, no. 1, pp. 961–968, 1957.
- [187] B. Manvel and P. K. Stockmeyer, “On reconstruction of matrices,” *Mathematics Magazine*, pp. 218–221, 1971.
- [188] R. P. Stanley, *What Is Enumerative Combinatorics?* Springer, 1986.
- [189] K.-I. Goh, M. E. Cusick, D. Valle, B. Childs, M. Vidal, and A.-L. Barabási, “The human disease network,” *Proceedings of the National Academy of Sciences*, vol. 104, no. 21, pp. 8685–8690, 2007.
- [190] N. K. Ahmed, C. Cole, and J. Neville, “Learning the latent state space of time-varying graphs,” *arXiv:1403.3707*, 2014.
- [191] R. A. Rossi and N. K. Ahmed, “Coloring large complex networks,” *Social Network Analysis and Mining*, vol. 4, no. 1, pp. 1–37, 2014.
- [192] R. Rossi and N. K. Ahmed, “Interactive data repositories: From data sharing to interactive data exploration & visualization,” in *Proceedings of the ACM SIGKDD Workshop on Interactive Data Exploration and Analytics*, 2015, pp. 1–5.

VITA

VITA

Nesreen Ahmed was born and raised in Cairo, Egypt. She received her Ph.D. from the Department of Computer Science at Purdue University. Her research in large-scale data mining and machine learning focuses on the design of efficient and scalable techniques for the analysis and modeling of network, social media, and time-series data. Nesreen earned her joint master's degree in statistics and computer science from Purdue University. She also earned another master's degree and her bachelor of science from Cairo University. She has published numerous papers at conferences, journals, and two patents. She has also given tutorials on network sampling at top data mining conferences. Nesreen worked as a researcher at Facebook, Intel data analytics, Adobe Advanced Technology labs, and the data mining and computer modeling Center of Excellence in Cairo. She was selected by the university of California Berkeley as a rising star, awarded to top Ph.D. candidates and postdocs. Nesreen is an author of the web-based network repository project (<http://networkrepository.com>), the first data repository with interactive visual graph analytics.